Table of Contents

1 What's New

25-Jan-24 New Features in SecureBridge 10.4.1

- Added support for RAD Studio 12 Athens
- Added support for macOS Sonoma
- Added support for iOS 17
- Added support for Android 13
- Added support for Lazarus 3.0 and FPC 3.2.2
- Added support for ChaCha20-Poly1305 cipher suites for TLS protocol
- Added support for simultaneous usage of public key and keyboard-interactive authentication on connecting to SSH server
- Added support for OAUTH2 authentication in TScSMTPClient
- Added support for DIGEST authentication in TScHttpWebRequest
- Added support for TOTPToken generation
- Added encoding EC private key to OpenSSH format
- Added the ability to use an SSH connection within the SSL protocol
- Added HTTPS server demo

05-Apr-23 New Features in SecureBridge 10.3

- Added support for RAD Studio 11 Alexandria Release 3
- Added support for iOS Simulator ARM 64-bit target platform
- Added support for Lazarus 2.2.6
- Added support the keDHGroup16Sha512, keDHGroup18Sha512, and keDHGroup14Sha256 key exchange algorithms for using in SSH protocol
- Added support the hmac-sha2-256-etm@openssh.com and hmac-sha2-512-etm@openssh.com HMAC algorithms for using in SSH protocol
- Added support for the PuTTY-User-Key-File-3 key format

12-Dec-22 New Features in SecureBridge 10.2

- Added support for RAD Studio 11 Alexandria Release 2
- Added support for iOS Simulator ARM 64-bit target platform
- Added support for Lazarus 2.2.4
- Added support for iOS 15
- Added support for Android 12

04-Jul-22 New Features in SecureBridge 10.1

- Added support for RAD Studio 11.1 Alexandria
- Added support for Lazarus 2.2.0
- Added support for Windows 11
- Added support for macOS Monterey
- Added support for the HTTP/HTTPS server in the TScHttpServer component
- Added the TScSSLServer component
- Added the TScHttpWebRequest.WriteAsStringParams method for writing encoded post data

- Added BeforeSendAttachment and AfterSendAttachment events in the TScSMTPClient component
- Added the TScSMTPClient.OnProgress event
- Added import of private key assigned with a certificate from CryptoAPI storage

17-Sep-21 New Features in SecureBridge 10.0

- RAD Studio 11 Alexandria is supported
- macOS ARM is supported
- Added the anonymous Diffie-Hellman cipher suites in the TLS protocol
- Added the TScSSHCustomChannel.ExitStatusCode property for getting the command status code
- Added the TScHubConnection.InvokeTimeout property

30-Mar-21 New Features in SecureBridge 9.5

- RAD Studio 10.4.2 Sydney is supported
- macOS 11 Big Sur is supported
- iOS 14 is supported
- Android 11 is supported
- Lazarus 2.0.12 is supported
- Support for the TLS/SSL server in the TScSSLServerConnection component is added
- Support for the TCP/IP server in the TScTCPServer component is added
- The BeforeReadData and BeforeWriteDataEx events in the TScSFTPClient component are added
- The ability to interrupt instance of a connection in TScSSHServer is added

11-Nov-20 New Features in SecureBridge 9.4

- Support for the SMTP protocol by the TScSMTPClient component is added
- Lazarus 2.0.10 and FPC 3.2.0 are supported
- The BeforeConnect, AfterConnect, BeforeDisconnect, and AfterDisconnect events in the TScFTPClient component are added
- The OnSendCommand and OnReadReply events for logging incoming and outgoing data in the TScFTPClient component are added
- The TScFTPClient.CEncoding and TScFTPClient.UseUTF8 properties are added

26-Jun-20 New Features in SecureBridge 9.3

- RAD Studio 10.4 is suppported
- Lazarus 2.0.8 is supported
- macOS 64-bit in Lazarus is supported
- Support for the Socks4 and Socks5 protocols is added
- Support for importing from PKCS #12 format in the TScPKCS12Processor class is added
- Support for the Signed Certificate Timestamp (SCT) certificate extension is added
- Support for SSH dynamic port forwarding in TScSSHChannel is added
- The TScHttpWebResponse.OnProgress event is added

26-Dec-19 New Features in SecureBridge 9.2

- Android 64-bit is supported
- Lazarus 2.0.6 is supported
- The TScSFTPClient.ReadDirectoryToList method to retrieve a directory listing is added
- Support for keyboard-interactive authentication in TScSSHServer is added
- Server certificate validation in the TLS/SSL protocol on Android is improved

05-Sep-19 New Features in SecureBridge 9.1

- macOS 64-bit is supported
- Release 2 for RAD Studio 10.3 Rio, Delphi 10.3 Rio, and C++Builder 10.3 Rio is now required
- Lazarus 2.0.4 is supported
- Support for the SignalR protocol version 2.2 is added
- The TScHubConnection component to support the SignalR client is added
- Support for the x25519 algorithm for the TLS/SSL protocol is added
- Support for Certificate Revocation List (CRL) is added
- Server certificate validation in the TLS/SSL protocol is improved
- The chunked transfer encoding for sending out data via HTTP/HTTPS is added
- The TScHttpWebRequest.BeforeSendData event is added

14-Dec-18 New Features in SecureBridge 9.0

- RAD Studio 10.3 Rio is supported
- Lazarus 1.8.4 is supported
- Support for the TLS 1.3 protocol is added
- Support for the WebSocket protocol by the TScWebSocketClient component is added
- Support for Elliptic-Curve keys and certificates is added
- Support for ECDSA algorithm for TLS/SSL protocol is added
- Support for GCM encryption mode is added
- Security for SSL client is improved
- The TScUser.SSHChannelPermissions property to manage available user permissions is added
- Interface for the OnData, BeforeWriteData, and AfterWriteData events in the TScSFTPClient component is changed

24-Apr-18 New Features in SecureBridge 8.2

- Lazarus 1.8 and FPC 3.0.4 are supported
- Support for the FTP and FTPS protocols is added
- The TScFTPClient component to support access to remote files using FTP protocol is added
- Possibility to connect to a remote host through a proxy server is added
- Now TScStorage is thread-safe
- TScSSHConnectionInfo.LocalSockAddr to retrieve a local IP address is added
- The TScSSHServerOptions.MaxConnections property to limit the number of active connections is added
- The TScSSHServer.BeforeClientConnect event is added

05-Oct-17 New Features in SecureBridge 8.1

- Support for the HTTP and HTTPS protocols is added
- The TScHttpWebRequest component to support the request/response model for accessing data using HTTP/HTTPS protocol is added
- Performance of downloading and uploading a file using TScSFTPClient is improved
- The TScSFTPClient.PipelineLength property to indicate the number of pending requests is added
- The TScSSHClientOptions.SocketReceiveBufferSize and SocketSendBufferSize properties to increase socket performance are added

24-Apr-17 New Features in SecureBridge 8.0

- RAD Studio 10.2 Tokyo is supported
- Linux in RAD Studio 10.2 Tokyo is supported
- Lazarus 1.6.4 and Free Pascal 3.0.2 is supported
- The TScCMSProcessor component for encrypting, decrypting, signing, and verifying data and files is added
- The TScCMSSignedData class for signing and verifying of CMS/PKCS #7 messages is added
- The TScCMSEnvelopedData class for representing encrypted data in CMS/PKCS #7 structure is added
- Elliptic Curve Key Exchange algorithm in SSH protocol is supported

16-Jan-17 New Features in SecureBridge 7.3

- Elliptic Curve Cryptography cipher suites is supported
- TScSSLClient.ClientHelloExtensions property allowing to add additional information to the client hello message is added
- TScSSLClient.ServerHelloExtensions property for additional information processing from the server hello message is added
- TTLSServerNameExtension class for support the server name indication extension is added
- TTLSExtendedMasterSecretExtension class for support the session hash and extended master secret extension is added
- TTLSSignatureAlgorithmsExtension class for support the signature algorithms extension is added
- TTLSEllipticCurvePointFormatsExtension and TTLSEllipticCurvesExtension classes for setting supported Elliptic Curves algorithms is added
- TTLSRenegotiationIndicationExtension for support the renegotiation indication extension is added
- The TScSSLClient.OnServerCertValidate event declaration is changed

10-Nov-16 New Features in SecureBridge 7.2

- Support for the TLS 1.1 and TLS 1.2 protocols is added
- Support for the Diffie-Hellman Group and Key Exchange algorithm is added
- TScSFTPServer.OnRequestFileSecurityAttributes event for an ability to increase a directory reading speed is added
- TScSFTPServer.DefaultRootPath property is added
- TScSFTPServer.OnGetFullPath event is added
- TScSSHServerOptions.ListenBacklog property is added
- Import a key from the Putty format is added

29-Jun-16 New Features in SecureBridge 7.1

- RAD Studio 10.1 Berlin is supported
- Performance of file download is improved

24-Mar-16 New Features in SecureBridge 7.0

- Lazarus 1.6 and FPC 3.0.0 is supported
- TScMemoryStorage component for storing information about keys, users and certificates in virtual memory is added
- Working with certificates avoiding CryptoAPI is supported
- Working with certificates on Mobile platforms is supported

- The SHA-2-256, SHA-2-512, SHA-2-224, SHA-2-384, and MD5 hash algorithms are supported
- The 'hmac-sha2' HMAC algorithms for using in SSH protocol are supported
- The TScSSHClient.HMACAlgorithms property for specifying the list of acceptable HMAC algorithms is added
- The TScSSHServer.HMACs property for specifying the list of the used HMAC algorithms is added
- The TScSFTPClient.BeforeWriteData event is added
- The capability to import a private key encrypted with AES-CBC algorithm is added

16-Oct-15 New Features in SecureBridge 6.6

- RAD Studio 10 Seattle is supported
- Added property TScSSHClient.HttpOptions that contains settings for HTTP connection
- Added property TScSSLClient.HttpOptions that contains settings for HTTP connection
- Support for CTR encryption mode is added
- Now Trial for Win64 is a fully functional Professional Edition

05-May-15 New Features in SecureBridge 6.5

- RAD Studio XE8 is supported
- Support for simultaneous usage of public key and password authentication on connecting to SSH server is added

30-Sep-14 New Features in SecureBridge 6.4

- RAD Studio XE7 is supported
- Lazarus 1.2.4 is supported

20-May-14 New Features in SecureBridge 6.3

- RAD Studio XE6 is supported
- Android in C++Builder XE6 is supported
- Lazarus 1.2.2 and FPC 2.6.4 is supported

04-Feb-14 New Features in SecureBridge 6.2

- iOS in C++Builder XE5 is supported
- RAD Studio XE5 Update 2 is now required

14-Oct-13 New Features in SecureBridge 6.1

- Rad Studio XE5 is supported
- Application development for Android is supported
- IPv6 protocol support is added
- Lazarus 1.0.12 is supported

09-Jul-13 New Features in SecureBridge 6.0

- Rad Studio XE4 is supported
- NEXTGEN compiler is supported
- Application development for iOS is supported

01-Feb-13 New Features in SecureBridge 5.5

- TScSFTPServer component is added

02-Oct-12 New Features in SecureBridge 5.0

- Rad Studio XE3 is supported
- Lazarus 1.0 and FPC 2.6.0 are supported
- Mac OS X in Lazarus is supported
- Linux x32 in Lazarus is supported (excluding certificate support)
- Linux x64 in Lazarus is supported (excluding certificate support)
- FreeBSD in Lazarus is supported (excluding certificate support)
- Windows 8 is supported

28-Dec-11 New Features in SecureBridge 4.1

- Update 2 for RAD Studio XE2, Delphi XE2, and C++Builder XE2 is now required
- Mac OS X in RAD Studio XE2 is supported (excluding certificate support)
- The TScSSHChannel.UseUnicode property is added

13-Oct-11 New Features in SecureBridge 4.0

- Embarcadero RAD Studio XE2 is supported
- Application development for 64-bit Windows is supported
- FireMonkey application development platform is supported

14-Oct-10 New features in SecureBridge 3.00:

- Embarcadero RAD Studio XE suppored
- Added automatic conversion of EOL symbols for text files

21-Sep-09 New features in SecureBridge 2.60:

- Embarcadero RAD Studio 2010 suppored

09-Jun-09 New features in SecureBridge 2.50:

- Added the full support for SFTP protocols versions from 1 to 6
- Added the SFTP client with extended setting abilities
- Added support of keyboard-interactive authentication method

10-Oct-08 New features in SecureBridge 2.20:

- Support for Delphi 2009 for Win 32 and C++Builder 2009 added
- Improved stability of the TScSSHServer component
- Improved work of the SSH Server demo
- Fixed bug with hanging of the TScSSLClient component

14-Nov-07 New features in SecureBridge 2.00:

- Added the full support for SSL 3.0 and TLS 1.0 protocols with no external units
- Added the SSL client with extended setting abilities
- Added ability to work with [X.509] certificates
- Added ability to access system and external certificate storages through CryptoAPI
- Remote commands execution with SSH server supported

23-Jul-07 New features in SecureBridge 1.10:

- C++Builder 2007 supported

22-May-07 New features in SecureBridge 1.00:

- SecureBridge released

2 General Information

This section contains general information about SecureBridge
Overview
Features
Compatibility
Component List
Hierarchy Chart
Editions
Requirements
Licensing and Subscriptions
Getting Support

2.1 Overview

SecureBridge is a library of non visual components for Delphi, C++Builder, and Lazarus (Free Pascal) designed to protect network connections from unauthorized access.

SecureBridge can protect any TCP traffic using SSH or TLS/SSL protocol. These secure transport layer protocols provide authentication, strong encryption, and data integrity verification. SecureBridge can be used in conjunction with data access components to prevent data interception or modification in an untrusted network.

The SecureBridge library is actively developed and supported by the Devart Team. If you have a questions about SecureBridge, email the developers at sbridge@devart.com or visit SecureBridge online at https://www.devart.com/sbridge/.

Advantages of SecureBridge Library

SecureBridge is very convenient in setup and usage. It is enough to place several components on the form and specify the server address and the user login information to establish a secure connection. Applications that have to work with secure information are easy to deploy, as they do not require any external files.

Wide Support for Secure Protocols

SecureBridge supports SSH and TLS/SSL protocols, which are one of the most reliable protocols for data encryption and integrity verification. These protocols are acknowledged industry standards in the area of secure data transfer through unprotected connections.

SSH Client

SecureBridge SSH Client, that is implemented in the TScSSHClient component, can work with different SSH servers like OpenSSH, WinSSHD. It allows you achieve high performance due to connection parameters management. SSH client unites several unprotected channels from client to server in one protected connection. Logical channels can exist in different threads.

SSH Server

High-performance SSH server with wide abilities for connection setup and users management. SSH Server works with different types of SSH clients such as OpenSSH, PuTTY etc. Number of the clients connected simultaneously is limited only by system resources.

SFTP Client

SecureBridge SFTP client, that is implemented in the TScSFTPClient component, serves for secure file transfer.

SFTP Server

SecureBridge SFTP server, that is implemented in the TScSFTPServer component, serves for secure file transfer.

SSL Client

SecureBridge TLS/SSL client, that is implemented in the TScSSLClient component, can work with other applications using TLS 1.3, 1.2, 1.1, 1.0, and SSL 3.0 protocols. It allows you achieve high performance due to connection parameters management. SecureBridge does not require external units.

SSL Server

SecureBridge TLS/SSL server, that is implemented in the TScSSLServer and the TScSSLServerConnection components, can work with different types of TLS/SSL clients using TLS 1.3, 1.2, 1.1, 1.0, and SSL 3.0 protocols. SecureBridge does not require external units.

FTPS Client

SecureBridge FTP/FTPS client, that is implemented in the TScFTPClient component, serves for secure file transfer.

HTTP/HTTPS Server

SecureBridge HTTP/HTTPS server, that is implemented in the TScHttpServer component, supports request/response model for accessing data from a Web server by the HTTP protocol.

HTTP/HTTPS Client

SecureBridge HTTP/HTTPS client, that is implemented in the TScHttpWebRequest component and the TScHttpWebResponse class, supports request/response model for accessing data from a Web server by the HTTP protocol.

WebSocket Client

SecureBridge WebSocket client, that is implemented in the TScWebSocketClient component, serves for accessing data from a Web server by the WebSocket protocol.

SignalR Client

SecureBridge SignalR client, that is implemented in the TScHubConnection component, serves for connection with a hub server.

SMTP/SMTPS Client

SecureBridge SMTP/SMTPS client, that is implemented in the TScSMTPClient component, serves for sending email messages to recipients using the SMTP server.

REST compatibility

SecureBridge allows to create REST applications using the TScHttpWebRequest component.

Protection Against Diverse Attacks

SecureBridge protects transferred data against different kinds of attacks. SecureBridge uses the Diffie-Hellman key exchange algorithm for connection establishing. A reliable random number generator is used for keys generating. To protect data against illegal access, information is encrypted by symmetric algorithms that provide high speed and reliability. For data integrity verification hash algorithms like SHA2 are used.

Support for CMS format to encrypt and sign data

The TScCMSProcessor component implements the Cryptographic Message Syntax (CMS) - syntax for data protection. It supports digital signatures and encryption.

Support for Third Party Components

SecureBridge supports Internet Direct components (Indy), MySQL Data Access Components (MyDAC) and PostgreSQL Data Access Components (PgDAC). This allows you to implement all the advantages of the encrypted connection within a single application without any external files.

Key features

The following list describes the main features of SecureBridge Components:

- Full support for SSH2 protocol
- Full support for TLS 1.3, 1.2, 1.1, 1.0, and SSL 3.0 protocols
- Support for all versions of the SFTP protocol
- Fast and customizable SSH server, SSH client, SFTP server, SFTP client, SSL server, SSL client, FTP/FTPS client, HTTP/HTTPS server, HTTP/HTTPS client, WebSocket client and SignalR client
- Support for most SSH2-compatible clients and servers including OpenSSH
- Compatible with any application that works through TCP with protocols like SMTP, POP, IMAP, etc.
- Ability to work with system and external certificate storages through CryptoAPI
- Protection against diverse crypto attacks
- Support for AES128, AES192, AES256, ChaCha20-Poly1305, Blowfish, Cast128, and TripleDES symmetric algorithms
- Support for Elliptic-Curve, RSA and DSA asymmetric algorithms
- Support for SHA-2, SHA-1, and MD5 hashing algorithms

- Authentication by password or by public key
- Support for Cryptographic Message Syntax (CMS) to encrypt, decrypt, sign, and verify data
- Deep integration with Indy, MySQL Data Access Components, and PostgreSQL Data Access Components
- High performance
- Reliable and convenient maintenance of asymmetric keys
- Facility for storing users, passwords, and public keys for an SSH server

2.2 Features

Compatibility:

- Support for Delphi 6, 7, C++Builder 6, Borland Delphi Studio 2006, Code Gear RAD Studio 2007, 2009, Embarcadero RAD Studio 12, 2010, XE, XE2, XE3, XE4, XE5, XE6, XE7, XE8, Seattle, Berlin, Tokyo, Rio, Sydney, Alexandria, Athens
- Support for Lazarus 3.0 and Free Pascal 3.2.2 on Windows, Linux and macOS (32-bit and 64-bit)
- Support for macOS Sonoma, iOS 17, iOS Simulator ARM 64-bit target platform, Android 13
- Support for Indy, an open source socket library for Internet communications
- Compatible with OpenSSH 3.8 and PuTTY
- Compatible with TLS 1.3, 1.2, 1.1, 1.0, and SSL 3.0 protocols
- Support for MySQL Data Access Components (MyDAC)
- Support for PostgreSQL Data Access Components (PgDAC)

Common features:

- Robust protection against different kinds of cryptographic attacks
- High performance
- Ability to work with system and external certificate storages through CryptoAPI
- File, registry, and temporary memory storages for certificates, keys, and users
- High quality random number generator
- Working in synchronous and asynchronous mode
- External modules are not required

Algorithms support:

- Support for AES128, AES192, AES256, ChaCha20-Poly1305, Blowfish, Cast128, and TripleDES symmetric algorithms
- Support for Elliptic-Curver, RSA and DSA asymmetric algorithms
- Support for SHA-2, SHA-1, and MD5 hashing algorithms
- Reliable and convenient storage, transfer, and verification of asymmetric keys

SSH:

- Full support for the SSH2 protocol
- SSH client with advanced configuration options
- Fast and customizable SSH server
- Remote commands execution via SSH
- Support for most SSH2-compatible clients and servers including OpenSSH
- Compatible with any applications that work through TCP with protocols like SMTP, POP, IMAP, etc.
- Facility for storing users, passwords, and public keys for an SSH server

- Authentication by password or public key
- Transferring data from several logical connections through a single SSH tunnel

SFTP:

- Full support for the SFTP protocols from version 1 to 6
- SFTP client with advanced configuration options
- Fast and customizable SFTP server

SSL/TLS:

- Full support for the TLS 1.3, 1.2, 1.1, 1.0 protocols without any external units
- Support for SSL 3.0 protocol for compatibility with older applications
- SSL/TLS client with advanced configuration options
- Fast and customizable SSL server
- Support for X.509 certificates
- Complete validation of the server certificate chain, including authority and CRL

FTP/FTPS:

- Full support for the FTP/FTPS (FTP-over-SSL) protocols

HTTP/HTTPS:

- Full support for HTTP/HTTPS (HTTP-over-SSL) protocols
- Support for request/response model for accessing HTTP data
- Fast and customizable HTTP/HTTPS server
- Ability to create REST applications using the TScHttpWebRequest component
- Support for DIGEST authentication in TScHttpWebRequest

WebSocket Secure:

- Full support for the WebSocket secure protocol

SignalR:

- Full support for the SignalR protocol

SMTP/SMTPS:

- Full support for the SMTP/SMTPS (SMTP-over-SSL) protocols

Cryptographic Message Syntax (CMS):

- Simple interface to encrypt, decrypt, sign, and verify content of any type and store it in CMS/PKCS #7 format
- Full support for a CMS signed message that allows to store the required information and enables message signing and verifying
- Full support for a CMS enveloped message that allows to store the required information and enables message encryption and decryption

Licensing and support:

- One year free support for registered users
- Licensed royalty-free per developer, per team, or per site

2.3 Compatibility

SSH servers compatibility

SecureBridge has been tested with OpenSSH 3.8 and PuTTY.

SSL/TLS compatibility

SecureBridge is compatible with TLS 1.3, 1.2, 1.1, 1.0, and SSL 3.0 protocols.

IDE compatibility

SecureBridge can be used with the following integrated development environments:

- Embarcadero RAD Studio 12 Athens
  - Embarcadero Delphi 12 Athens for Windows
  - Embarcadero Delphi 12 Athens for macOS
  - Embarcadero Delphi 12 Athens for Linux
  - Embarcadero Delphi 12 Athens for iOS
  - Embarcadero Delphi 12 Athens for Android
  - Embarcadero C++Builder 12 Athens for Windows
  - Embarcadero C++Builder 12 Athens for iOS
  - Embarcadero C++Builder 12 Athens for Android
- Embarcadero RAD Studio (up to 11.3) Alexandria
  - Embarcadero Delphi (up to 11.3) Alexandria for Windows
  - Embarcadero Delphi (up to 11.3) Alexandria for macOS
  - Embarcadero Delphi (up to 11.3) Alexandria for Linux
  - Embarcadero Delphi (up to 11.3) Alexandria for iOS
  - Embarcadero Delphi (up to 11.3) Alexandria for Android
  - Embarcadero C++Builder (up to 11.3) Alexandria for Windows
  - Embarcadero C++Builder (up to 11.3) Alexandria for iOS
  - Embarcadero C++Builder (up to 11.3) Alexandria for Android
- Embarcadero RAD Studio 10.4 Sydney (Requires Release 1 or Release 2)
  - Embarcadero Delphi 10.4 Sydney for Windows
  - Embarcadero Delphi 10.4 Sydney for macOS
  - Embarcadero Delphi 10.4 Sydney for Linux
  - Embarcadero Delphi 10.4 Sydney for iOS
  - Embarcadero Delphi 10.4 Sydney for Android
  - Embarcadero C++Builder 10.4 Sydney for Windows
  - Embarcadero C++Builder 10.4 Sydney for iOS
  - Embarcadero C++Builder 10.4 Sydney for Android
- Embarcadero RAD Studio 10.3 Rio (Requires Release 2 or Release 3)
  - Embarcadero Delphi 10.3 Rio for Windows
  - Embarcadero Delphi 10.3 Rio for macOS
  - Embarcadero Delphi 10.3 Rio for Linux
  - Embarcadero Delphi 10.3 Rio for iOS
  - Embarcadero Delphi 10.3 Rio for Android
  - Embarcadero C++Builder 10.3 Rio for Windows

  - Embarcadero C++Builder 10.3 Rio for macOS
  - Embarcadero C++Builder 10.3 Rio for iOS
  - Embarcadero C++Builder 10.3 Rio for Android
- Embarcadero RAD Studio 10.2 Tokyo
  - Embarcadero Delphi 10.2 Tokyo for Windows
  - Embarcadero Delphi 10.2 Tokyo for macOS
  - Embarcadero Delphi 10.2 Tokyo for Linux
  - Embarcadero Delphi 10.2 Tokyo for iOS
  - Embarcadero Delphi 10.2 Tokyo for Android
  - Embarcadero C++Builder 10.2 Tokyo for Windows
  - Embarcadero C++Builder 10.2 Tokyo for macOS
  - Embarcadero C++Builder 10.2 Tokyo for iOS
  - Embarcadero C++Builder 10.2 Tokyo for Android
- Embarcadero RAD Studio 10.1 Berlin
  - Embarcadero Delphi 10.1 Berlin for Windows
  - Embarcadero Delphi 10.1 Berlin for macOS
  - Embarcadero Delphi 10.1 Berlin for iOS
  - Embarcadero Delphi 10.1 Berlin for Android
  - Embarcadero C++Builder 10.1 Berlin for Windows
  - Embarcadero C++Builder 10.1 Berlin for macOS
  - Embarcadero C++Builder 10.1 Berlin for iOS
  - Embarcadero C++Builder 10.1 Berlin for Android
- Embarcadero RAD Studio 10 Seattle
  - Embarcadero Delphi 10 Seattle for Windows
  - Embarcadero Delphi 10 Seattle for macOS
  - Embarcadero Delphi 10 Seattle for iOS
  - Embarcadero Delphi 10 Seattle for Android
  - Embarcadero C++Builder 10 Seattle for Windows
  - Embarcadero C++Builder 10 Seattle for macOS
  - Embarcadero C++Builder 10 Seattle for iOS
  - Embarcadero C++Builder 10 Seattle for Android
- Embarcadero RAD Studio XE8
  - Embarcadero Delphi XE8 for Windows
  - Embarcadero Delphi XE8 for macOS
  - Embarcadero Delphi XE8 for iOS
  - Embarcadero Delphi XE8 for Android
  - Embarcadero C++Builder XE8 for Windows
  - Embarcadero C++Builder XE8 for macOS
  - Embarcadero C++Builder XE8 for iOS
  - Embarcadero C++Builder XE8 for Android
- Embarcadero RAD Studio XE7
  - Embarcadero Delphi XE7 for Windows
  - Embarcadero Delphi XE7 for macOS
  - Embarcadero Delphi XE7 for iOS
  - Embarcadero Delphi XE7 for Android
  - Embarcadero C++Builder XE7 for Windows
  - Embarcadero C++Builder XE7 for macOS

  - Embarcadero C++Builder XE7 for iOS
  - Embarcadero C++Builder XE7 for Android
- Embarcadero RAD Studio XE6
  - Embarcadero Delphi XE6 for Windows
  - Embarcadero Delphi XE6 for macOS
  - Embarcadero Delphi XE6 for iOS
  - Embarcadero Delphi XE6 for Android
  - Embarcadero C++Builder XE6 for Windows
  - Embarcadero C++Builder XE6 for macOS
  - Embarcadero C++Builder XE6 for iOS
  - Embarcadero C++Builder XE6 for Android
- Embarcadero RAD Studio XE5 (Requires Update 2)
  - Embarcadero Delphi XE5 for Windows
  - Embarcadero Delphi XE5 for macOS
  - Embarcadero Delphi XE5 for iOS
  - Embarcadero Delphi XE5 for Android
  - Embarcadero C++Builder XE5 for Windows
  - Embarcadero C++Builder XE5 for macOS
  - Embarcadero C++Builder XE5 for iOS
- Embarcadero RAD Studio XE4
  - Embarcadero Delphi XE4 for Windows
  - Embarcadero Delphi XE4 for macOS
  - Embarcadero Delphi XE4 for iOS
  - Embarcadero C++Builder XE4 for Windows
  - Embarcadero C++Builder XE4 for macOS
- Embarcadero RAD Studio XE3 (Requires Update 2)
  - Embarcadero Delphi XE3 for Windows
  - Embarcadero Delphi XE3 for macOS
  - Embarcadero C++Builder XE3 for Windows
  - Embarcadero C++Builder XE3 for macOS
- Embarcadero RAD Studio XE2 (Requires Update 4 Hotfix 1)
  - Embarcadero Delphi XE2 for Windows
  - Embarcadero Delphi XE2 for macOS
  - Embarcadero C++Builder XE2 for Windows
  - Embarcadero C++Builder XE2 for macOS
- Embarcadero RAD Studio XE
  - Embarcadero Delphi XE
  - Embarcadero C++Builder XE
- Embarcadero RAD Studio 2010
  - Embarcadero Delphi 2010
  - Embarcadero C++Builder 2010
- CodeGear RAD Studio 2009 (Requires Update 3)
  - CodeGear Delphi 2009
  - CodeGear C++Builder 2009
- CodeGear RAD Studio 2007
  - CodeGear Delphi 2007
  - CodeGear C++Builder 2007

- CodeGear RAD Studio 2006
  - CodeGear Delphi 2006
  - CodeGear C++Builder 2006
- Borland Delphi 7
- Borland Delphi 6 (Requires Update Pack 2 - Delphi 6 Build 6.240)
- Borland C++Builder 6 (Requires Update Pack 4 - C++Builder 6 Build 10.166)
- Lazarus 3.0 and Free Pascal 3.2.2 for Windows, macOS, and Linux

SecureBridge supports only Professional, Enterprise, and Architect IDE editions.

Note that support for Windows 64-bit is available since RAD Studio XE2. Support for iOS 64-bit is available since RAD Studio XE8. Support for Android 32-bit is available since RAD Studio XE5. Support for Linux 64-bit is available since RAD Studio 10.2 Tokyo. Support for macOS 64-bit is available since RAD Studio 10.3 Rio. Support for Android 64-bit is available since RAD Studio 10.3.3 Rio.

Supported Target Platforms

- Windows 32-bit and 64-bit
- macOS 64-bit and ARM (Apple Silicon M1)
- iOS 64-bit
- Android 32-bit and 64-bit
- Linux 32-bit (only in Lazarus and Free Pascal) and 64-bit

2.4 Component List

This topic presents a brief description of the components included in the SecureBridge Components library. Click on the name of each component for more information. These components are added to the SecureBridge page of the Component palette.

	TScSSHClient	SSH client unites several logical unprotected connections to the server into one protected connection. Logical connections can exist in different threads.
	TScSSHChannel	Logical connection to SSH client within the client secure area. Receives/sends data from/to SSH server or forwards from the TCP port of one computer to another computer through a secure channel.
	TScSSHShell	Serves for remote commands execution using abilities of an SSH server.
	TScSFTPClient	Serves for implementing the functionality of SFTP protocol that provides secure file transfer.
	TScSSHServer	Implements the SSH server functionality.
	TScSFTPServer	Implements the SFTP server functionality.

	TScSSLClient	TLS/SSL client, supports TLS 1.3, 1.2, 1.1, 1.0, and SSL 3.0 protocols. It validates server certificate, encrypts/ decrypts data transferred through a network.
	TScSSLServerConnection	Implements the TLS/SSL server functionality. It supports TLS 1.3, 1.2, 1.1, 1.0, and SSL 3.0 protocols. The server validates client certificates and encrypts/decrypts data transferred through a network.
	TScFTPClient	Serves for implementing the functionality of FTP/FTPS protocols that provides file transfer.
	TScMemoryStorage	Stores list of certificates, keys, and users in RAM memory.
	TScFileStorage	Stores list of certificates, keys, and users in files.
	TScRegStorage	Stores list of certificates, keys, and users in the system registry.
	TScCryptoAPIStorage	Stores list of certificates and keys in system and external storages using CryptoAPI functionality.
	TScCMSProcessor	Provides a simple interface to encrypt, decrypt, sign, and verify content of any type and store them in CMS/PKCS #7 format.
	TScHttpWebRequest	Provides client-side functionality for accessing data by the HTTP/HTTPS protocols.
	TScWebSocketClient	Provides client-side functionality for accessing data by the WebSocket protocol.
	TScHubConnection	Provides client-side functionality for connecting to the SignalR server.
	TScSMTPClient	Implements the functionality of an SMTP/SMTPS client that uses the SMTP protocol to send email messages to recipients using the SMTP server.
	TScTCPServer	Implements the functionality of a TCP/IP server.
	TScIdIOHandler	Provides easy integration with Indy components to protect data transferred through network by Indy.
	TCRSSHIOHandler	Enables MyDAC to connect to the MySQL server through the SSH protocol (this component is included into MyDAC as a demo project).
	TCRSSLIOHandler	Enables MyDAC to connect to the MySQL server through an SSL connection (this component is included into MyDAC as a demo project).

See Also

Hierarchy chart

2.5 Hierarchy Chart

Many SecureBridge classes are inherited from standard VCL/LCL classes. The inheritance hierarchy chart for SecureBridge is shown below. The SecureBridge classes are represented by hyperlinks that point to their description in this documentation. A description of the standard classes can be found in the documentation of your IDE.

```
TObject
  |-TList
  |     |-TTLSHelloExtensions
  |-TPersistent
  |     |-TCollection
  |     |     |-TScAttachmentCollection
  |     |     |-TScCollection
  |     |     |     |-TScSFTPACEs
  |     |     |     |-TScSSHCiphers
  |     |     |     |-TScSSHHMacAlgorithms
  |     |     |     |-TScSSHHostKeyAlgorithms
  |     |     |     |-TScSSHKeyExchangeAlgorithms
  |     |     |     |-TScSSL
  |     |     |-TScFTPDirectoryListing
  |     |     |-TScMailAddressList
  |     |     |-TScSASLCollection
  |     |-TCollectionItem
  |     |     |-TScCollectionItem
  |     |     |     |-TScSFTPACEItem
  |     |     |     |-TScSSHCipherItem
  |     |     |     |-TScSSHHMacAlgorithmItem
  |     |     |     |-TScSSHHostKeyAlgorithmItem
  |     |     |     |-TScSSHKeyExchangeAlgorithmItem
  |     |     |     |-TScSSLCipherSuiteItem
  |     |     |-TScCustomAttachment
  |     |     |     |-TScAttachment
  |     |     |     |-TScTextAttachment
  |     |     |-TScFTPListItem
  |     |     |-TScMailAddressItem
  |     |     |-TScSASLItem
  |     |-TComponent
  |     |     |-TIdIOHandler
  |     |     |     |-TScIdIOHandler
  |     |     |-TMyIOHandler
  |     |     |   |-TMySSHIOHandler
  |     |     |   |-TMySSLIOHandler
  |     |     |-TScCMSProcessor
  |     |     |-TScFTPClient
  |     |     |-TScHttpWebRequest
  |     |     |-TScHubConnection
  |     |     |-TScSFTPClient
  |     |     |-TScSFTPServer
  |     |     |-TScSMTPClient
```

```
|     |     |-TTLSExtendedMasterSecretExtension
|     |     |-TTLSRenegotiationIndicationExtension
|     |     |-TTLSServerNameExtension
|     |     |-TTLSSessionTicketExtension
|     |     |-TTLSSignatureAlgorithmsExtension
|     |     |-TTLSSupportedGroupsExtension
|-TScCancellationToken
|-TScCMSContentInfo
|-TScCMSData
|     |-TScCMSEnvelopedData
|     |-TScCMSSignedData
|-TScCMSOriginatorIdentifierOrKey
|-TScCMSRecipient
|-TScCMSSubjectIdentifier
|-TScHandle
|-TScLogger
|-TScMimeBoundary
|-TScOAEPParams
|-TScPersistent
|     |-TScASN1AlgorithmIdentifier
|     |     |-TScSignatureAlgorithmIdentifier
|     |-TScASN1Attribute
|     |-TScCertificateExtension
|     |     |-TScCertAlternativeNameExtension
|     |     |-TScCertAuthorityInfoAccessExtension
|     |     |-TScCertAuthorityKeyIdExtension
|     |     |-TScCertBasicConstraintsExtension
|     |     |-TScCertCRLDistributionPointsExtension
|     |     |-TScCertExtendedKeyUsageExtension
|     |     |-TScCertKeyUsageExtension
|     |     |-TScCertPoliciesExtension
|     |     |-TScCertPolicyMappingsExtension
|     |     |-TScCertSubjectDirectoryAttributesExtension
|     |     |-TScCertSubjectInfoAccessExtension
|     |     |-TScCertSubjectKeyIdExtension
|     |     |-TScCRLCertificateIssuerExtension
|     |     |-TScCRLDeltaIndicatorExtension
|     |     |-TScCRLInvalidityDateExtension
|     |     |-TScCRLIssuingDistributionPointExtension
|     |     |-TScCRLNumberExtension
|     |     |-TScCRLReasonCodeExtension
|     |-TScCMSRecipientInfo
|     |     |-TScCMSKEKRecipientInfo
|     |     |-TScCMSKeyAgreeRecipientInfo
|     |     |-TScCMSKeyTransRecipientInfo
|     |     |-TScCMSPasswordRecipientInfo
|     |-TScCMSSignerInfo
|     |     |-TScCMSSignature
|     |-TScDistinguishedName
|     |-TScGeneralName
|     |-TScOId
|     |-TScPKCS7Attribute
|     |-TScPolicy
```

```
|     |-TScPolicyMapping
|     |-TScRelativeDistinguishedName
|-TScPersistentObjectList
|     |-TScASN1AlgorithmIdentifiers
|     |-TScASN1Attributes
|     |     |-TScCMSSMIMEAttributes
|     |-TScCMSRecipientInfos
|     |-TScCMSSignatures
|     |-TScDistinguishedNameList
|     |-TScExtensions
|     |-TScGeneralNames
|     |-TScOIds
|     |-TScPKCS7Attributes
|     |     |-TScCMSSignedAttributes
|     |     |-TScCMSUnsignedAttributes
|     |-TScPolicyList
|     |-TScPolicyMappingList
|-TScPSSParams
|-TScRandom
|     |-TScRandom_LFSR
|-TScRetryPolicy
|-TScSASLMechanism
|     |-TScSASLAnonymous
|     |-TScSASLUserPassMechanism
|     |     |-TScSASLCRAMMD5
|     |     |-TScSASLCRAMSHA1
|     |     |-TScSASLLogin
|     |     |-TScSASLOTP
|     |     |-TScSASLPlain
|     |     |-TScSASLSKey
|-TScSearchRec
|-TScSFTPServerProperties
|-TScSFTPSessionInfo
|-TScSSHChannelInfo
|-TScSSHConnectionInfo
|     |-TScSSHClientInfo
|-TScSSLSessionInfo
|-TScStorageList
|     |-TScCertificateList
|     |-TScCRLList
|     |-TScKeyList
|     |-TScUserList
|-TScStreamInfo
|-TScTCPConnection
|-TStream
|     |-TScSSHStream
|-TStrValueStringList
|     |-TScHeaderList
|     |-TScWebHeaderCollection
|     |     |-TScWebRequestHeaderCollection
|     |     |-TScWebResponseHeaderCollection
```

2.6 Editions

SecureBridge comes in three editions: Standard Edition, Professional Edition,and Trial Edition.

The Standard edition includes the SecureBridge basic secure connectivity components. SecureBridge Standard Edition is a cost-effective solution for database application developers who are looking for an overall high-performance security tool.

The Professional edition shows off the full power of SecureBridge, enhancing SecureBridge Standard Edition with server functionality.

The Trial Edition is the evaluation version of SecureBridge. It includes all the functionality of SecureBridge Professional Edition with a trial limitation of 60 days. C++Builder has additional trial limitations.

You can get source code of all the component classes in SecureBridge by purchasing the special SecureBridge Professional Edition with Source Code.

The matrix below compares features of the SecureBridge editions. The detailed list of all SecureBridge features you can find at the SecureBridge Features page.

Features	Professional	Standard
Mobile Development		
iOS application development	✔	✘
Android application development		
Server functionality		
TScSFTPServer	✔	✘
TScSSHServer		
TScSSLServerConnection		
TScTCPServer		
Client functionality		
TScSMTPClient	✔	✔
TScFTPClient		
TScSFTPClient		
TScSSHClient		
TScSSLClient		
Storage functionality		
TScCryptoAPIStorage	✔	✔
TScFileStorage		
TScMemoryStorage		
TScRegStorage		
Secure channel	✔	✔

Features	Professional	Standard
TScSSHChannel		
Remote commands execution TScSSHShell	✔	✔
Data encryption and signing TScCMSProcessor	✔	✔
HTTP/HTTPS client functionality TScHttpWebRequest TScWebSocketClient TScHubConnection	✔	✔
SMTP/SMTPS client functionality TScSMTPClient	✔	✔
Integration with Indy components TScIdIOHandler	✔	✔
Integration with DAC components TCRSSHIOHandler TCRSSLIOHandler	✔	✔
Cross IDE Support Lazarus and Free Pascal support	✔*	✘

* Available only in editions with source code.

See also

Overview

2.7 Requirements

SecureBridge is an all-sufficient library and it does not require any external files on the target computer.

2.8 Licensing and Subscriptions

PLEASE READ THIS LICENSE AGREEMENT CAREFULLY. BY INSTALLING OR USING THIS SOFTWARE, YOU INDICATE ACCEPTANCE OF AND AGREE TO BECOME BOUND BY THE TERMS AND CONDITIONS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS

LICENSE, DO NOT INSTALL OR USE THIS SOFTWARE AND PROMPTLY RETURN IT TO DEVART.

INTRODUCTION

This Devart end-user license agreement ("Agreement") is a legal agreement between you (either an individual person or a single legal entity) and Devart, for the use of SecureBridge software application, source code, demos, intermediate files, printed materials, and online or electronic documentation contained in this installation file. For the purpose of this Agreement, the software program(s) and supporting documentation will be referred to as the "Software".

LICENSE

1. GRANT OF LICENSE

The enclosed Software is licensed, not sold. You have the following rights and privileges, subject to all limitations, restrictions, and policies specified in this Agreement.

1.1. If you are a legally licensed user, depending on the license type specified in the registration letter you have received from Devart upon purchase of the Software, you are entitled to either:

- install and use the Software on one or more computers, provided it is used by 1 (one) for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Single Developer License"); or

- install and use the Software on one or more computers, provided it is used by up to 4 (four) developers within a single company at one physical address for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Team Developer License"); or

- install and use the Software on one or more computers, provided it is used by developers in a single company at one physical address for the sole purposes of developing, testing, and deploying applications in accordance with this Agreement (the "Site License").

1.2. If you are a legally licensed user of the Software, you are also entitled to:

- make one copy of the Software for archival purposes only, or copy the Software onto the hard disk of your computer and retain the original for archival purposes;

- develop and test applications with the Software, subject to the Limitations below;

- create libraries, components, and frameworks derived from the Software for personal use only;

- deploy and register run-time libraries and packages of the Software, subject to the Redistribution policy defined below.

1.3. You are allowed to use evaluation versions of the Software as specified in the Evaluation section.

No other rights or privileges are granted in this Agreement.

2. LIMITATIONS

Only legally registered users are licensed to use the Software, subject to all of the conditions of this Agreement. Usage of the Software is subject to the following restrictions.

2.1. You may not reverse engineer, decompile, or disassemble the Software.

2.2. You may not build any other components through inheritance for public distribution or commercial sale.

2.3. You may not use any part of the source code of the Software (original or modified) to build any other components for public distribution or commercial sale.

2.4. You may not reproduce or distribute any Software documentation without express written permission from Devart.

2.5. You may not distribute and sell any portion of the Software without integrating it into your Applications as Executable Code, except Trial edition that can be distributed for free as original Devart's SecureBridge Trial package.

2.6. You may not transfer, assign, or modify the Software in whole or in part. In particular, the Software license is non-transferable, and you may not transfer the Software installation package.

2.7. You may not remove or alter any Devart's copyright, trademark, or other proprietary rights notice contained in any portion of Devart units, source code, or other files that bear such a notice.

3. REDISTRIBUTION

The license grants you a non-exclusive right to compile, reproduce, and distribute any new software programs created using SecureBridge. You can distribute SecureBridge only in compiled Executable Programs or Dynamic-Link Libraries with required run-time libraries and packages.

All Devart's units, source code, and other files remain Devart's exclusive property.

4. TRANSFER

You may not transfer the Software to any individual or entity without express written permission from Devart. In particular, you may not share copies of the Software under “Single Developer License” and “Team License” with other co-developers without obtaining proper license of these copies for each individual.

5. TERMINATION

Devart may immediately terminate this Agreement without notice or judicial resolution in the event of any failure to comply with any provision of this Agreement. Upon such termination you must destroy the Software, all accompanying written materials, and all copies.

6. EVALUATION

Devart may provide evaluation ("Trial") versions of the Software. You may transfer or distribute Trial versions of the Software as an original installation package only. If the Software you have obtained is marked as a "Trial" version, you may install and use the Software for a period of up to 60 calendar days from the date of installation (the "Trial Period"), subject to the additional restriction that it is used solely for evaluation of the Software and not in conjunction with the development or deployment of any application in production. You may not use applications developed using Trial versions of the Software for any commercial purposes. Upon expiration of the Trial Period, the Software must be uninstalled, all its copies and all accompanying written materials must be destroyed.

7. WARRANTY

The Software and documentation are provided "AS IS" without warranty of any kind. Devart makes no warranties, expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or use.

8. SUBSCRIPTION AND SUPPORT

The Software is sold on a subscription basis. The Software subscription entitles you to download improvements and enhancement from Devart's web site as they become available, during the active subscription period. The initial subscription period is one year from the date of purchase of the license. The subscription is automatically activated upon purchase, and may be subsequently renewed by Devart, subject to receipt applicable fees. Licensed users of the Software with an active subscription may request technical assistance with using the Software over email from the Software development. Devart shall use its reasonable endeavours to answer queries raised, but does not guarantee that your queries or problems will be fixed or solved.

Devart reserves the right to cease offering and providing support for legacy IDE versions.

9. COPYRIGHT

The Software is confidential and proprietary copyrighted work of Devart and is protected by international copyright laws and treaty provisions. You may not remove the copyright notice from any copy of the Software or any copy of the written materials, accompanying the Software.

This Agreement contains the total agreement between the two parties and supersedes any other agreements, written, oral, expressed, or implied.

2.9 Getting Support

This page lists several ways you can find help with using SecureBridge and describes the SecureBridge Priority Support program.

Support Options

There are a number of resources for finding help on installing and using SecureBridge.

- You can find out more about SecureBridge installation or licensing by consulting the Licensing section.
- You can get community assistance and technical support on the SecureBridge Community Forum.
- You can get advanced technical assistance by SecureBridge developers through the **SecureBridge Priority Support** program.

If you have a question about ordering SecureBridge or any other Devart product, please contact sales@devart.com.

SecureBridge Priority Support

SecureBridge Priority Support is an advanced product support service for getting expedited individual assistance with SecureBridge-related questions from the SecureBridge developers themselves. Priority Support is carried out over email and has a two business day response policy. Priority Support is available for users with an active SecureBridge Subscription.

To get help through the SecureBridge Priority Support program, please send an email to sbridge@devart.com describing the problem you are having. Make sure to include the following information in your message:

- The version of Delphi or C++Builder you are using.
- Your SecureBridge Registration number.
- Full SecureBridge edition name and version number.
- A detailed problem description.
- If possible, a small test project that reproduces the problem. Please include definitions for all objects of other schemas and avoid using third-party components.

3 Getting Started

This page contains a quick introduction to setting up and using the SecureBridge library. It gives a walkthrough for each part of the SecureBridge usage process and points out the most relevant related topics in the documentation.

What is SecureBridge?
How does SecureBridge work?
Installing SecureBridge.

Working with the SecureBridge demo projects.
Compiling and deploying your SecureBridge project.
Using the SecureBridge documentation.
How to get help with SecureBridge.

What is SecureBridge?

SecureBridge is a component library which is designed for ensuring safe data transferring through insecure network areas. It helps you to improve security of transferred information in your applications. However it practically does not affect performance of the application and does not complicate its architecture.

An introduction to SecureBridge is provided in the Overview section.

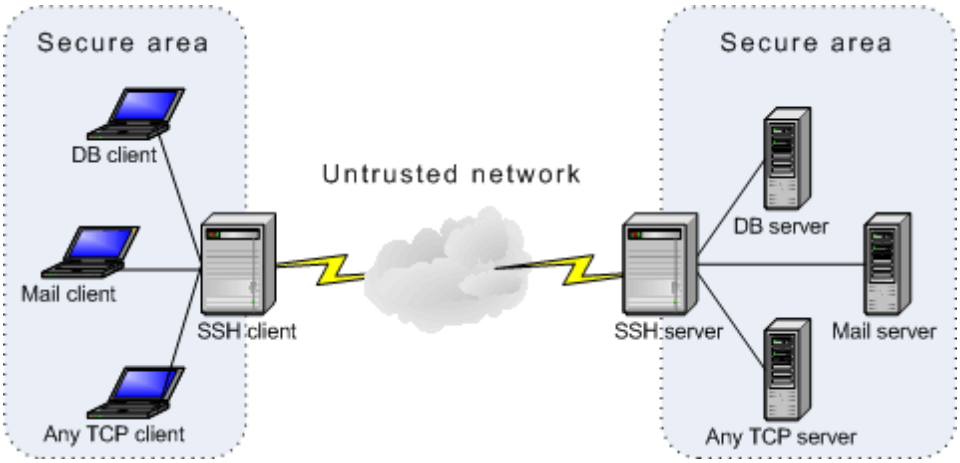
A list of the SecureBridge features you may find useful is listed in the Features section.

An overview of the SecureBridge component classes is provided in the Components List section.

How does SecureBridge work?

In order to ensure data safety in insecure networks, it is essential to take care of data protection and integrity, as well as of the data receiver identification. So before putting the data into the insecure area, it should be encrypted. To maintain data integrity, it is required to send a data hash along with the data itself. On the other side the data should be decrypted, and received hash should be verified.

SSH tunnel can ensure data transferring from several clients of one secure area to clients in another secure area through one protected TCP connection, at that authentication of the remote side is ensured. The general chart of computer ties when connecting through the SSH tunnel is presented below:

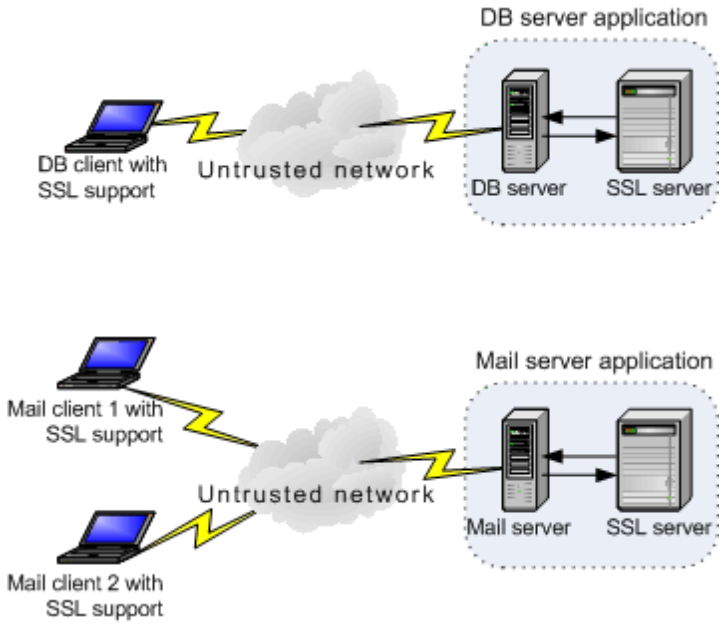


SSH tunnel diagram

SecureBridge can act as both SSH client (TScSSHClient) and SSH server (TScSSHServer).

SSL connection works in similar way. The difference is that SSL client and SSL server are embedded into the corresponding applications. To put some data into network, an application calls methods of the embedded SSL client (TScSSLClient), data is encrypted and sent. To get data from network, the application also calls methods of the embedded SSL client. So, SSL client/server exchange data with the application within the application's address space. The general chart of computer ties when

connecting through SSL is presented below:



SSL connection diagram

Installing SecureBridge

To install SecureBridge, complete the following steps.

1. Choose and download the version of the SecureBridge installation program that is compatible with your IDE. For more information, visit the SecureBridge download page.
2. Close all running IDEs.
3. Launch the SecureBridge installation program you downloaded in the first step and follow the instructions to install SecureBridge.

To check SecureBridge has been installed properly, launch your IDE and make sure that the SecureBridge page has been added to the Component Palette.

If you have bought SecureBridge Professional Edition with Source Code, you will be able to download both the compiled version of SecureBridge and the SecureBridge source code. The installation process for the compiled version is standard, as described above. The SecureBridge source code must be compiled and installed manually. Consult the supplied *ReadmeSrc.txt* file for more details.

To find out what gets installed with SecureBridge or to troubleshoot your SecureBridge installation, visit the Installation topic.

Working with the SecureBridge demo projects

The SecureBridge installation package includes demo projects that demonstrate SecureBridge capabilities and use patterns. The SecureBridge demo projects are automatically installed in the SecureBridge installation folder.

To quickly get started working with SecureBridge, choose a fit SecureBridge demo project, and launch it. A description of all the SecureBridge demos is located in the Demo Projects topic.

Compiling and deploying your SecureBridge project

Compiling SecureBridge-based projects

By default, to compile a project that uses SecureBridge classes, your IDE compiler needs to have access to the SecureBridge dcu (obj) files. If you are compiling a project with runtime packages, the compiler will also need to have access to the SecureBridge bpl files. **All the appropriate settings for both of these scenarios should take place automatically during installation of SecureBridge.** You should only need to modify your environment manually if you are using SecureBridge edition that comes with source code.

You can check that your environment is properly configured by trying to compile one of the SecureBridge demo projects. If you have no problem compiling and launching the SecureBridge demos, your environment has been properly configured.

For more information about which library files and environment changes are needed for compiling SecureBridge-based projects, consult the Installation topic.

Deploying SecureBridge-based projects

To deploy an application that uses SecureBridge, you will need to make sure the target workstation has access to the following files.

- The SecureBridge bpl files, if compiling with runtime packages.

If you are evaluating deploying projects with SecureBridge Trial Edition, you will also need to deploy some additional bpl files with your application even if you are compiling without runtime packages. As another trial limitation for C++Builder, applications written SecureBridge Trial Edition for C++Builder will only work if the C++Builder IDE is launched. More information about SecureBridge Trial Edition limitations is provided here.

Files which may need to be deployed with SecureBridge-based applications is included in the Deployment topic.

Using the SecureBridge documentation

The SecureBridge documentation describes how to install and configure SecureBridge, how to use SecureBridge Demo Projects, and how to use the SecureBridge library.

The SecureBridge documentation includes a detailed reference of all the SecureBridge components and classes. The product documentation also includes a summary of all members within each of these classes. To view a detailed description of a particular component, look it up in the Components List section. To find out more about a specific standard VCL class an SecureBridge component is inherited from, see the corresponding topic in your IDE documentation.

At install time, the SecureBridge documentation is integrated into your IDE. It can be invoked by pressing F1 in an object inspector or on a selected code segment.

How to get help with SecureBridge

There are a number of resources for finding help on using SecureBridge classes in your project.

- If you have a question about SecureBridge installation or licensing, consult the Licensing section.
- You can get community assistance and SecureBridge technical support on the SecureBridge Support Forum.
- To get help through the SecureBridge Priority Support program, send an email to the SecureBridge development team at securebridge@devart.com.

- If you have a question about ordering SecureBridge or any other Devart product, contact sales@devart.com.

For more information, consult the Getting Support topic.

3.1 Installation

This topic contains the environment changes made by the SecureBridge installer. If you are having problems with using SecureBridge or compiling SecureBridge-based products, check this list to make sure your system is properly configured.

Compiled versions of SecureBridge are installed automatically by the SecureBridge Installer for all supported IDEs. Versions of SecureBridge with Source Code must be installed manually.

Installed packages

Note: `%SecureBridge%` denotes the path to your SecureBridge installation directory.

Delphi/C++Builder Win32 project packages

Name	*Description*	*Location*
sbridgeXX.bpl	SecureBridge run-time package	Windows\System32
dclsbridgeXX.bpl	SecureBridge design-time package	Delphi\Bin
indy10sbridgeXX.bpl*	TScIdIOHandler compatible with Indy10	Delphi\Bin
indy9sbridgeXX.bpl*	TScIdIOHandler compatible with Indy9	Delphi\Bin

Environment Changes

To compile SecureBridge-based applications, your environment must be configured to have access to the SecureBridge libraries. Environment changes are IDE-dependent.

For all instructions, replace `%SecureBridge%` with the path to your SecureBridge installation directory.

Delphi

- `%SecureBridge%\Lib` should be included in the Library Path accessible from Tools | Environment options | Library.

The SecureBridge Installer performs Delphi environment changes automatically for compiled versions of SecureBridge.

C++Builder

C++Builder 6:

- `$(BCB)\SecureBridge\Lib` should be included in the Library Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.
- `$(BCB)\SecureBridge\Include` should be included in the Include Path of the Default Project Options accessible from Project | Options | Directories/Conditionals.

C++Builder 2006, 2007:

- `$(BCB)\SecureBridge\Lib` should be included in the Library search path of the Default Project Options accessible from Project | Default Options | C++Builder | Linker | Paths and Defines.
- `$(BCB)\SecureBridge\Include` should be included in the Include search path of the Default Project Options accessible from Project | Default Options | C++Builder | C++ Compiler | Paths and Defines.

The SecureBridge Installer performs C++Builder environment changes automatically for compiled versions of SecureBridge.

Lazarus on macOS

The SecureBridge installation program only copies SecureBridge files. You need to install SecureBridge packages to Lazarus IDE manually. In the IDE, select `Package > Open Package File (.lpk)`.
and open the `%SecureBridge%\Source\Lazarus1\sbridge10.lpk` file. In the window that opens, click `Options` and enter the path to SecureBridge source code (`~/usr/local/share/fpcsrc/packages/users/src`) in the `Compiler Options > Paths > Other unit files (-FU)` field. Press the `Install` button. After that Lazarus IDE will be rebuilded with SecureBridge packages.

If you use a trial version of SecureBridge, do not press the Compile button for the package. Compiling will fail because of lack of SecureBridge source code files.

To check that your environment has been properly configured, try to compile one of the demo projects included with SecureBridge. The SecureBridge demo projects are located in `%SecureBridge%/Demos`.

3.2 Demo Projects

SecureBridge includes demo projects that show off the main SecureBridge functionality and development patterns.

Where are the SecureBridge demo projects located?

All the SecureBridge demo projects are located in "%Public Documents%\Devart\SecureBridge for XX \Demos", for example: "c:\Users\Public\Documents\Devart\SecureBridge for RAD Studio 10.2\Demos \".

The structure of the demo project directory depends on the IDE version you are using.

Instructions for using the SecureBridge demo projects

To explore an SecureBridge demo project,

1. Launch your IDE.

2. In your IDE, choose File | Open Project from the menu bar.

3. Find the Demos folder of SecureBridge.

4. Browse through the demo project folders located here and open the project file of the demo you would like to use.

5. Compile and launch the demo. If it exists, consult the Readme.html file for more details.

Demo project descriptions

Name	*Description*
Indy10	This demo project represents a the TScIdIOHandler component for providing integration with Indy components version 10. SecureBridge installation wizard installs it for Delphi 10 and higher IDE versions if the "Indy Components" item is checked on the "Select Components" step of the installation. If your IDE has Indy9 installed, and Indy integration is needed, just uncheck "Indy Components" when installing and install TScIdIOHandler from the Indy9 directory.
Indy9	This demo is an equivalent to the Indy10 demo, except it supports Indy components version 9, and is automatically installed for Delphi 7.
SFTPClient	Uses the TScSFTPClient component for secure file transfer with remote machine. This demo allows to execute basic operations with files such as downloading, uploading files, creating and deleting directories, viewing the directory tree.
SSHClient	Uses the TScSSHClient component for establishing connection to an SSH server. Demonstrates organizing port forwarding with the TScSSHChannel component (see. SSH-tunnel principles).
SSHServer	Use the TScSSHServer component for building a full-blown SSH server. Demonstrates working with a user list, generating new keys that are used for authenticating when client connects to server.
SSHServerService	This is one more full-blown SSH sever, but it does not have graphic interface and does not provide key and user management tools like SSHServer demo does. This demo is intended to work as a Windows service. Take a look at the Readme file in the demo directory for some additional information.

Note, there is the Base directory among the demo directories. This directory does not contain a demo, it contains a common engine for some of demos. You should not remove this directory or files in it. If you do that, some of demos will not compile and work.

3.3 Deployment

SecureBridge applications can be built and deployed with or without run-time libraries. Using run-time libraries is managed with the "Build with runtime packages" check box in the Project Options dialog box.

Deploying Windows applications built without run-time packages

You do not need to deploy any files with SecureBridge-based applications built without run-time packages, provided you are using a registered version of SecureBridge.

You can check your application does not require run-time packages by making sure the "Build with runtime packages" check box is not selected in the Project Options dialog box.

Trial Limitation Warning

If you are evaluating deploying Windows applications with SecureBridge Trial Edition, you will need to deploy the sbridgeXX.bpl package and their dependencies (required IDE BPL files) with your application, even if it is built without run-time packages.

Deploying Windows applications built with run-time packages

You can set your application to be built with run-time packages by selecting the "Build with runtime packages" check box in the Project Options dialog box before compiling your application.

In this case, you will also need to deploy the sbridgeXX.bpl package with your Windows application.

4 Using SecureBridge

4.1 Secure connections destination

SSH (Secure Shell) and SSL (Secure Sockets Layer) are protocols for secure access to remote computers over insecure communication channels. Secure communication over non-secure networks generally involves three major areas of concern: privacy, authentication, and integrity.

Privacy

There is a possibility of an unauthorized access when transferring confidential information. To prevent the unauthorized access, data encryption is used. It is practically impossible to transform encrypted data to the initial view without the secret key if a good encryption algorithm is used. It was designed a quantity of algorithms for data encryption that differ in reliability and encryption speed. The SSH and TLS/SSL protocols support several algorithms of symmetric encryption and let using different algorithms for passed and received data.

When using these algorithms, it is necessary to have a secret session key, that is used for data encryption and decryption. Both SSH and TLS/SSL generate keys before beginning of data exchange. Also they allow regenerating this key when working to avoid cracking the key.

Authentication

Secure communications require that the individuals communicating know the identity of those with whom they communicate.

When the client tries to establish the connection to the server, it is necessary to be sure that the server is authentic (not supposititious). Also the server should verify whether the client is allowed co connect to it. To implement such requirements, asymmetric encrypting algorithms are used. In these algorithms a pair of keys is used. The first key, named private key, serves for encrypting or signing data blocks. The second key, named public key, serves for decryption data and signature verification. When pretty long keys are used, it is not possible to determine the private key for a reasonable time interval if the public key is known.

Each secure server must have a pair of keys. In order to authenticate the sever, the client must have a public key/certificate of the server. When creating the secure connection to authenticate the server, the client verifies the key/certificate and signature received from the server using by the public key that the client has. If the verification passes, the server is considered valid.

There are several ways to authenticate the client. The first way is when the server verifies user name password. The second way is when the client has a pair of his own keys or a certificate, and the public key has to be passed to the server. At that the client authentication is analogous to the server authentication described above.

Integrity

It is necessary to be sure that the data transferred through an insecure channel is not changed or lost. For that data integrity checking is required.

Integrity check of the received data is often done by sending not only the original data but also a verification message about that data. This message is called digital signature. Both the data and the verification message can be sent with a digital signature that proves the origin of both.

4.2 Network Tunneling

Connection through HTTP tunnel

Sometimes client machines are shielded by a firewall that does not allow you to connect to server directly at the specified port. If the firewall allows HTTP connections, you can use SecureBridge together with HTTP tunneling software to connect to an SSH server.

SecureBridge supports HTTP tunneling based on the PHP script.

An example of the web script tunneling usage can be the following: you have a remote website, and access to its SSH server through the port of this server is forbidden. Only access through HTTP port 80 is allowed, and you need to access the SSH server from a remote computer, like when using usual direct connection.

You need to deploy the tunnel.php script, which is included into the provider package on the web server. It allows access to the SSH server to use HTTP tunneling. The script must be available through the HTTP protocol. You can verify if it is accessible with a web browser. The script can be found in the HTTP subfolder of the installed provider folder, e. g. %Program Files%\Devart \SecureBridge for Delphi X\HTTP\tunnel.php. The only requirement to the server is PHP 5 support.

To connect to the SSH server, you should set TScSSHClient parameters for usual direct connection, which will be established from the web server side, the HttpOptions.Enabled property to True, and set

the following parameters, specific for the HTTP tunneling:

Property	Mandatory	Meaning
HttpOptions.Url	Yes	Url of the tunneling PHP script. For example, if the script is in the server root, the url can be the following: http://localhost/tunnel.php.
HttpOptions.Username, HttpOptions.Password	No	Set this properties if the access to the website folder with the script is available only for registered users authenticated with user name and password.

Connection through proxy and HTTP tunnel

Consider the previous case with one more complication.

HTTP tunneling server is not directly accessible from client machine. For example, client address is 10.0.0.2, server address is 192.168.0.10, and the SSH server listens on port 22. The client and server reside in different networks, so the client can reach it only through proxy at address 10.0.0.1, which listens on port 808. In this case in addition to the TScSSHClient.HttpOptions options you have to setup a HttpOptions.ProxyOptions object as follows:

```
ScSSHClient := TScSSHClient.Create(self);
ScSSHClient.KeyStorage := ScFileStorage;
ScSSHClient.HostName := '192.168.0.10';
ScSSHClient.Port := 22;
ScSSHClient.User := 'test';
ScSSHClient.Password := 'test';
ScSSHClient.HttpOptions.Enabled := True;
ScSSHClient.HttpOptions.Url := 'http://server/tunnel.php';
ScSSHClient.HttpOptions.ProxyOptions.Hostname := '10.0.0.1';
ScSSHClient.HttpOptions.ProxyOptions.Port := 808;
ScSSHClient.HttpOptions.ProxyOptions.Username := 'ProxyUser';
ScSSHClient.HttpOptions.ProxyOptions.Password := 'ProxyPassword';
ScSSHClient.Connect;
```

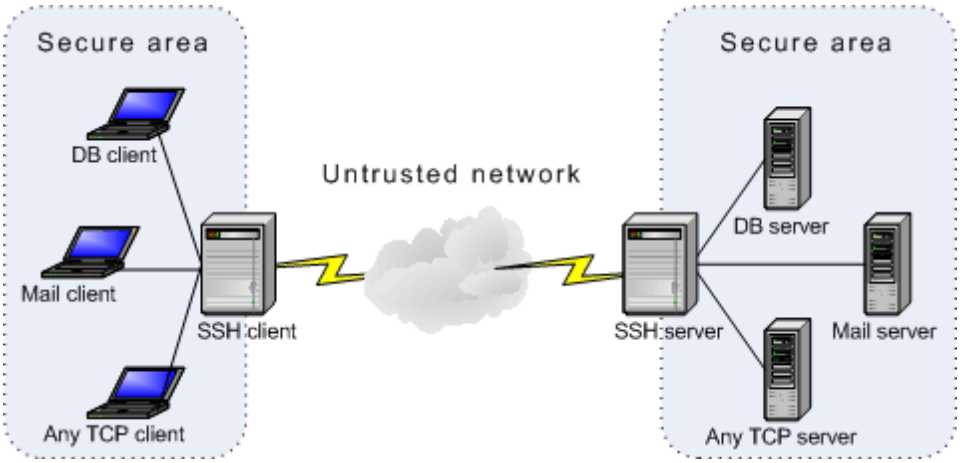
Note that setting parameters of ScSSHClient.HttpOptions.ProxyOptions automatically enables proxy server usage.

4.3 SSH specific

4.3.1 SSH-tunnel principles

SSH (Secure Shell) is the protocol for secure access to remote computers over insecure communication channels.

The general chart of computer ties when connecting through the SSH tunnel is presented below:



C1, C2, ..., Cn - computers from the client side of the SSH tunnel.

S1, S2, ..., Sn - computers from the server side of the SSH tunnel. This can be a database server, http server, or just other client computers.

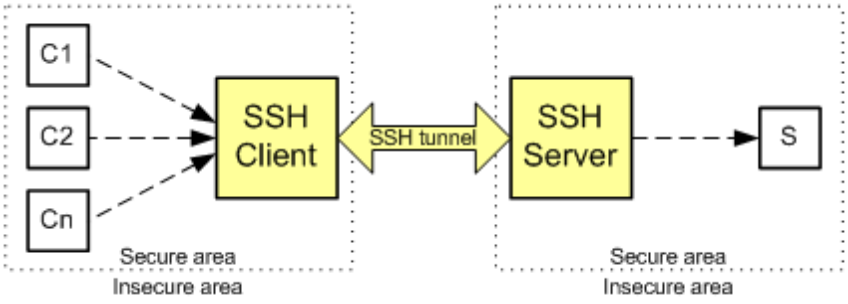
This connection method provides the secure connection between SSH client and SSH server that can go through insecure communication channels, like Internet.

Connections between Si and SSH server, and between SSH client and Ci are insecure, therefore they should go through secure communication channels. In the confluent case, Si and SSH server can be located on the same computer. The same is related to the SSH client and Ci.

The principle of working of the SSH connections is described below. The SSH server listens to the specified TCP/IP port. When SSH client tries to connect to this port, the SSH server authenticates the client. If the authentication passes, the connection is established. Then the client should create connections to Si objects. The SSH client sends an inquiry to establish necessary connection to SSH server, and the server establishes it.

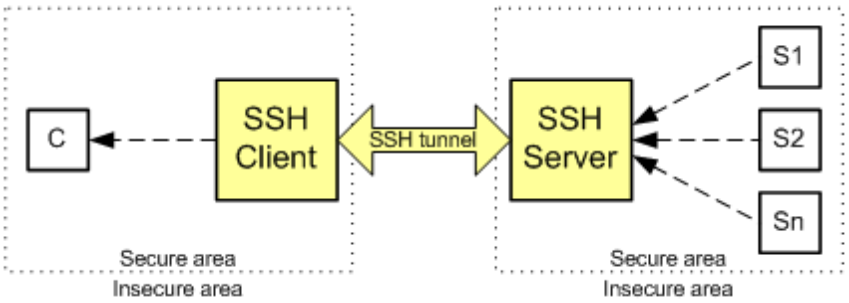
Also you can work in port forwarding mode. Port forwarding, or tunneling, is a way to forward otherwise insecure TCP traffic through SSH Secure Shell. There are two kinds of port forwarding: Local port forwarding and Remote port-forwarding.

Local port-forwarding



In this mode the SSH client listens the specified port. If a Ci computer from the client side of the tunnel needs to connect to the server S, Ci should connect to SSH client and the SSH client creates the secure channel to S via the SSH server.

Remote port-forwarding



In this mode SSH client sends a request to SSH to listen a specific port. If a Si computer from the server side wants to connect to the client C, Si should connect to the SSH server through the specified port, and the SSH server will create a secure channel to C through the SSH client.

4.3.2 Attack types and countermeasures

This article includes the general description of possible attack types on data transferred through insecure are, and recommendations for increasing data protection level.

1. Fitting seed for random number generator

This kind of attack allows attacker to decrypt data transferred through network and read it. The encrypted data can be intercepted and saved locally for future decryption.
SSH protocol binds each session key to the session by including random, session specific data in the hash used to produce session keys. It it necessary to ensure that all of the random numbers are of good quality, so the pseudo-random number generator should be cryptographically secure (i.e., its next output not easily guessed even when knowing all previous outputs).
SecureBridge uses a pseudo random number generator having high cryptographic security and high enough entropy. Also there is a possibility for user to assign the initial random sequence that will be used for generating random numbers. This sequence can be formed by using processor steps counter, system timer information, or information of random mouse movements or pressure of keyboard keys.

Recommendation: *To ensure high protection level, you should use a reliable initial sequence for random number generator. The sequence can be based, for example, on information about random mouse movements.*

2. Symmetric encryption algorithms cracking

SecureBridge uses different encryption algorithms, such as AES, 3DES, Blowfish, and Cast128. They have no vulnerabilities found till now. SecureBridge supports session key changing provided by SSH protocol. As a rule changing the session key after transferring of certain data amount is enough to prevent an attacker from cracking the key. CBC and CTR encryption modes (contain previous block encrypting output) of some ciphers is theoretically vulnerable to cipher-text attacks because of the high predictability of the start of packet sequence. However, this attack is deemed difficult and not considered fully practicable, especially if relatively long block sizes are used. In addition, CBC mode vulnerability can be reduced with insertion of packets, containing random data.

Recommendation: *Use the RekeyLimit property (Client.Options, Server.Options) for determining what data size should be transferred before session key is regenerated.*

3. Data substitution

This kind of attacks consists in the following: attacker gets access to the data packet transferred through insecure network area, changes it and, and transmits further. To determine whether the data was changed when transferring through the insecure area, data integrity checking methods are used. SecureBridge, within the bounds of the SSH protocol, inserts the MAC field into every sending packet. This field is calculated on basis of session key, packet sequence number and packet contents. SHA1 or SHA2 hashing algorithms, which are secure enough, is basically used for MAC field calculation. Because MACs use a 32-bit sequence number, they might start to leak information after 2**32 packets have been sent. Changing the session key after transfer of certain data amount increases degree of data protection.

4. Man-in-the-middle

There are some cases of man-in-the-middle attacks to consider.
If the attacker tries to connect between the client and the server before the client initiates the connection. When the client initiates session, attacker, that mimics SSH server, offers its server public key. If the client already has the server public key, it can verify the key sent by attacker, and warn the user about this spurious server public key. If the user does not accept this unverified key, attacker will not be able to make this attack work since the attacker will not be able to correctly sign packets containing this session-specific data from the server, since he does not have the private key of that server.
If the server public key was not securely delivered to the client and then verified, the client risk to accept the key substituted by the attacker, and the client cannot be sure that it is connected to the authentic server. This lets attacker to intercept and change the data transferred between the server and the client. Server administrators must make host key fingerprints available for checking by some means whose security does not rely on the integrity of the actual host keys. Possible mechanisms may include certification by a trusted certification authority (CA), secured Web pages, physical pieces of paper, etc. In summary, the use of this protocol without a reliable association of the binding between a host and its host keys is inherently insecure and is not recommended.

Recommendation: *It is necessary to care of safe server public keys transferring (see the Keys transferring topic).*

5. Denial of Service (DoS) attack

One of few weaknesses of the SSH protocol is vulnerability to Denial of Service attacks. Attacker can heap server with authentication requests that takes all server computational resources, and the server becomes unable to handle inquiries. One of the ways to resolve this problem is to allow connecting only from a subset of clients known to have valid users.

Recommendation: *Setup the MaxStartups server option of TScSSHServer, that specifies the maximum number of concurrent unauthenticated connections.*

6. Server substitution when password authentication

The password authentication mechanism assumes that the server has not been compromised. So, a violator can mimic an SSH server for the client that initiates the authentication procedure and recognize the password, that is fraught with serious consequences.
This vulnerability can be mitigated by using an alternative form of authentication, like public key authentication.

7. Client substitution when public key authentication

Public key authentication assumes that client public key passed to the server is not compromised. To ensure that the client public key accepted by the server is not substituted, it is recommended to use pass phrases on private keys, smart cards, or other technology.

Recommendation: *Keep the private client key in an encrypted form specifying the Password and Algorithm properties of a Storage object. The public key should be transferred to the server with maximum caution to prevent key substitution. (see the Keys transferring topic).*

4.3.3 Keys transferring

When creating a connection between an SSH client and an SSH server, often asymmetric encryption algorithms and keys are used for authentication (see the TScKey description). One of sides generates a pair of keys - private key and public key. The private key is used for signing data. Public key is used for signature verification. It should be passed to another side. It is important to take care about safe keys transferring.

Note: The private key should be protected and it should be known only to another side.

The is a possibility to intercept and substitute the public key when transferring.

- Key interception does not have any consequences. If a violator obtains a public key, he will not be able to read or change any data transferred through an SSH channel.
- When the public key is substituted, the violator will have a possibility to replace the SSH server with his own computer. This lets the violator to intercept and to change data that is transferred between the client and the server.
- If the public key of the client is substituted, the violator will have a possibility to replace the user's computer with his own computer and have an access to the SSH server.

There are several ways for safe keys transferring.

1. Key can be transferred though secure communication links. However, in most cases this method is unacceptable by technical reasons. Therefore other ways are used.
2. When obtaining a key form the other side, you should create a print from the key and verify it in any reliable way, for example by a phone. However, you should trust the person you are talking to. To get a finger print, you can use the GetFingerprint method of the TScKey class.
3. You can pass the signature of the key along with the key itself. The receiver verifies the key and the signature. If the signature is correct, the key is considered valid. In this case it is required both sides to have a certificate that will be used for signing the transferred key. This certificate can be obtained from one of two sources: A certificate authority (CA) such as VeriSign or GTE can provide certificates, or a privately controlled certificate server can issue certificates as well. To create a certificate, you should create a paid of keys. The private key remains on your computer, whereas the public key should be passed to CA for certification. After that the each side will be able to verify received certificate contacting with the corresponding CA.
4. One more way is to transfer the key along with its signature encrypted by asymmetric algorithms using certificates. For information on how to get certificates, see above.

4.3.4 Step-by-step tutorial

4.3.4.1 Configuring and starting the SSH server

When setting up the SSH server, first of all the storage should be set. It is used to store keys and user list that can connect to the server.

Storage setup

- Place the TScFileStorage or TScRegStorage component onto the form.
- Specify the path to store information about keys and users in the Path or KeyPath property.

First of all you should create a pair of keys that will be used for authentication server by the client.

Keys generating

- Open the editor of the storage object (double click on the component) and go to the Keys page.
- Press the New button to add a new key.
- Select an algorithm and a key length.
- Press the Generate button to generate a new key. A pair of keys must be created for the each used asymmetric algorithm.
- Pass the created public key to the server (see the Keys transferring topic).

It is required to add to the storage the information about the each user that will be connected to the SSH server.

Users creation

- Open the Users page of the storage editor.
- Press the New button to create a new user.
- Specify the user name.
- Select available authentication methods.
- If the authentication by a password is used, specify a password for the user. This password should by pretty complicated to be hard to crack it.
- If the authentication by a public key is used, specify the key for the user. This key is generated by the client and should be passed to the server carefully (see the Keys transferring topic). Press the "Import from..." button to import the key from a file.

SSH server setup

- Place the TScSSHServer component onto the form.
- Select required storage in the Storage property.
- Specify names of the generated server keys for the RSA or DSA algorithm in the KeyNameRSA or KeyNameDSA property correspondingly.
- Start the SSH server by setting the Active property to True.

4.3.4.2 SSH client setup

Use storage to store public server key, and private key when setting the SSH client. Private key is used in case of using the authentication method by key.

Storage setup

- Place the TScFileStorage or TScRegStorage component onto the form.
- Specify the path to be used to store information about keys in the Path / KeyPath property

- It is required to obtain server public key in order to authenticate the server. There are two ways to obtain this key:

1. The key can be previously obtained from the server as described in the Keys transferring topic.
2. Upon the first connect to the server you receive its public key that has to be stored in the storage for the future use to authenticate the server. However, in this case the key is passed through the unprotected channel and can be substituted by a malefactor.

- Add obtained key to the storage:

1. Open the component editor of the storage component by double click on the component and select the Keys tab.
2. Add a new key by pressing the New button.
3. Type the key name.
4. Import information from the obtained file by using the "Import from..." button.

If the authentication by a key is used, it is required to create the user key:

1. Open the component editor of the storage component by double click on the component and select the Keys tab.
2. Pressing the New button and type the key name.
3. Choose the algorithm to use and the needed key length.
4. Push the Generate button to generate a new key.
5. Export the public key and pass it to the server in order to the server be able to authenticate the client.

SSH client setup

- Place the TScSSHClient component onto the form.
- Select a storage object in the KeyStorage property.
- Specify the host name on which the SSH server is located in the HostName property.
- Specify the server public key in the HostKeyName property.
- Specify the user name in the User property.
- Choose authentication algorithm in the Authentication property.
- If authentication by password is used, specify password in the Password property.
- If authentication by key is used, specify the private key name in the PrivateKeyName property. The HostName value is used as a default key name. You can find steps to create a new key above in this topic.
- Establish connection to the server setting the Connected property to True.

You should create an SSH channel in order to exchange data with a remote host.

SSH channel setup

- Place the TScSSHChannel component onto the form.
- Select an SSH client in the Client property.
- Specify the host name in DestHost and the TCP/IP port number in DestPort to which the connection should be established.
- Specify the port number in SourcePort, data from which will be forwarded to the remote host to which the connection is established.
- Open the SSH channel setting the Connected property to True.

Random numbers generating

When establishing a connection to the SSH server, random numbers for creating session keys are generated. These keys will be used in the data encryption algorithms. For getting random numbers, pseudo random number generators are used. Before using the pseudo random number generator, you should initialize it, by setting a start seed value. This seed value can be obtained in different ways: using processor step counter, sound card noise, information of random mouse movements, or pressure of keyboard keys. However, the first two ways is not reliable.

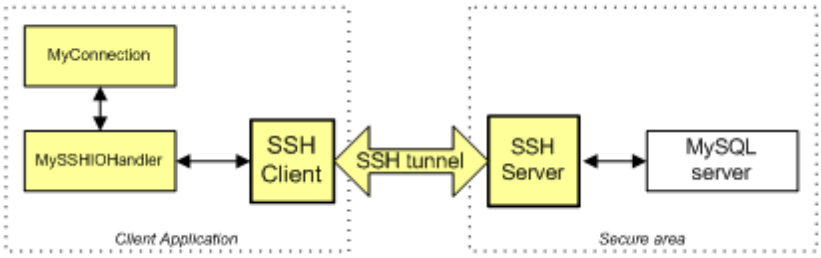
One of such ways is implemented in the SSHClient demo.

When using the SecureBridge component library, you should pass a sequence of the random values to the Randomize method of the global Random object.

4.3.4.3 MySQL Data Access Components integration

In order to create a secure connection via SSH tunnel between MySQL Data Access Components (MyDAC) and MySQL, you should use the TMySSHIOHandler component. This component is an adapter between a database client and an SSH client. The secure connection can be used to transfer data through unprotected communication channels, like Internet.

The communication chart between database client and database server with use of TMySSHIOHandler is presented in the following diagram:



Data exchange between TMyConnection and TScSSHClient which plays as an SSH client is safe because it is carried out by calling methods of TMySSHIOHandler within the single application.

Connection between SSH server and MySQL server is not secure, therefore you should take care that it goes through secure communication channels.

Step-by-step setup of MySSHIOHandler

- Place the TScSSHClient component onto the form and setup it to connect to the SSH server as described in the Client setup topic.
- Place the TMySSHIOHandler component onto the form.
- Select the TScSSHClient object in the Client property.
- Place the TMyConnection component onto the form and setup it to connect to the MySQL server.
- Assign the TMySSHIOHandler object to the IOHandler property of TMyConnection.
- Connection to MySQL server by setting TMyConnection.Connected to True.

4.4 SSL specific

4.4.1 SSL/TLS principles

SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are protocols for secure access to remote computers over insecure communication channels.

The SSL/TLS protocols run above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

- SSL server authentication allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.
- SSL client authentication allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.
- An encrypted SSL connection requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering--that is, for automatically determining whether the data has been altered in transit.

4.4.2 Step-by-step tutorial

4.4.2.1 SSL/TLS client setup

Use storage to store server and client certificates when setting the TLS/SSL client. Server certificate is used for the authentication of a TLS/SSL server. Client certificates can be used for the client authentication. In this case the certificate must contain the private key.

Storage setup

- Place one of the storage components onto the form: TScCryptoAPIStorage, TScFileStorage, or TScRegStorage.
- Specify the path to be used to store information about certificates in the CertStoreName / Path / KeyPath property (depending on the the storage component type).
- Add server and client certificates to the storage:
  1. Open the editor of the storage component by double click on it, and select the Certificates tab.
  2. Add a new certificate by pressing the New button.
  3. Type the certificate name.
  4. Import information from a file that contains a certificate by using the "Import from..." button.

SSL/TLS client setup

- Place the TScSSLClient component onto the form.
- In the HostName property specify the name of the host on which the TLS/SSL server is located.
- In the Port property specify the port number for TCP/IP connection with the TLS/SSL server.
- Select already created storage object in the Storage property.
- Specify the server certificate in the CACertName property.
- If necessary, specify the client certificate in the CertName property.
- Establish connection to the server setting the Connected property to True.
- To make the connection secure, turn the IsSecure property to True.

Random numbers generating

When establishing connection to an TLS/SSL server, random numbers for creating session keys are generated. These keys will be used in the data encryption algorithms. For getting random numbers, pseudo random number generators are used. Before using the pseudo random number generator, you should initialize it, by setting a start seed value. This seed value can be obtained in different ways: using processor step counter, sound card noise, information of random mouse movements, or pressure of keyboard keys. However, the first two ways is not reliable.

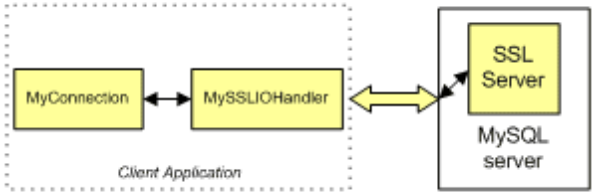
One of such ways is implemented in the SSHClient demo.

When using the SecureBridge component library, you should pass a sequence of the random values to the Randomize method of the global Random object.

4.4.2.2 MySQL Data Access Components integration

In order to create a secure connection via TLS/SSL between MySQL Data Access Components (MyDAC) and MySQL server, you should use the TMySSLIOHandler component. This component is an adapter between a database client and an TLS/SSL client. The secure connection can be used to transfer data through unprotected communication channels, like Internet.

The communication chart between database client and database server with use of TMySSLIOHandler is presented in the following diagram:



Data exchange between TMyConnection and TScSSLClient which plays as an TLS/SSL client is safe because it is carried out by calling methods of TMySSLIOHandler within the single application.

Step-by-step setup of MySSLIOHandler

- Place the TMySSLIOHandler component onto the form.
- Select a storage object in the Storage property. More information about storage setup you will find in

the SSL client setup topic.
- Specify the server certificate in the CACertName property.
- Specify the client certificate in the CertName property.
- Place the TMyConnection component onto the form and setup it to connect to the MySQL server.
- Assign the TMySSLIOHandler object to the IOHandler property of TMyConnection.
- Connect to MySQL server by setting TMyConnection.Connected to True.

5 SecureBridge Alphabetical Object and Component Listing

5.1 EScError

5.1.1 Description

Unit

ScUtils

Description

EScError arise, when an error occurs in SecureBridge classes, for example when interaction between SSH client and SSH server, TLS/SSL client and server, or when working with keys.

Use **EScError** in exception-handling blocks.

5.1.2 Properties

5.1.2.1 ErrorCode

```
property ErrorCode: TScErrorCode;
```

Description

The **ErrorCode** property holds the code of the error, which occurs in various cases in all SecureBridge components.

This property is read-only.

5.2 EScFTPError

5.2.1 Description

Unit

ScFTPClient

Description

EScFTPError arises when an error occurs while executing any command to the FTP/FTPS server.

The FTPErrorCode property contains the code of the error returned by the server.

Use **EScFTPError** in exception-handling blocks.

See also

TScFTPClient

5.2.2 Properties

5.2.2.1 FTPErrorCode

```
property FTPErrorCode: integer;
```

Description

The **FTPErrorCode** property holds the code of the error returned by an FTP server.

This property is read-only.

5.3 EScSFTPError

5.3.1 Description

Unit

ScSFTPUtils

Description

EScSFTPError arises, when the SFTP server returns an error and client is in the NonBlocking = False mode.

The ErrorCode property contains the code of the error returned by the server.

Use **EScSFTPError** in exception-handling blocks.

See also

TScSFTPClient

5.3.2 Properties

5.3.2.1 ErrorCode

```
property ErrorCode: integer;
```

Description

The **ErrorCode** property holds the code of the error returned by an SFTP server.
This property is read-only.

Here is a list of the constants of possible error codes:

```
SSH_FX_OK                          = 0;
SSH_FX_EOF                         = 1;
SSH_FX_NO_SUCH_FILE                = 2;
SSH_FX_PERMISSION_DENIED           = 3;
SSH_FX_FAILURE                     = 4;
SSH_FX_BAD_MESSAGE                 = 5;
SSH_FX_NO_CONNECTION               = 6;
SSH_FX_CONNECTION_LOST             = 7;
SSH_FX_OP_UNSUPPORTED              = 8;
SSH_FX_INVALID_HANDLE              = 9;
SSH_FX_NO_SUCH_PATH                = 10;
SSH_FX_FILE_ALREADY_EXISTS         = 11;
SSH_FX_WRITE_PROTECT               = 12;
SSH_FX_NO_MEDIA                    = 13;
SSH_FX_NO_SPACE_ON_FILESYSTEM      = 14;
SSH_FX_QUOTA_EXCEEDED              = 15;
SSH_FX_UNKNOWN_PRINCIPAL           = 16;
SSH_FX_LOCK_CONFLICT               = 17;
SSH_FX_DIR_NOT_EMPTY               = 18;
SSH_FX_NOT_A_DIRECTORY             = 19;
SSH_FX_INVALID_FILENAME            = 20;
SSH_FX_LINK_LOOP                   = 21;
SSH_FX_CANNOT_DELETE               = 22;
SSH_FX_INVALID_PARAMETER           = 23;
SSH_FX_FILE_IS_A_DIRECTORY         = 24;
SSH_FX_BYTE_RANGE_LOCK_CONFLICT    = 25;
SSH_FX_BYTE_RANGE_LOCK_REFUSED     = 26;
SSH_FX_DELETE_PENDING              = 27;
SSH_FX_FILE_CORRUPT                = 28;
SSH_FX_OWNER_INVALID               = 29;
SSH_FX_GROUP_INVALID               = 30;
```

You can find more detailed information about these error codes by the following link: https://tools.ietf.org/html/draft-ietf-secsh-filexfer-13

5.4 EScSMTPError

5.4.1 Description

Unit

ScSMTPClient

Description

EScSMTPError arises when an error occurs while executing a command on the SMTP/SMTPS server.

The SMTPErrorCode property contains the code of the error returned by the server.

Use **EScSMTPError** in exception-handling blocks.

See also

TScSMTPClient

5.4.2 Properties

5.4.2.1 SMTPErrorCode

```
property SMTPErrorCode: integer;
```

Description

The **SMTPErrorCode** property holds the code of the error returned by an SMTP server.

This property is read-only.

For more information about the error codes, see the SMTP RFC.

5.5 HttpException

5.5.1 Description

Unit

ScHttp

Description

HttpException arises, when errors occur in the TScHttpWebRequest.GetResponse method while accessing a resource.

The StatusCode property contains a TScHttpStatusCode value that indicates the source of the error.

The ServerMessage property holds the string message of the error returned by the HTTP server.

Use **HttpException** in exception-handling blocks.

See also

TScHttpWebRequest.GetResponse

5.5.2 Properties

5.5.2.1 ServerMessage

```
property ServerMessage: string;
```

Description

The **ServerMessage** property holds the string message of the error returned by the HTTP server.

This property is read-only.

5.5.2.2 StatusCode

```
property StatusCode: TScHttpStatusCode;
```

Description

The **StatusCode** property holds a value that indicates the status of the HTTP response. The expected values for status are defined in the TScHttpStatusCode enumeration.

This property is read-only.

5.6 HubException

5.6.1 Description

Unit

ScSignalRProtocol

Description

HubException is raised when an error occurs in the TScHubConnection component while processing any message from the SignalR server.

Use **HubException** in exception-handling blocks.

See also

TScHubConnection

5.7 WebSocketException

5.7.1 Description

Unit

ScWebSocketClient

Description

WebSocketException arises when errors occur in the TScWebSocketClient component during the WebSocket session.

Use **WebSocketException** in exception-handling blocks.

See also

TScWebSocketClient

5.8 TScCertificateExtension

5.8.1 Description

Unit

ScCertificateExts

Description

The **TScCertificateExtension** class is used for certificate extensions support. Certificate extensions represent information fields that contain an additional certificate information. Certificate extensions let extending abilities of the basic data standard of the X.509 certificate. Several fields of the extension contain an additional information about certificate identification. Other fields contain an additional information about certificate encryption abilities.

In its most basic form, an X.509 extension has an object identifier (Oid), a boolean value describing whether the extension is considered critical or not (Critical), and ASN-encoded data (RawData).

This class is used as a base class for other certificate extensions classes. Also it can be used to specify non-standard certificate extensions.

See Also

TScCertificate.Extensions

TScCertAlternativeNameExtension

TScCertAuthorityKeyIdExtension

TScCertBasicConstraintsExtension

TScCertExtendedKeyUsageExtension
TScCertKeyUsageExtension
TScCertPoliciesExtension
TScCertPolicyMappingsExtension
TScCertSubjectDirectoryAttributesExtension
TScCertSubjectKeyIdExtension

5.8.2 Properties

5.8.2.1 Critical

```
property Critical: Boolean;
```

Description

Use the **Critical** property to determine whether the certificate extension is critical.
This property is read-only.

5.8.2.2 Oid

```
property Oid: TScOid;
```

Description

Use the **Oid** property to read the Object Identifier of the certificate extension.
This property is read-only.

5.8.2.3 RawData

```
property RawData: TBytes;
```

Description

The **RawData** property is a byte array that represents the body of the certificate extension.
RawData contains the Abstract Syntax Notation One (ASN.1) data in BER format.
This property is read-only.

5.8.3 Methods

5.8.3.1 Create

```
constructor Create(const OId: string; Critical: boolean; const DERValue:
```

```
TBytes); virtual;
```

Description

Create **TScCertificateExtension** instance.

The `OId` parameter is an object that represents Object Identifier of the certificate extension. The OId property is set from the value of this parameter.

The `Critical` parameter is a boolean value that determines whether the extension is critical. The Critical property is set from the value of this parameter.

The `DERValue` parameter is a byte array that represents the body of the extension. `DERValue` should contain the Abstract Syntax Notation One (ASN.1) data in BER format. The RawData property is set from the value of this parameter.

5.8.3.2 ToString

```
function ToString: string; virtual;
```

Description

Use the **ToString** method to display the certificate extension data in text format.

5.9 TScCertAlternativeNameExtension

5.9.1 Description

Unit

ScCertificateExts

Description

The **TScCertAlternativeNameExtension** class represents the subject and issuer alternative names extensions, that are used to associate Internet style identities with the certificate subject/issuer. **TScCertAlternativeNameExtension** contains a list of the TScGeneralName objects.

The following paragraph is taken from RFC 5280, section 4.2.1.6:

"The subject alternative name extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate. Defined options include an Internet electronic mail address, a DNS name, an IP address, and a Uniform Resource Identifier (URI). Other options exist, including completely local definitions.

Multiple name forms, and multiple instances of each name form, MAY be included. Whenever such identities are to be bound into a certificate, the subject alternative name (or issuer alternative name) extension MUST be used; however, a DNS name MAY also be represented in the subject field using the domainComponent attribute."

See Also

TScCertificateExtension

TScGeneralName

5.9.2 Properties

5.9.2.1 GeneralNames

```
property GeneralNames: TScGeneralNames;
```

Description

The **GeneralNames** property contains Alternative Name information in form of list of TScGeneralName.

This property is read-only.

5.10 TScCertAuthorityInfoAccessExtension

5.10.1 Description

Unit

ScCertificateExts

Description

The **TScCertAuthorityInfoAccessExtension** class represents the authority information access extension that contains a list of the TScInfoAccess objects, that indicate how to access information and services for the issuer of the certificate.

The following paragraph is taken from RFC 5280, section 4.2.2.1:

"The authority information access extension indicates how to access information and services for the issuer of the certificate in which the extension appears. Information and services may include on-line validation services and CA policy data.

Each entry in the sequence AuthorityInfo describes the format and location of additional information provided by the issuer of the certificate in which this extension appears. The type and format of the information are specified by the accessMethod field; the accessLocation field specifies the location of the information. The retrieval mechanism may be implied by the accessMethod or specified by accessLocation."

See Also

TScCertificateExtension

TScInfoAccess

5.10.2 Properties

5.10.2.1 AuthorityInfoAccessList

```
property AuthorityInfoAccessList: TScInfoAccessList;
```

Description

AuthorityInfoAccessList maintains a list of the TScInfoAccess object references, that indicate how to access information and services for the issuer of the certificate.

This property is read-only.

5.11 TScCertAuthorityKeyIdExtension

5.11.1 Description

Unit

ScCertificateExts

Description

The **TScCertAuthorityKeyIdExtension** class represents the authority key identifier extension that is used to keep a "Fingerprint" of issuer's public key in order to distinguish different certificates which belong to the same issuer.

The following paragraph is taken from RFC 5280, part 4.2.1.1:

"The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a certificate. This extension is used where an issuer has multiple signing keys (either due to multiple concurrent key pairs or due to changeover). The identification MAY be based on either the key identifier (the subject key identifier in the issuer's certificate) or the issuer name and serial number."

See Also

TScCertificateExtension

5.11.2 Properties

5.11.2.1 CertIssuers

```
property CertIssuers: TScGeneralNames;
```

Description

The **CertIssuers** property describes the issuer of the certificate in form of GeneralName.

This property is read-only.

5.11.2.2 CertSerialNumber

```
property CertSerialNumber: string;
```

Description

The **CertSerialNumber** property contains the serial number of issuer's certificate.

This property is read-only.

5.11.2.3 KeyIdentifier

```
property KeyIdentifier: string;
```

Description

The **KeyIdentifier** property contains the key identifier of issuer's certificate. Key identifier is usually a fingerprint (message digest), calculated from issuer's public key.

This property is read-only.

5.12 TScCertBasicConstraintsExtension

5.12.1 Description

Unit

ScCertificateExts

Description

The **TScCertBasicConstraintsExtension** class represents the basic constraints extension that provides properties to describe the basic constraint set on a certificate. These constraints are used during the certificate chain verification process.

The following paragraph is taken from RFC 5280, section 4.2.1.9:

"The basic constraints extension identifies whether the subject of the certificate is a CA and the maximum depth of valid certification paths that include this certificate.

The cA boolean indicates whether the certified public key may be used to verify certificate signatures.

The pathLenConstraint field is meaningful only if the cA boolean is asserted and the key usage extension, if present, asserts the keyCertSign bit. In this case, it gives the maximum number of non-self-issued intermediate certificates that may follow this certificate in a valid certification path. A pathLenConstraint of zero indicates that no non-self-issued intermediate CA certificates may follow in a valid certification path. Where it appears, the pathLenConstraint field MUST be greater than or equal

to zero. Where pathLenConstraint does not appear, no limit is imposed."

See Also

TScCertificateExtension

5.12.2 Properties

5.12.2.1 CertificateAuthority

```
property CertificateAuthority: Boolean;
```

Description

Use the **CertificateAuthority** property to determine if the certificate is a certification authority (CA) certificate. **CertificateAuthority** is set to True for all CA certificates.

This property is read-only.

5.12.2.2 HasPathLengthConstraint

```
property HasPathLengthConstraint: Boolean;
```

Description

A certificate issuer can restrict the number of levels in a certificate path. The **HasPathLengthConstraint** property indicates whether the certificate has this restriction. If this value is True, you can use the PathLengthConstraint property to determine the number of levels allowed.

This property is read-only.

See Also

PathLengthConstraint

5.12.2.3 PathLengthConstraint

```
property PathLengthConstraint: Integer;
```

Description

If a certificate has a constraint on the number of levels in the certificate path, the **PathLengthConstraint** property indicates how many levels are allowed.

PathLengthConstraint must be greater than the number of already processed CA certificates, starting with the end-entity certificate and moving up the chain. This constraint can be omitted if all of the higher level CA certificates in the chain does not include this constraint when the extension is present.

This property is read-only.

See Also

HasPathLengthConstraint

5.13 TScCertCRLDistributionPointsExtension

5.13.1 Description

Unit

ScCertificateExts

Description

The **TScCertCRLDistributionPointsExtension** class represents the distribution points extension that contains a list of the TScCRLDistributionPoint objects, that identifie how CRL information is obtained.

The following paragraph is taken from RFC 5280, section 4.2.1.13:

"The CRL distribution points extension identifies how CRL information is obtained. The cRLDistributionPoints extension is a sequence of DistributionPoint. A DistributionPoint consists of three fields, each of which is optional: DistributionPoint, Reasons, and CRLIssuer. While each of these fields is optional, a DistributionPoint MUST NOT consist of only the reasons field; either distributionPoint or cRLIssuer MUST be present. If the certificate issuer is not the CRL issuer, then the cRLIssuer field MUST be present and contain the Name of the CRL issuer. If the certificate issuer is also the CRL issuer, then conforming CAs MUST omit the cRLIssuer field and MUST include the distributionPoint field."

See Also

TScCertificateExtension

TScCertFreshestCRLExtension

TScCRLDistributionPoint

5.13.2 Properties

5.13.2.1 CRLDistributionPoints

```
property CRLDistributionPoints: TScCRLDistributionPointList;
```

Description

CRLDistributionPoints maintains a list of the TScCRLDistributionPoint object references, those contain the information about distribution point, that identifies how CRL information is obtained.

This property is read-only.

5.14 TScCertExtendedKeyUsageExtension

5.14.1 Description

Unit

ScCertificateExts

Description

The **TScCertExtendedKeyUsageExtension** class represents the extended key usage extension that is a collection of object identifiers (OIDs) that indicate the applications that use the key.

The extended key usage extension indicates the purposes for which the certified public key may be used. These purposes may be in addition to or in place of the basic purposes indicated in Certificate Key Usage extension.
The extended key usage must include Online Certificate Status Protocol (OCSP) signing in an OCSP responder's certificate. The exception is that the CA signing key that signed the certificates validated by the responder is also the OCSP signing key. The OCSP responder's certificate must be issued directly by the CA that signs certificates the responder will validate.
The Certificate Key Usage, Certificate Extended Key Usage, and Certificate Basic Constraints extensions act together to define the purposes for which the certificate is intended to be used. Applications can use these extensions to disallow the use of a certificate in inappropriate contexts.

This extension is specified in RFC 5280 section 4.2.1.12.

See Also

TScCertificateExtension

TScCertKeyUsageExtension

5.14.2 Properties

5.14.2.1 ExtendedKeyUsages

```
property ExtendedKeyUsages: TScOIds;
```

Description

Gets the collection of object identifiers (OIDs) that indicate the applications that use the key.

Use **ExtendedKeyUsages[Index]** to obtain a pointer to a specific TScOid. The `Index` parameter indicates the index of the object identifier. 0 is the index of the first object identifier.

This property is read-only.

See Also

TScOid

5.15 TScCertIssuerAlternativeNameExtension

5.15.1 Description

Unit

ScCertificateExts

Description

The **TScCertIssuerAlternativeNameExtension** class represents the issuer alternative names extensions, that are used to associate Internet style identities with the certificate issuer. **TScCertIssuerAlternativeNameExtension** contains a list of the TScGeneralName objects.

The following paragraph is taken from RFC 5280, section 4.2.1.6:

"The subject alternative name extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate. Defined options include an Internet electronic mail address, a DNS name, an IP address, and a Uniform Resource Identifier (URI). Other options exist, including completely local definitions.

Multiple name forms, and multiple instances of each name form, MAY be included. Whenever such identities are to be bound into a certificate, the subject alternative name (or issuer alternative name) extension MUST be used; however, a DNS name MAY also be represented in the subject field using the domainComponent attribute."

See Also

TScCertSubjectAlternativeNameExtension

TScCertificateExtension

TScGeneralName

5.16 TScCertFreshestCRLExtension

5.16.1 Description

Unit

ScBridge

Description

The **TScCertFreshestCRLExtension** class represents the freshest CRL extension that contains a list of the TScCRLDistributionPoint objects, that identifie how delta CRL information is obtained.

TScCertFreshestCRLExtension is inherited from the TScCertCRLDistributionPointsExtension class and has the same interface.

See Also

TScCertificateExtension

TScCertCRLDistributionPointsExtension

TScCRLDistributionPoint

5.17 TScCertKeyUsageExtension

5.17.1 Description

Unit

ScCertificateExts

Description

The **TScCertKeyUsageExtension** class represents the certificate key usage extension that uses the flags in the TScKeyUsageFlag enumeration to define key usage.

A certificate lets a subject to perform certain tasks. In order to control usage of a certificate out of designated scopes, the corresponding restrictions are automatically included in the certificate. The Key Usage extension is a restriction method that determines, for what purposes the certificate can be used. This lets to produce certificates that can be used both for tasks restricted by certain scopes, and for different tasks.

This extension is specified in RFC 5280 section 4.2.1.3.

See Also

TScCertificateExtension

5.17.2 Properties

5.17.2.1 KeyUsages

```
property KeyUsages: TScKeyUsageFlags;
```

Description

The **KeyUsages** property returns a value from the TScKeyUsageFlags enumeration that indicates how the certificate key can be used.

This property is read-only.

5.18 TScCertPoliciesExtension

5.18.1 Description

Unit

ScCertificateExts

Description

The **TScCertPoliciesExtension** class represents the certificate policies extension that contains a list of the TScPolicy objects.

The following paragraph is taken from RFC 5280, section 4.2.1.4:

"The certificate policies extension contains a sequence of one or more policy information terms, each of which consists of an object identifier (OID) and optional qualifiers. Optional qualifiers, which MAY be present, are not expected to change the definition of the policy. A certificate policy OID MUST NOT appear more than once in a certificate policies extension.

Applications with specific policy requirements are expected to have a list of those policies that they will accept and to compare the policy OIDs in the certificate to that list. If this extension is critical, the path validation software MUST be able to interpret this extension (including the optional qualifier), or MUST reject the certificate. "

See Also

TScCertificateExtension

TScPolicy

5.18.2 Properties

5.18.2.1 Policies

```
property Policies: TScPolicyList;
```

Description

Policies maintains a list of the TScPolicy object references, those contain the information about certificate policies.

This property is read-only.

5.19 TScCertPolicyMappingsExtension

5.19.1 Description

Unit

ScCertificateExts

Description

The **TScCertPolicyMappingsExtension** class represents the policy mappings extension that contains a list of the TScPolicyMapping objects.

The following paragraph is taken from RFC 5280, section 4.2.1.5:

"The Policy Mappings extension is used in CA certificates. It lists one or more pairs of OIDs; each pair includes an issuerDomainPolicy and a subjectDomainPolicy. The pairing indicates the issuing CA considers its issuerDomainPolicy equivalent to the subject CA's subjectDomainPolicy.

The issuing CA's users might accept an issuerDomainPolicy for certain applications. The policy mapping defines the list of policies associated with the subject CA that may be accepted as comparable to the issuerDomainPolicy.

Each issuerDomainPolicy named in the policy mappings extension SHOULD also be asserted in a certificate policies extension in the same certificate. Policies MUST NOT be mapped either to or from the special value anyPolicy."

See Also

TScCertificateExtension

TScPolicyMapping

5.19.2 Properties

5.19.2.1 PolicyMappings

```
property PolicyMappings: TScPolicyMappingList;
```

Description

PolicyMappings maintains a list of the TScPolicyMapping object references, those contain the information about policy mappings.

This property is read-only.

5.20 TScCertSubjectAlternativeNameExtension

5.20.1 Description

Unit

ScCertificateExts

Description

The **TScCertSubjectAlternativeNameExtension** class represents the subject alternative names extensions, that are used to associate Internet style identities with the certificate subject. **TScCertSubjectAlternativeNameExtension** contains a list of the TScGeneralName objects.

The following paragraph is taken from RFC 5280, section 4.2.1.6:

"The subject alternative name extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate. Defined options include an Internet electronic mail address, a DNS name, an IP address, and a Uniform Resource Identifier (URI). Other options exist, including completely local definitions.

Multiple name forms, and multiple instances of each name form, MAY be included. Whenever such identities are to be bound into a certificate, the subject alternative name (or issuer alternative name) extension MUST be used; however, a DNS name MAY also be represented in the subject field using the domainComponent attribute."

See Also

TScCertIssuerAlternativeNameExtension

TScCertificateExtension

TScGeneralName

5.21 TScCertSubjectDirectoryAttributesExtension

5.21.1 Description

Unit

ScCertificateExts

Description

The **TScCertSubjectDirectoryAttributesExtension** class represents the subject directory attributes extension that contains a list of the TScPKCS7Attribute objects.

The following paragraph is taken from RFC 5280, section 4.2.1.8:

"The subject directory attributes extension is used to convey identification attributes (e.g., nationality) of the subject. The extension is defined as a sequence of one or more attributes."

See Also

TScCertificateExtension

TScPKCS7Attribute

5.21.2 Properties

5.21.2.1 DirectoryAttributes

```
property DirectoryAttributes: TScPKCS7Attributes;
```

Description

DirectoryAttributes maintains a list of the TScPKCS7Attribute object references, those represent subject identification attributes.

This property is read-only.

5.22 TScCertSubjectInfoAccessExtension

5.22.1 Description

Unit

ScCertificateExts

Description

The **TScCertSubjectInfoAccessExtension** class represents the subject information access extension that contains a list of the TScInfoAccess objects, that indicate how to access information and services for the subject of the certificate.

The following paragraph is taken from RFC 5280, section 4.2.2.2:

"The subject information access extension indicates how to access information and services for the subject of the certificate in which the extension appears. When the subject is a CA, information and services may include certificate validation services and CA policy data. When the subject is an end entity, the information describes the type of services offered and how to access them. In this case, the contents of this extension are defined in the protocol specifications for the supported services."

See Also

TScCertificateExtension

TScInfoAccess

5.22.2 Properties

5.22.2.1 SubjectInfoAccessList

```
property SubjectInfoAccessList: TScInfoAccessList;
```

Description

SubjectInfoAccessList maintains a list of the TScInfoAccess object references, that indicate how to access information and services for the subject of the certificate.

This property is read-only.

5.23 TScCertSubjectKeyIdExtension

5.23.1 Description

Unit

ScCertificateExts

Description

The **TScCertSubjectKeyIdExtension** class defines a string that identifies a certificate's subject key identifier (SKI).

The SKI provides a unique identification for the subject of the certificate. The SKI is often used when working with XML digital signing.
The SKI extension identifies the public key certified by this certificate. This extension provides a way of distinguishing public keys if more than one is available for a given subject name.

This extension is specified in RFC 5280 section 4.2.1.2.

See Also

TScCertificateExtension

5.23.2 Properties

5.23.2.1 SubjectKeyIdentifier

```
property SubjectKeyIdentifier: string;
```

Description

SubjectKeyIdentifier is a string, encoded in hexadecimal format, that represents the subject key identifier (SKI). The SKI provides a unique identification for the subject of the certificate. The SKI is often used when working with XML digital signing.

This property is read-only.

5.24 TScCRLCertificateIssuerExtension

5.24.1 Description

Unit

ScCertificateExts

Description

The **TScCRLCertificateIssuerExtension** class represents the certificate issuer extension, that identifies the certificate issuer associated with an entry in an indirect CRL.

The following paragraph is taken from RFC 5280, section 5.3.3:

"This CRL entry extension identifies the certificate issuer associated with an entry in an indirect CRL, that is, a CRL that has the indirectCRL indicator set in its issuing distribution point extension. When present, the certificate issuer CRL entry extension includes one or more names from the issuer field and/or issuer alternative name extension of the certificate that corresponds to the CRL entry."

See Also

TScCertificateExtension

5.24.2 Properties

5.24.2.1 CertificateIssuer

```
property CertificateIssuer: TScGeneralNames;
```

Description

The **CertificateIssuer** property specifies the certificate issuer associated with an entry in an indirect CRL.

Conforming CRL issuers includes in this extension the distinguished name (DN) from the issuer field of the certificate that corresponds to this CRL entry.

This property is read-only.

5.25 TScCRLDeltaIndicatorExtension

5.25.1 Description

Unit

ScCertificateExts

Description

The **TScCRLDeltaIndicatorExtension** class represents the delta CRL indicator extension, that identifies a CRL as being a delta CRL.

The following paragraph is taken from RFC 5280, section 5.2.4:

"The delta CRL indicator is a critical CRL extension that identifies a CRL as being a delta CRL. Delta CRLs contain updates to revocation information previously distributed, rather than all the information that would appear in a complete CRL. The use of delta CRLs can significantly reduce network load and processing time in some environments. Delta CRLs are generally smaller than the CRLs they update, so applications that obtain delta CRLs consume less network bandwidth than applications that obtain the corresponding complete CRLs. Applications that store revocation information in a format other than the CRL structure can add new revocation information to the local database without reprocessing information."

See Also

TScCertificateExtension

TScCRLNumberExtension

5.25.2 Properties

5.25.2.1 BaseCRLNumber

```
property BaseCRLNumber: string;
```

Description

The **BaseCRLNumber** property identifies the CRL number, complete for a given scope, that was used as the starting point in the generation of this delta CRL.

This property is read-only.

5.26 TScCRLInvalidityDateExtension

5.26.1 Description

Unit

ScCertificateExts

Description

The **TScCRLInvalidityDateExtension** class represents the invalidity date extension, that provides the date on which the private key was compromised or the certificate otherwise became invalid.

The following paragraph is taken from RFC 5280, section 5.3.2:

"The invalidity date is a non-critical CRL entry extension that provides the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid. This date may be earlier than the revocation date in the CRL entry, which is the date at which the CA processed the revocation. When a revocation is first posted by a CRL issuer in a CRL, the invalidity date may precede the date of issue of earlier CRLs, but the revocation date SHOULD NOT precede the date of issue of earlier CRLs. Whenever this information is available, CRL issuers are strongly encouraged to share it with CRL users."

See Also

TScCertificateExtension

5.26.2 Properties

5.26.2.1 InvalidityDate

```
property InvalidityDate: TDateTime;
```

Description

The **InvalidityDate** property specifies the date on which the private key was compromised or the

certificate otherwise became invalid.

This property is read-only.

5.27 TScCRLIssuingDistributionPointExtension

5.27.1 Description

Unit

ScCertificateExts

Description

The **TScCRLIssuingDistributionPointExtension** class represents the issuing distribution point extension, that identifies the CRL distribution point and scope for a particular CRL.

The following paragraph is taken from RFC 5280, section 5.2.5:

"The issuing distribution point is a critical CRL extension that identifies the CRL distribution point and scope for a particular CRL, and it indicates whether the CRL covers revocation for end entity certificates only, CA certificates only, attribute certificates only, or a limited set of reason codes."

See Also

TScCertificateExtension

TScCRLNumberExtension

5.27.2 Properties

5.27.2.1 DistributionPointName

```
property DistributionPointName: TScGeneralNames;
```

Description

The **DistributionPointName** property contains a sequence of general names, each name describes a different mechanism to obtain the same CRL. For example, the same CRL could be available for retrieval through both LDAP and HTTP.

This property is read-only.

5.27.2.2 IndirectCRL

```
property IndirectCRL: boolean;
```

Description

The **IndirectCRL** property specifies if the scope of the CRL only includes certificates issued by the CRL issuer.

If the scope of the CRL only includes certificates issued by the CRL issuer, then **IndirectCRL** is set to False. Otherwise, if the scope of the CRL includes certificates issued by one or more authorities other than the CRL issuer, the **IndirectCRL** property is set to True.

This property is read-only.

5.27.2.3 OnlyContainsAttributeCerts

```
property OnlyContainsAttributeCerts: boolean;
```

Description

The **OnlyContainsAttributeCerts** property indicates whether the CRL covers revocation for attribute certificates only.

If the scope of the CRL covers revocation for attribute certificates only, then **OnlyContainsAttributeCerts** is set to True. Otherwise, the **OnlyContainsAttributeCerts** property is set to False.

This property is read-only.

5.27.2.4 OnlyContainsCACerts

```
property OnlyContainsCACerts: boolean;
```

Description

The **OnlyContainsCACerts** property indicates whether the CRL covers revocation for CA certificates only.

If the scope of the CRL covers revocation for CA certificates only, then **OnlyContainsCACerts** is set to True. Otherwise, the **OnlyContainsCACerts** property is set to False.

This property is read-only.

5.27.2.5 OnlyContainsUserCerts

```
property OnlyContainsUserCerts: boolean;
```

Description

The **OnlyContainsUserCerts** property indicates whether the CRL covers revocation for end entity certificates only.

If the scope of the CRL covers revocation for end entity certificates only, then **OnlyContainsUserCerts** is set to True. Otherwise, the **OnlyContainsUserCerts** property is set to False.

This property is read-only.

5.27.2.6 OnlySomeReasons

```
property OnlySomeReasons: TScCRLReasons;
```

Description

The **OnlySomeReasons** property indicates whether the CRL covers revocation for a limited set of reason codes. **OnlySomeReasons** contains set of reason codes that the scope of the CRL covers revocation for.

This property is read-only.

5.28 TScCRLNumberExtension

5.28.1 Description

Unit

ScCertificateExts

Description

The **TScCRLNumberExtension** class represents the CRL number extension, that conveys a monotonically increasing sequence number for a given CRL scope and CRL issuer.

The following paragraph is taken from RFC 5280, section 5.2.3:

"The CRL number is a non-critical CRL extension that conveys a monotonically increasing sequence number for a given CRL scope and CRL issuer. This extension allows users to easily determine when a particular CRL supersedes another CRL. CRL numbers also support the identification of complementary complete CRLs and delta CRLs."

See Also

TScCertificateExtension

TScCRLDeltaIndicatorExtension

5.28.2 Properties

5.28.2.1 CRLNumber

```
property CRLNumber: string;
```

Description

The **CRLNumber** property specifies sequence number for a given CRL scope and CRL issuer.

This property is read-only.

5.29 TScCRLReasonCodeExtension

5.29.1 Description

Unit

ScCertificateExts

Description

The **TScCRLReasonCodeExtension** class represents the Reason Code extension, that identifies the reason for the certificate revocation.

The following paragraph is taken from RFC 5280, section 5.3.1:

"The reasonCode is a non-critical CRL entry extension that identifies the reason for the certificate revocation. CRL issuers are strongly encouraged to include meaningful reason codes in CRL entries; however, the reason code CRL entry extension SHOULD be absent instead of using the unspecified reasonCode value."

See Also

TScCertificateExtension

5.29.2 Properties

5.29.2.1 CRLReason

```
property CRLReason: TScCRLReason;
```

Description

The **CRLReason** property specifies the reason for the certificate revocation.

This property is read-only.

5.30 TScExtensions

5.30.1 Description

Unit

ScCertificateExts

Description

TScExtensions maintains a list of the TScCertificateExtension objects.

Use **TScExtensions** to store and maintain a list of objects. **TScExtensions** provides properties and

methods to add, delete, locate, and access objects. **TScExtensions** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScExtensions** instance is itself destroyed.

See also

TScCertificateExtension

5.30.2 Properties

5.30.2.1 Extensions

```
property Extensions[Index: integer]: TScCertificateExtension; default;
```

Description

Lists the TScCertificateExtension object references.

Use **Extensions** to access objects in the list. **Extensions** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Extensions** with the Count property to iterate through the list.

Reassigning an **Extensions** index frees the object that previously occupied that position in the list.

Note: Extensions is the default property of TScExtensions. This means you can omit the property name.

See also

Count

FindExtensionByClass

TScCertificateExtension

5.30.3 Methods

5.30.3.1 FindExtensionByClass

```
function FindExtensionByClass(AClass: TScCertificateExtensionClass):
TScCertificateExtension;
```

Description

Call **FindExtensionByClass** to determine if a specified extension class is referenced in the Extensions list. `AClass` is the class of the object for which to search. If **FindExtensionByClass** finds an item with a matching class, it returns the TScCertificateExtension object for the specified item. Otherwise it returns nil.

5.31 TScPersistent

5.31.1 Description

Unit

ScUtils

Description

TScPersistent is an abstract class, which is the ancestor for all objects that have assignment and cloning capabilities. For this purpose, **TScPersistent** introduces methods that can be overridden to:

- Provide the means to assign the contents of one object to another.
- Provide the means to create an exact copy of the object.

Do not create instances of **TScPersistent**. Use **TScPersistent** as a base class when declaring objects that have their properties assigned to other objects.

See also

TScPersistentObjectList

5.31.2 Methods

5.31.2.1 Assign

```
procedure Assign(Source: TScPersistent); virtual; abstract;
```

Description

Copies the contents of another similar object. **Assign** copies properties and other attributes of the specified `Source` object to the current object.

This method should be overridden in descendant classes to handle the assignment of properties from similar objects.

5.31.2.2 Clone

```
function Clone: TScPersistent; virtual; abstract;
```

Description

Creates a clone of the current instance.

This method should be overridden in descendant classes to create a clone of the current instance.

5.32 TScPersistentObjectList

5.32.1 Description

Unit

ScUtils

Description

TScPersistentObjectList maintains a list of the TScPersistent objects.

Use **TScPersistentObjectList** to store and maintain a list of objects. **TScPersistentObjectList** provides properties and methods to add, delete, locate, and access objects.
TScPersistentObjectList controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScPersistentObjectList** instance is itself destroyed.

See also

TScPersistent

5.32.2 Properties

5.32.2.1 Count

```
property Count: Integer;
```

Description

Read **Count** to determine the number of entries in the Items array.

This property is read-only.

See also

Items

5.32.2.2 Items

```
property Items[Index: integer]: TScPersistent;
```

Description

Lists the TScPersistent object references.

Use **Items** to access objects in the list. **Items** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index,

or use **Items** with the Count property to iterate through the list.

Reassigning an **Items** index frees the object that previously occupied that position in the list.

See also

Count

5.32.3 Methods

5.32.3.1 Assign

```
procedure Assign(Source: TScPersistentObjectList); virtual;
```

Description

Call the **Assign** method to assign the elements of another list to this one.

5.32.3.2 Add

```
function Add(Item: TScPersistent): integer;
```

Description

Call **Add** to insert an object at the end of the list. **Add** places the object after the last item, even if the array contains nil references, and returns the index of the inserted object. The first object in the list has an index of 0.

Add increments Count and, if necessary, allocates memory.

See also

Insert

Items

5.32.3.3 Clear

```
procedure Clear;
```

Description

Deletes all items from the list and frees all objects.

Call **Clear** to empty the Items array and set the Count to 0. **Clear** also frees the memory used to store the Items array.

See also

Items

5.32.3.4 Delete

```
procedure Delete(Index: integer);
```

Description

Removes the item at the position given by the `Index` parameter. **Delete** frees the object in addition to removing it from the list.

Call **Delete** to remove the item at a specific position from the list. The index is zero-based, so the first item has an index value of 0, the second item has an index value of 1, and so on. Calling **Delete** moves up all items in the Items array that follow the deleted item, and reduces the Count.

See also

Remove

Items

5.32.3.5 IndexOf

```
function IndexOf(Item: TScPersistent): integer;
```

Description

Returns the index of the first object in the list with a specified value.

Call **IndexOf** to get the index for a specified object in the list, where the first object has index 0, the second object has index 1, and so on. If an object is not in the list, **IndexOf** returns -1. If an object appears more than once, **IndexOf** returns the index of the first appearance.

See also

Items

5.32.3.6 Insert

```
procedure Insert(Index: integer; Item: TScPersistent);
```

Description

Call **Insert** to add an object at a specified position in the list, shifting the item that previously occupied that position (and all subsequent items) up. **Insert** increments Count and, if necessary, allocates memory.

The `Index` parameter is zero-based, so the first position in the list has an index of 0.

See also

Add

Items

5.32.3.7 Remove

```
function Remove(Item: TScPersistent): integer;
```

Description

Call **Remove** to delete a specific object from the list when its index is unknown. The value returned is the index of the object in the Items array before it was removed. If the specified object is not found on the list, **Remove** returns –1. **Remove** frees the object in addition to removing it from the list.

After an object is deleted, all the objects that follow it are moved up in index position and Count is decremented. If an object appears more than once on the list, **Remove** deletes only the first appearance. Hence, removing an object that appears more than once results in empty object references later in the list.

To use an index position (rather than an object reference) to specify the object to be removed, call Delete.

See also

Delete

Items

5.33 TScRelativeDistinguishedName

5.33.1 Description

Unit

ScCertificateExts

Description

The **TScRelativeDistinguishedName** class contains a Relative Distinguished Name (RDN). The RDN is a sequence of attributes with an associated value in the form Object Identifier (OID)=Value, connected by commas.

TScRelativeDistinguishedName maintains a list of the TScASN1Attribute objects.

To access OIDs use the Names property. To get a value of an attribute, use the Values property.

The **TScRelativeDistinguishedName** class allows to encode the information in the object into a PKCS #7 message.

Class is read-only.

See Also

Encode

Items

Count

5.33.2 Properties

5.33.2.1 Count

```
property Count: integer;
```

Description

Read **Count** to determine the number of entries in the Items array, that lists the TScASN1Attribute object references. The property specifies the number of pairs "Object Identifier=Value" in the list.

This property is read-only.

See also

Items

5.33.2.2 Items

```
property Items[Index: integer]: TScASN1Attribute; default;
```

Description

Lists the TScASN1Attribute object references.

Use **Items** to access objects in the list. **Items** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read the value at a specific index, or use **Items** with the Count property to iterate through the list.

Note: Items is the default property of TScRelativeDistinguishedName. This means you can omit the property name.

See also

Count

5.33.2.3 Names

```
property Names[Index: integer]: string;
```

Description

Lists the Object Identifiers of Relative Distinguished Name (RDN). (The RDN is a sequence of attributes with an associated value in the form Object Identifier=Value.)

Use **Names** to obtain the Object Identifier (OID). **Names** is a zero-based array: the first OID is

indexed as 0, the second OID is indexed as 1, and so on. The `Index` parameter indicates the index of the OID. You can read the value at a specific index, or use **Names** with the Count property to iterate through the list.

This property is read-only.

See also

Count

5.33.2.4 Values

```
property Values[const OId: string]: string;
```

Description

Use **Values** for getting the value for specified Object Identifier (OID) of Relative Distinguished Name (RDN). (The RDN is a sequence of attributes with an associated value in the form Object Identifier=Value.)

The `OId` parameter indicates the dotted number or friendly name of the OID.

This property is read-only.

To iterate through all of the values in the list, use the ValueFromIndex and Count properties.

See also

ValueFromIndex

Count

5.33.2.5 ValueFromIndex

```
property ValueFromIndex[Index: integer]: string;
```

Description

Lists the values of Relative Distinguished Name (RDN). (The RDN is a sequence of attributes with an associated value in the form Object Identifier=Value.)

Use **ValueFromIndex** to get the value of RDN. **ValueFromIndex** is a zero-based array: the first value is indexed as 0, the second value is indexed as 1, and so on. The `Index` parameter indicates the index of the value. You can read the value at a specific index, or use **ValueFromIndex** with the Count property to iterate through the list.

This property is read-only.

See also

Count

5.33.3 Methods

5.33.3.1 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the information in the object into a PKCS #7 message.

PKCS #7 ASN.1 attribute syntax:

```
RelativeDistinguishedName ::=
  SET SIZE (1..MAX) OF SEQUENCE {
    attributeType OBJECT IDENTIFIER,
    attributeValue ANY}
```

5.33.3.2 Equals

```
function Equals(Value: TScRelativeDistinguishedName): boolean;
```

Description

Use the **Equals** method to compare content of two RDN objects. If both values coincide, the method returns True.

5.33.3.3 ToString

```
function ToString: string;
```

Description

Use **ToString** to represent the comma-delimited Relative Distinguished Name from an X509 certificate as a string, like this:

`"CN=Company CA, O=Corporation, C=US"`.

5.34 TScDistinguishedName

5.34.1 Description

Unit

ScCertificateExts

Description

The **TScDistinguishedName** class contains a Distinguished Name (DN). The DN is a sequence of relative distinguished names (RDN) connected by commas. The RDN is a sequence of attributes with an associated value in the form Object Identifier (OID)=Value.

TScDistinguishedName maintains a list of the TScRelativeDistinguishedName objects.

To access OIDs use the Names property. To get a value of an OID, use the Values property.

This class is like an extension to the SubjectName or IssuerName property, which is the name of the person or entity that the certificate is being issued to.

The **TScDistinguishedName** class allows to encode the information in the object into a PKCS #7 message.

Class is read-only.

See Also

Encode

Items

Count

TScCertificate.IssuerName

TScCertificate.SubjectName

5.34.2 Properties

5.34.2.1 Count

```
property Count: integer;
```

Description

Read **Count** to determine the number of entries in the Items array, that lists the TScRelativeDistinguishedName object references.

This property is read-only.

See also

Items

5.34.2.2 Items

```
property Items[Index: integer]: TScRelativeDistinguishedName; default;
```

Description

Lists the TScRelativeDistinguishedName object references.

Use **Items** to access objects in the list. **Items** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read the value at a specific index, or use **Items** with the Count property to iterate through the list.

Note: Items is the default property of TScDistinguishedName. This means you can omit the property name.

See also

Count

5.34.2.3 Names

```
property Names[Index: integer]: string;
```

Description

Lists the Object Identifiers of Relative Distinguished Names (RDN) sequence. (The RDN is a sequence of attributes with an associated value in the form Object Identifier=Value.)

Use **Names** to obtain the Object Identifier (OID). **Names** is a zero-based array: the first OID is indexed as 0, the second OID is indexed as 1, and so on. The `Index` parameter indicates the index of the OID. You can read the value at a specific index, or use **Names** with the ValueCount property to iterate through the list.

This property is read-only.

See also

ValueCount

5.34.2.4 Values

```
property Values[const OId: string]: string;
```

Description

Use **Values** for getting the value for specified Object Identifier (OID) of Relative Distinguished Names (RDN) sequence. (The RDN is a sequence of attributes with an associated value in the form OID=Value.)

This property is read-only.

To iterate through all of the values in the list, use the ValueFromIndex and ValueCount properties.

See also

ValueFromIndex

ValueCount

5.34.2.5 ValueFromIndex

```
property ValueFromIndex[Index: integer]: string;
```

Description

Lists the values of Relative Distinguished Names (RDN) sequence. (The RDN is a sequence of attributes with an associated value in the form Object Identifier=Value.)

Use **ValueFromIndex** to get the value of RDN. **ValueFromIndex** is a zero-based array: the first value is indexed as 0, the second value is indexed as 1, and so on. The `Index` parameter indicates the index of the value. You can read the value at a specific index, or use **ValueFromIndex** with the ValueCount property to iterate through the list.

This property is read-only.

See also

ValueCount

5.34.2.6 ValueCount

```
property ValueCount: integer;
```

Description

Read **ValueCount** to determine the number of pairs "Object Identifier=Value" in the Names and Values lists.

This property is read-only.

See also

Names

ValueFromIndex

5.34.3 Methods

5.34.3.1 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the information in the object into a PKCS #7 message.

PKCS #7 ASN.1 attribute syntax:

```
DistinguishedName ::=
```

```
SEQUENCE OF {
  SET SIZE (1..MAX) OF SEQUENCE {
    attributeType OBJECT IDENTIFIER,
    attributeValue ANY}
}
```

5.34.3.2 Equals

```
function Equals(Value: TScDistinguishedName): boolean;
```

Description

Use the **Equals** method to compare content of two Distinguished Name (DN) objects. If both values coincide, the method returns True.

5.34.3.3 ToString

```
function ToString: string;
```

Description

Use **ToString** to represent the comma-delimited Distinguished Name (DN) from an X509 certificate as a string, like this:

`"CN=Company CA, O=Corporation, C=US"`.

5.35 TScDistinguishedNameList

5.35.1 Description

Unit

ScCertificateExts

Description

TScDistinguishedNameList maintains a list of the TScDistinguishedName objects.

Use **TScDistinguishedNameList** to store and maintain a list of objects. **TScDistinguishedNameList** provides properties and methods to add, delete, locate, and access objects. **TScDistinguishedNameList** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScDistinguishedNameList** instance is itself destroyed.

See also

TScDistinguishedName

5.35.2 Properties

5.35.2.1 Names

```
property Names[Index: integer]: TScDistinguishedName; default;
```

Description

Lists the TScDistinguishedName object references.

Use **Names** to access objects in the list. **Names** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Names** with the Count property to iterate through the list.

Reassigning an **Names** index frees the object that previously occupied that position in the list.

Note: Names is the default property of TScDistinguishedNameList. This means you can omit the property name.

See also

Count

TScDistinguishedName

5.36 TScOid

5.36.1 Description

Unit

ScCertificateExts

Description

The **TScOid** class represents a cryptographic object identifier.

Cryptographic object identifiers consist of a value/name pair. If one property in a pair is set to a known value, the other property is updated automatically to a corresponding value.

5.36.2 Properties

5.36.2.1 FriendlyName

```
property FriendlyName: string;
```

Description

Gets or sets the friendly name of the identifier.

If the Value property is set to a known value, the **FriendlyName** is updated automatically to a corresponding value.

See Also

Value

5.36.2.2 Value

```
property Value: string;
```

Description

Gets or sets the dotted number of the identifier.

If the FriendlyName property is set to a known value, the **Value** is updated automatically to a corresponding value.

See Also

FriendlyName

5.37 TScOIds

5.37.1 Description

Unit

ScCertificateExts

Description

TScOIds maintains a list of the TScOId objects.

Use **TScOIds** to store and maintain a list of objects. **TScOIds** provides properties and methods to add, delete, locate, and access objects. **TScOIds** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScOIds** instance is itself destroyed.

See also

TScOId

5.37.2 Properties

5.37.2.1 OIds

```
property OIds[Index: integer]: TScOId; default;
```

Description

Lists the TScOId object references.

Use **OIds** to access objects in the list. **OIds** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **OIds** with the Count property to iterate through the list.

Reassigning an **OIds** index frees the object that previously occupied that position in the list.

Note: OIds is the default property of TScOids. This means you can omit the property name.

See also

Count

TScOId

5.38 TScASN1AlgorithmIdentifier

5.38.1 Description

Unit

ScCertificateExts

Description

The **TScASN1AlgorithmIdentifier** class defines the algorithm used for a cryptographic operation.

The definition of the algorithm identifier is taken from [X.509-88] ASN1 notation.

The algorithm identifier consists of two parts - object identifier and parameters. The Object Identifier component identifies the algorithm (such as digest algorithm, signature algorithm, encryption algorithm, MAC algorithm, etc.). The contents of the optional parameters field can vary according to the algorithm identified.

5.38.2 Properties

5.38.2.1 Algorithm

```
property Algorithm: TScOId;
```

Description

Gets or sets the TScOId object that specifies the object identifier for the algorithm (such as digest algorithm, signature algorithm, encryption algorithm, MAC algorithm, etc.).

See Also

Parameters

5.38.2.2 Parameters

```
property Parameters: TBytes;
```

Description

Gets or sets any parameters required by the algorithm.

The **Parameters** is an array of byte values that varies according to the algorithm identified in the Algorithm property.

See Also

Algorithm

5.39 TScASN1AlgorithmIdentifiers

5.39.1 Description

Unit

ScCertificateExts

Description

TScASN1AlgorithmIdentifiers maintains a list of the TScASN1AlgorithmIdentifier objects.

Use **TScASN1AlgorithmIdentifiers** to store and maintain a list of objects. **TScASN1AlgorithmIdentifiers** provides properties and methods to add, delete, locate, and access objects. **TScASN1AlgorithmIdentifiers** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScASN1AlgorithmIdentifiers** instance is itself destroyed.

See also

TScASN1AlgorithmIdentifier

5.39.2 Properties

5.39.2.1 AlgorithmIdentifiers

```
property AlgorithmIdentifiers[Index: integer]:
TScASN1AlgorithmIdentifier; default;
```

Description

Lists the TScASN1AlgorithmIdentifier object references.

Use **AlgorithmIdentifiers** to access objects in the list. **AlgorithmIdentifiers** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **AlgorithmIdentifiers** with the Count property to iterate through the list.

Reassigning an **AlgorithmIdentifiers** index frees the object that previously occupied that position in the list.

Note: AlgorithmIdentifiers is the default property of TScASN1AlgorithmIdentifiers. This means you can omit the property name.

See also

Count

TScASN1AlgorithmIdentifier

5.40 TScSignatureAlgorithmIdentifier

5.40.1 Description

Unit

ScCertificateExts

Description

The **TScSignatureAlgorithmIdentifier** class defines the algorithm used for signing and verifying signature operations.

The definition of the algorithm identifier is taken from [X.509-88] ASN1 notation.

The algorithm identifier consists of two parts - object identifier and parameters. The Object Identifier component identifies the operation's algorithm. The contents of the optional parameters field can vary according to the algorithm identified.

5.40.2 Properties

5.40.2.1 HashAlgorithm

```
property HashAlgorithm: TScHashAlgorithm;
```

Description

The **HashAlgorithm** property specifies the hash algorithm used in conjunction with the specified padding mode for signing and verifying signature operations.

See Also

PaddingMode

5.40.2.2 PaddingMode

```
property PaddingMode: TScPaddingMode;
```

Description

The **PaddingMode** property specifies the padding mode used for signing and verifying signature operations.

See Also

HashAlgorithm

PSSParams

5.40.2.3 PSSParams

```
property PSSParams: TScPSSParams;
```

Description

The **PSSParams** property specifies the PSS padding parameters to use with signing and verifying signature operations.

See Also

PaddingMode

5.41 TScASN1Attribute

5.41.1 Description

Unit

ScCertificateExts

Description

The **TScASN1Attribute** class represents ASN.1-encoded data and Object Identificator that provides information about the type of attribute associated with this object.

Abstract Syntax Notation One (ASN.1), which is defined in CCITT Recommendation X.208, is a way to specify abstract objects that will be serially transmitted. The set of ASN.1 rules for representing such objects as strings of ones and zeros is called the Distinguished Encoding Rules (DER), and is defined in CCITT Recommendation X.509, Section 8.7.

5.41.2 Properties

5.41.2.1 ASN1DataType

```
property ASN1DataType: TScASN1DataType;
```

Description

Use **ASN1DataType** to determine the ASN.1-encoded data type.

See Also

TScASN1DataType

RawData

5.41.2.2 AsString

```
property AsString: string;
```

Description

Use **AsString** to represent the ASN.1-encoded data as a string.

See Also

RawData

5.41.2.3 OId

```
property OId: TScOId;
```

Description

Gets or sets the TScOid object that represents the type of attribute associated with this object.

This property can be used to provide information about the ASN.1-encoded data, such as the algorithm used to encrypt the data.

See Also

RawData

5.41.2.4 RawData

```
property RawData: TBytes;
```

Description

Use **RawData** to obtain the ASN.1-encoded data represented in a byte array.

See Also

ASN1DataType

AsString

OId

5.41.3 Methods

5.41.3.1 Create

```
constructor Create(const OId: string; const Value: TBytes; DataType:
TScASN1DataType); overload;
constructor Create(const OId: string; const Value: string; DataType:
TScASN1DataType); overload;
```

Description

Create **TScASN1Attribute** instance.

The `OId` parameter is an object that represents the type of attribute associated with this object. The OId property is set from the value of this parameter.

The `Value` parameter is a byte array or a string that contains the ASN.1-encoded data. The RawData property is set from the value of this parameter.

The `DataType` parameter determines the ASN.1 type of the encoded data. The ASN1DataType property is set from the value of this parameter.

5.41.3.2 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the information in the object into a PKCS #7 message.

PKCS #7 ASN.1 attribute syntax:

```
ASN1Attribute ::=
  attributeType OBJECT IDENTIFIER, attributeValue ANY
```

5.41.3.3 Equals

```
function Equals(AttrValue: TScASN1Attribute): boolean;
```

Description

Use the **Equals** method to compare content of two raw values of the object's attribute. If both values coincide, the method returns True.

5.42 TScASN1Attributes

5.42.1 Description

Unit

ScCertificateExts

Description

TScASN1Attributes maintains a list of the TScASN1Attribute objects.

Use **TScASN1Attributes** to store and maintain a list of objects. **TScASN1Attributes** provides properties and methods to add, delete, locate, and access objects. **TScASN1Attributes** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScASN1Attributes** instance is itself destroyed.

See also

TScASN1Attribute

5.42.2 Properties

5.42.2.1 Attributes

```
property Attributes[Index: integer]: TScASN1Attribute; default;
```

Description

Lists the TScASN1Attribute object references.

Use **Attributes** to access objects in the list. **Attributes** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Attributes** with the Count property to iterate through the list.

Reassigning an **Attributes** index frees the object that previously occupied that position in the list.

Note: Attributes is the default property of TScASN1Attributes. This means you can omit the property name.

See also

Count

TScASN1Attribute

5.43 TScPKCS7Attribute

5.43.1 Description

Unit

ScCertificateExts

Description

The **TScPKCS7Attribute** class represents an attribute used for CMS/PKCS #7 and PKCS #9 operations.

TScPKCS7Attribute contains Object Identificator that provides information about the type of attribute associated with this object and list of the TScASN1Attribute objects.

The **TScPKCS7Attribute** class allows to encode the information in the object into a PKCS #7 message.

See Also

Encode

OId

Values

ValueCount

5.43.2 Properties

5.43.2.1 OId

```
property OId: TScOId;
```

Description

Gets or sets the TScOId object that represents the type of attribute associated with this object.

OId provides information about the attribute, such as the algorithm used to encrypt the data.

5.43.2.2 ValueCount

```
property ValueCount: integer;
```

Description

Read **ValueCount** to determine the number of entries in the Values array.

See Also

Values

5.43.2.3 Values

```
property Values[Index: integer]: TScASN1Attribute;
```

Description

Lists the TScASN1Attribute object references, those represent ASN.1-encoded data.

Use **Values** to access objects in the list. **Values** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Values** with the ValueCount property to iterate through the list.

Reassigning an **Values** index frees the object that previously occupied that position in the list.

See Also

ValueCount

5.43.3 Methods

5.43.3.1 AddValue

```
procedure AddValue(Item: TScASN1Attribute); overload;
procedure AddValue(const Value: TBytes; ASN1DataType: TScASN1DataType);
overload;
procedure AddValue(const Value: string; ASN1DataType: TScASN1DataType);
overload;
```

Description

Call **AddValue** to insert an attribute's value at the end of the Values list. **AddValue** increments ValueCount.

The `Value` parameter is a byte array or a string that contains the ASN.1-encoded data.

The `ASN1DataType` parameter determines the ASN.1 type of the encoded data.

5.43.3.2 ClearValues

```
procedure ClearValues;
```

Description

Deletes all values from the Values list and frees all objects.

Call **ClearValues** to empty the Values array and set the ValueCount to 0.

5.43.3.3 DeleteValue

```
procedure DeleteValue(Index: integer);
```

Description

Removes the value at the position given by the `Index` parameter. **DeleteValue** frees the object in addition to removing it from the list.

Call **DeleteValue** to remove the value at a specific position from the list. The index is zero-based, so the first item has an Index value of 0, the second item has an Index value of 1, and so on. Calling **DeleteValue** moves up all items in the Values array that follow the deleted item, and reduces the ValueCount.

5.43.3.4 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the information in the object into a PKCS #7 message.

PKCS #7 ASN.1 attribute syntax:

```
Attribute ::= SEQUENCE {
  type OBJECT IDENTIFIER,
  values SET OF SEQUENCE {
    type OBJECT IDENTIFIER,
    value ANY
  }
```

```
}
```

5.44 TScPKCS7Attributes

5.44.1 Description

Unit

ScCertificateExts

Description

TScPKCS7Attributes maintains a list of the TScPKCS7Attribute objects.

Use **TScPKCS7Attributes** to store and maintain a list of objects. **TScPKCS7Attributes** provides properties and methods to add, delete, locate, and access objects. **TScPKCS7Attributes** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScPKCS7Attributes** instance is itself destroyed.

See also

TScPKCS7Attribute

5.44.2 Properties

5.44.2.1 Attributes

```
property Attributes[Index: integer]: TScPKCS7Attribute; default;
```

Description

Lists the TScPKCS7Attribute object references.

Use **Attributes** to access objects in the list. **Attributes** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Attributes** with the Count property to iterate through the list.

Reassigning an **Attributes** index frees the object that previously occupied that position in the list.

Note: Attributes is the default property of TScPKCS7Attributes. This means you can omit the property name.

See also

Count

TScPKCS7Attribute

5.44.3 Methods

5.44.3.1 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the information in the object into a PKCS #7 message.

PKCS #7 ASN.1 attribute syntax:

```
Attributes ::= SET OF SEQUENCE {
  SEQUENCE {
    type OBJECT IDENTIFIER,
    values SET OF SEQUENCE {
      type OBJECT IDENTIFIER,
      value ANY
    }
  }
}
```

5.45 TScGeneralName

5.45.1 Description

Unit

ScCertificateExts

Description

The **TScGeneralName** class represents various information about identity. See RFC 5280 section 4.2.1.6 for details.

The following syntax shows the Abstract Syntax Notation One (ASN.1) structure of the General Name.

```
GeneralName ::= CHOICE
{
  otherName              [0] OtherName,
  rfc822Name             [1] IA5STRING,
  dNSName                [2] IA5STRING,
  x400Address            [3] SeqOfAny,
  directoryName          [4] Name,
  ediPartyName           [5] SeqOfAny,
  uniformResourceLocator [6] IA5STRING,
  iPAddress              [7] OCTET STRING,
  registeredID           [8] OBJECT IDENTIFIER
```

```
}
```

See Also

TScCertAlternativeNameExtension

5.45.2 Properties

5.45.2.1 Name

```
property Name: string;
```

Description

Use the **Name** property to specify the content of this instance.

5.45.2.2 Value

```
property Value: string;
```

Description

Use the **Value** property to store various information about identity.

See Also

Values

5.45.2.3 DirectoryName

```
property DirectoryName: TScDistinguishedName;
```

Description

Use this property to store information about identity in form of Distinguished Name (DN).

The DN is a sequence of relative distinguished names (RDN) connected by commas. The RDN is a sequence of attributes with an associated value in the form Object Identifier (OID)=Value.

See Also

TScDistinguishedName

5.45.3 Methods

5.45.3.1 Equals

```
function Equals(GName: TScGeneralName): boolean; overload;
function Equals(DName: TScDistinguishedName): boolean; overload;
```

Description

Use the **Equals** method to compare content of two General Name objects or Distinguished Name (DN) and this General Name. If both values coincide, the method returns True.

5.45.3.2 ToString

```
function ToString: string;
```

Description

Use **ToString** to represent the General Name as a string.

5.46 TScGeneralNames

5.46.1 Description

Unit

ScCertificateExts

Description

TScGeneralNames maintains a list of the TScGeneralName objects.

Use **TScGeneralNames** to store and maintain a list of objects. **TScGeneralNames** provides properties and methods to add, delete, locate, and access objects. **TScGeneralNames** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScGeneralNames** instance is itself destroyed.

See also

TScGeneralName

TScCertAlternativeNameExtension

5.46.2 Properties

5.46.2.1 GeneralNames

```
property GeneralNames[Index: integer]: TScGeneralName; default;
```

Description

Lists the TScGeneralName object references.

Use **GeneralNames** to access objects in the list. **GeneralNames** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **GeneralNames** with the Count property to iterate through the list.

Reassigning an **GeneralNames** index frees the object that previously occupied that position in the list.

Note: GeneralNames is the default property of TScGeneralNames. This means you can omit the property name.

See also

Count

TScGeneralName

FindByName

5.46.3 Methods

5.46.3.1 Equals

```
function Equals(Value: TScGeneralNames): boolean;
```

Description

Use the **Equals** method to compare content of two list of General Name objects. If both values coincide, the method returns True.

5.46.3.2 FindByName

```
function FindByName(const AName: string): TScGeneralName;
```

Description

Call **FindByName** to determine if a specified name is referenced in the GeneralNames list. `AName` is the name of the object for which to search. If **FindByName** finds an item with a matching name, it returns the TScGeneralName object for the specified item. Otherwise it returns nil.

See also

GeneralNames

5.46.3.3 ToString

```
function ToString: string;
```

Description

Use the **ToString** method to represent the list of General Name objects as a string.

5.47 TScPolicy

5.47.1 Description

Unit

ScCertificateExts

Description

The **TScPolicy** class represents the information about a single certificate policy.

The following paragraph is taken from RFC 5280, section 4.2.1.4:

"The certificate policies extension contains a sequence of one or more policy information terms, each of which consists of an object identifier (OID) and optional qualifiers. Optional qualifiers, which MAY be present, are not expected to change the definition of the policy."

See Also

TScCertificatePoliciesExtension

5.47.2 Properties

5.47.2.1 Identifier

```
property Identifier: string;
```

Description

Gets or sets the **Identifier** property that contains the Object Identifier (OID) of the policy information (see Description).

See Also

Qualifiers

5.47.2.2 Qualifiers

```
type
  TScQualifier = record
    QualifierId: string;
    CpsUri: string;
    NoticeReferenceOrganization: string;
    ExplicitText: string;
  end;

property Qualifiers: array of TScQualifier;
```

Description

The **Qualifiers** property is an array of the `TScQualifier` records.

The content of this property is determined by the Identifier property.

See Also

Identifier

5.48 TScPolicyList

5.48.1 Description

Unit

ScCertificateExts

Description

TScPolicyList maintains a list of the TScPolicy objects.

Use **TScPolicyList** to store and maintain a list of objects. **TScPolicyList** provides properties and methods to add, delete, locate, and access objects. **TScPolicyList** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScPolicyList** instance is itself destroyed.

See also

TScPolicy

TScCertificatePoliciesExtension

5.48.2 Properties

5.48.2.1 Policies

```
property Policies[Index: integer]: TScPolicy; default;
```

Description

Lists the TScPolicy object references.

Use **Policies** to access objects in the list. **Policies** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Policies** with the Count property to iterate through the list.

Reassigning an **Policies** index frees the object that previously occupied that position in the list.

Note: Policies is the default property of TScPolicyList. This means you can omit the property name.

See also

Count

TScPolicy

5.49 TScPolicyMapping

5.49.1 Description

Unit

ScCertificateExts

Description

The **TScPolicyMapping** class corresponds to a single policy mapping.

The following paragraph is taken from RFC 5280, section 4.2.1.5:

"The Policy Mappings extension is used in CA certificates. It lists one or more pairs of OIDs; each pair includes an IssuerDomainPolicy and a SubjectDomainPolicy. The pairing indicates the issuing CA considers its IssuerDomainPolicy equivalent to the subject CA's SubjectDomainPolicy."

See Also

TScCertPolicyMappingsExtension

5.49.2 Properties

5.49.2.1 IssuerDomainPolicy

```
property IssuerDomainPolicy: TScOId;
```

Description

This property specifies the Object Identifier of Issuer Domain Policy (see Description).

See Also

TScOId

SubjectDomainPolicy

5.49.2.2 SubjectDomainPolicy

```
property SubjectDomainPolicy: TScOId;
```

Description

This property specifies the Object Identifier of Subject Domain Policy (see Description).

See Also

TScOId

IssuerDomainPolicy

5.49.3 Methods

5.49.3.1 Create

```
constructor Create; overload;
constructor Create(const IssuerDomainPolicy, SubjectDomainPolicy:
string); overload;
```

Description

Create **TScPolicyMapping** instance.

The `IssuerDomainPolicy` parameter is a string that represents the Object Identifier of Issuer Domain Policy. The IssuerDomainPolicy property is set from the value of this parameter.

The `SubjectDomainPolicy` parameter is a string that represents the Object Identifier of Subject Domain Policy. The SubjectDomainPolicy property is set from the value of this parameter.

5.50 TScPolicyMappingList

5.50.1 Description

Unit

ScCertificateExts

Description

TScPolicyMappingList maintains a list of the TScPolicyMapping objects.

Use **TScPolicyMappingList** to store and maintain a list of objects. **TScPolicyMappingList** provides properties and methods to add, delete, locate, and access objects. **TScPolicyMappingList** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScPolicyMappingList** instance is itself destroyed.

See also

TScPolicyMapping

TScCertPolicyMappingsExtension

5.50.2 Properties

5.50.2.1 PolicyMappings

```
property PolicyMappings[Index: integer]: TScPolicyMapping; default;
```

Description

Lists the TScPolicyMapping object references.

Use **PolicyMappings** to access objects in the list. **PolicyMappings** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **PolicyMappings** with the Count property to iterate through the list.

Reassigning an **PolicyMappings** index frees the object that previously occupied that position in the list.

Note: PolicyMappings is the default property of TScPolicyMappingList. This means you can omit the property name.

See also

Count

TScPolicyMapping

5.51 TScCRLDistributionPoint

5.51.1 Description

Unit

ScBridge

Description

The **TScCRLDistributionPoint** class corresponds to a single distribution point, that identifies how CRL information is obtained.

The following paragraph is taken from RFC 5280, section 4.2.1.13:

"The CRL distribution points extension identifies how CRL information is obtained. The cRLDistributionPoints extension is a sequence of DistributionPoint. A DistributionPoint consists of three fields, each of which is optional: DistributionPoint, Reasons, and CRLIssuer. While each of these fields is optional, a DistributionPoint MUST NOT consist of only the reasons field; either distributionPoint or cRLIssuer MUST be present. If the certificate issuer is not the CRL issuer, then the cRLIssuer field MUST be present and contain the Name of the CRL issuer. If the certificate issuer is also the CRL issuer, then conforming CAs MUST omit the cRLIssuer field and MUST include the distributionPoint field."

See Also

TScCRLDistributionPointList

TScCertCRLDistributionPointsExtension

5.51.2 Properties

5.51.2.1 CRLIssuer

```
property CRLIssuer: TScGeneralNames;
```

Description

The **CRLIssuer** property specifies the entity that signs and issues the CRL. If present, the **CRLIssuer** contains the distinguished name from the issuer field of the CRL to which the Distribution Point is pointing. The encoding of the name in the **CRLIssuer** field must be exactly the same as the encoding in issuer field of the CRL. If the **CRLIssuer** field is included and the distinguished name in that field does not correspond to an X.500 or LDAP directory entry where CRL is located, then conforming CAs must include the Distribution Point field.

See Also

DistributionPointName

5.51.2.2 DistributionPointName

```
property DistributionPointName: TScGeneralNames;
```

Description

The **DistributionPointName** property contains either a sequence of general names or a single value, NameRelativeToCRLIssuer. If the **DistributionPointName** contains multiple values, each name describes a different mechanism to obtain the same CRL. For example, the same CRL could be available for retrieval through both LDAP and HTTP.

If the **DistributionPointName** contains the single value NameRelativeToCRLIssuer, the value provides a distinguished name fragment. The fragment is appended to the X.500 distinguished name of the CRL issuer to obtain the distribution point name.

See Also

CRLIssuer

5.51.2.3 Reasons

```
property Reasons: TScCRLReasons;
```

Description

The **Reasons** property specifies a list of revocation reasons supported by the CRL.

5.52 TScCRLDistributionPointList

5.52.1 Description

Unit

ScBridge

Description

TScCRLDistributionPointList maintains a list of the TScCRLDistributionPoint objects.

Use **TScCRLDistributionPointList** to store and maintain a list of objects. **TScCRLDistributionPointList** provides properties and methods to add, delete, locate, and access objects. **TScCRLDistributionPointList** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScCRLDistributionPointList** instance is itself destroyed.

See also

TScCRLDistributionPoint

TScCertCRLDistributionPointsExtension

5.52.2 Properties

5.52.2.1 CRLDistributionPoints

```
property CRLDistributionPoints[Index: integer]: TScCRLDistributionPoint;
default;
```

Description

Lists the TScCRLDistributionPoint object references.

Use **CRLDistributionPoints** to access objects in the list. **CRLDistributionPoints** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **CRLDistributionPoints** with the Count property to iterate through the list.

Reassigning an **CRLDistributionPoints** index frees the object that previously occupied that position in the list.

Note: CRLDistributionPoints is the default property of TScCRLDistributionPointList. This means you can omit the property name.

See also

Count

TScCRLDistributionPoint

5.53 TScInfoAccess

5.53.1 Description

Unit

ScBridge

Description

The **TScInfoAccess** class corresponds to the authority information access entry, that indicates how to access information and services for the issuer of the certificate.

The following paragraph is taken from RFC 5280, section 4.2.2.1:

"The authority information access extension indicates how to access information and services for the issuer of the certificate in which the extension appears. Information and services may include on-line validation services and CA policy data.

Each entry in the sequence AuthorityInfo describes the format and location of additional information provided by the issuer of the certificate in which this extension appears. The type and format of the information are specified by the AccessMethod field; the AccessLocation field specifies the location of the information. The retrieval mechanism may be implied by the accessMethod or specified by accessLocation."

See Also

TScInfoAccessList

TScCertAuthorityInfoAccessExtension

5.53.2 Properties

5.53.2.1 AccessLocation

```
property AccessLocation: TScGeneralName;
```

Description

The **AccessLocation** property contains the location of additional information provided by the issuer of the certificate in which the extension appears.

The following paragraph is taken from RFC 5280, section 4.2.2.1:

"This profile defines two accessMethod OIDs: id-ad-calssuers and id-ad-ocsp. In a public key certificate, the id-ad-calssuers OID is used when the additional information lists certificates that were issued to the CA that issued the certificate containing this extension. The referenced CA issuers description is intended to aid certificate users in the selection of a certification path that terminates at a point trusted by the certificate user.

When id-ad-calssuers appears as accessMethod, the accessLocation field describes the referenced description server and the access protocol to obtain the referenced description. The accessLocation field is defined as a GeneralName, which can take several forms."

See Also

AccessMethod

5.53.2.2 AccessMethod

```
property AccessMethod: TScOid;
```

Description

The **AccessMethod** property contains the format of additional information provided by the issuer of the certificate in which the extension appears.

The following paragraph is taken from RFC 5280, section 4.2.2.1:

"This profile defines two accessMethod OIDs: id-ad-calssuers and id-ad-ocsp. In a public key certificate, the id-ad-calssuers OID is used when the additional information lists certificates that were issued to the CA that issued the certificate containing this extension. The referenced CA issuers description is intended to aid certificate users in the selection of a certification path that terminates at a point trusted by the certificate user.

When id-ad-calssuers appears as accessMethod, the accessLocation field describes the referenced description server and the access protocol to obtain the referenced description. The accessLocation field is defined as a GeneralName, which can take several forms."

See Also

AccessLocation

5.54 TScInfoAccessList

5.54.1 Description

Unit

ScBridge

Description

TScInfoAccessList maintains a list of the TScInfoAccess objects.

Use **TScInfoAccessList** to store and maintain a list of objects. **TScInfoAccessList** provides properties and methods to add, delete, locate, and access objects. **TScInfoAccessList** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScInfoAccessList** instance is itself destroyed.

See also

TScInfoAccess

TScCertAuthorityInfoAccessExtension

5.54.2 Properties

5.54.2.1 InfoAccesses

```
property InfoAccesses[Index: integer]: TScInfoAccess; default;
```

Description

Lists the TScInfoAccess object references.

Use **InfoAccesses** to access objects in the list. **InfoAccesses** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **InfoAccesses** with the Count property to iterate through the list.

Reassigning an **InfoAccesses** index frees the object that previously occupied that position in the list.

Note: InfoAccesses is the default property of TScInfoAccessList. This means you can omit the property name.

See also

Count

TScInfoAccess

5.55 TScOAEPParams

5.55.1 Description

Unit

ScBridge

Description

The **TScOAEPParams** class specifies padding parameters for the PKCS#1 v2.1 RSAES-OAEP encryption algorithm.

See Also

TScPaddingMode

5.55.2 Properties

5.55.2.1 HashAlgorithm

```
property HashAlgorithm: TScHashAlgorithm;
```

Description

The **HashAlgorithm** property specifies the hash algorithm used in conjunction with the OAEP padding mode.

5.55.2.2 MaskGenHashAlgorithm

```
property MaskGenHashAlgorithm: TScHashAlgorithm;
```

Description

The **MaskGenHashAlgorithm** property specifies the hash algorithm used in the mask generation function in conjunction with the OAEP padding mode.

5.55.3 Methods

5.55.3.1 Assign

```
procedure Assign(Source: TScOAEPParams);
```

Description

Copies the contents of another similar object. **Assign** copies properties of the specified `Source` object to the current object.

5.55.3.2 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the information in the object into a PKCS #7 message.

PKCS #7 ASN.1 attribute syntax:

```
RSASES-OAEP-params ::= SEQUENCE {
  hashAlgorithm            [0] HashAlgorithm     DEFAULT sha1,
  maskGenerationAlgorithm  [1] MaskGenAlgorithm  DEFAULT mgf1SHA1,
  pSourceAlgorithm         [2] PSourceAlgorithm
}
```

See Also

Decode

5.55.3.3 Decode

```
procedure Decode(const RawData: TBytes);
```

Description

The **Decode** method decodes the information from a PKCS #7 message into the object.

See Also

Encode

5.56 TScPSSParams

5.56.1 Description

Unit

ScBridge

Description

The **TScPSSParams** class specifies padding parameters for the PKCS#1 RSASSA-PSS signature scheme.

See Also

TScPaddingMode

5.56.2 Properties

5.56.2.1 HashAlgorithm

```
property HashAlgorithm: TScHashAlgorithm;
```

Description

The **HashAlgorithm** property specifies the hash algorithm used in conjunction with the PSS padding mode.

5.56.2.2 MaskGenHashAlgorithm

```
property MaskGenHashAlgorithm: TScHashAlgorithm;
```

Description

The **MaskGenHashAlgorithm** property specifies the hash algorithm used in the mask generation function in conjunction with the PSS padding mode.

5.56.2.3 SaltLength

```
property SaltLength: integer;
```

Description

The **SaltLength** field is the octet length of the salt. For a given HashAlgorithm, the recommended value of **SaltLength** is the number of octets in the hash value.

See Also

HashAlgorithm

5.56.3 Methods

5.56.3.1 Assign

```
procedure Assign(Source: TScPSSParams);
```

Description

Copies the contents of another similar object. **Assign** copies properties of the specified `Source` object to the current object.

5.56.3.2 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the information in the object into a PKCS #7 message.

PKCS #7 ASN.1 attribute syntax:

```
RScASSA-PSS-params ::= SEQUENCE {
  hashAlgorithm            [0] HashAlgorithm     DEFAULT sha1,
  maskGenerationAlgorithm  [1] MaskGenAlgorithm  DEFAULT mgf1SHA1,
  saltLength               [2] INTEGER           DEFAULT 20,
  trailerField             [3] TrailerField      DEFAULT
trailerFieldBC
}
```

See Also

Decode

5.56.3.3 Decode

```
procedure Decode(const RawData: TBytes);
```

Description

The **Decode** method decodes the information from a PKCS #7 message into the object.

See Also

Encode

5.57 TScCertificate

5.57.1 Description

Unit

ScBridge

Description

The **TScCertificate** class is used for working with X.509 certificates. The X.509 structure originated in

the International Organization for Standardization (ISO) working groups. This structure can be used to represent various types of information including identity, entitlement, and holder attributes.

The certificate contains a public key, and some additional information (e.g. information about the certification center produced the certificate, information about certificate user, the service period of the certificate, etc.).

Certificates can be used for organizations authentication, for authenticity verification of transferred information, and for data encryption.

Certificates can be stored in different formats. To import/export a certificate in one of formats, you should use the ImportFrom or ExportTo methods correspondingly. To store a set of certificates in storage, the CertificateList property is used.

The **TScCertificate** lets you to sign data with the private key, which associated with the certificate, and verify signature with the certificate public key by using the Sign and VerifySign methods. The data signing is used for checking data integrity.

Also **TScCertificate** lets encrypting and decrypting information using Encrypt and Decrypt methods.

See Also

TScCertificateList

TScStorage

5.57.2 Properties

5.57.2.1 CertificateList

```
property CertificateList: TScCertificateList;
```

Description

The **CertificateList** property is used for automatic loading and storing certificates in Storage. If the certificate was not loaded or imported, and you are trying to invoke functions that use data of the certificate, it is automatically loaded from underlying Storage using CertName.

See Also

Ready

CertificateList.Storage

5.57.2.2 CertName

```
property CertName: string;
```

Description

The **CertName** property represents a certificate name, that is used for automatic loading and saving the certificate in CertificateList.

See Also

TScStorage

5.57.2.3 CRLReason

```
property CRLReason: TScCRLReason;
```

Description

The **CRLReason** property contains the reason for the certificate revocation. The property is set automatically when verifying CRL revocation or verifying certificate chain.

See Also

TScCRL

5.57.2.4 Extensions

```
property Extensions: TScExtensions;
```

Description

Gets a collection of TScCertificateExtension objects. The extensions defined in the X.509 certificate format allow additional data to be included in the certificate. The property is set automatically when loading or importing certificates.

Use **Extensions[Index]** to obtain a pointer to a specific extension. The `Index` parameter indicates the index of the extension. 0 is the index of the first extension.

This property is read-only.

See Also

TScCertificateExtension

5.57.2.5 Handle

```
property Handle: Pointer;
```

Description

Gets a handle to a Cryptographic API certificate context. The **Handle** property is set when loading

the certificate only from the TScCryptoAPIStorage storage.

This property is read-only.

See Also

TScCryptoAPIStorage

5.57.2.6 Issuer

```
property Issuer: string;
```

Description

The **Issuer** property represents a name of the certificate authority (CA) that issued the X.509 certificate. The property is set automatically when loading or importing the certificate.
This property is read-only.

See Also

IssuerName

Subject

5.57.2.7 IssuerName

```
property IssuerName: TScDistinguishedName;
```

Description

Gets the distinguished name of the certification authority (CA) that issued the certificate. The property is set automatically when loading or importing the certificate.

This property is read-only.

See Also

Issuer

SubjectName

5.57.2.8 Key

```
property Key: TScKey;
```

Description

The asymmetric key of RSA or DSA types associated with a certificate. The key is automatically loaded on certificate load. You can use the Key.IsPrivate property to determine whether the key is private or public. In order to associate a private key with this certificate, use Key.ImportFrom. At the same time the public key and the key to be imported must be equivalent, otherwise an exception will

be raised. It is not allowed to import a public key.
This property is read-only.

5.57.2.9 NotAfter

```
property NotAfter: TDateTime;
```

Description

Gets the date in local time after which a certificate is no longer valid. The property is set automatically when loading or importing the certificate.

This property is read-only.

See Also

NotBefore

5.57.2.10 NotBefore

```
property NotBefore: TDateTime;
```

Description

Gets the date in local time on which a certificate becomes valid. The property is set automatically when loading or importing the certificate.

This property is read-only.

See Also

NotAfter

5.57.2.11 Ready

```
property Ready: Boolean;
```

Description

The **Ready** property determines whether the certificate is ready to use. Set **Ready** to True, to load data from CertificateList automatically. If the CertificateList is not assigned, an exception will be raised.

Note: If the certificate was not loaded or imported, and you are trying to invoke functions that use data of the certificate, it is automatically loaded from the underlying Storage using CertName.

See Also

CertName

CertificateList

5.57.2.12 SerialNumber

```
property SerialNumber: string;
```

Description

The serial number of the certificate. It is a unique number issued by the certificate issuer, which is also called the Certification Authority. The property is set automatically when loading or importing the certificate.

This property is read-only.

5.57.2.13 Signature

```
property Signature: TBytes;
```

Description

Signature contains the signature of the certificate content. The property is set automatically when loading or importing the certificate.

This property is read-only.

See Also

SignatureAlgorithm

5.57.2.14 SignatureAlgorithm

```
property SignatureAlgorithm: TScSignatureAlgorithmIdentifier;
```

Description

SignatureAlgorithm identifies the type of signature algorithm used by the certificate. The property is set automatically when loading or importing the certificate.

This property is read-only.

See Also

Signature

5.57.2.15 Subject

```
property Subject: string;
```

Description

The **Subject** property represents a subject name from the certificate. The property is set automatically when loading or importing the certificate.
This property is read-only.

See Also

SubjectName

Issuer

5.57.2.16 SubjectName

```
property SubjectName: TScDistinguishedName;
```

Description

Gets the distinguished name of the certificate user. The property is set automatically when loading or importing the certificate.

This property is read-only.

See Also

Subject

IssuerName

5.57.2.17 SubjectKeyIdentifier

```
property SubjectKeyIdentifier: string;
```

Description

The **SubjectKeyIdentifier** property is a string, encoded in hexadecimal format, that represents the subject key identifier (SKI). The SKI provides a unique identification for the subject of the certificate. The SKI is often used when working with XML digital signing.
The property is set automatically when loading or importing the certificate.
This property is read-only.

See Also

Subject

TScCertSubjectKeyIdExtension

5.57.2.18 Version

```
property Version: Integer;
```

Description

Gets the X.509 format version of a certificate. Possible values are 1, 2 or 3. The property is set automatically when loading or importing a certificate.

This property is read-only.

5.57.3 Methods

5.57.3.1 Decrypt

```
function Decrypt(const Data: TBytes): TBytes;
```

Description

Use the **Decrypt** method to decrypt data using the private key associated with the certificate. The function returns the source data passed to the Encrypt method.

If the certificate key is not private (Key.IsPrivate = False), an exception will be raised.

Note: If the Ready property is False, the certificate will be automatically loaded.

See also

Encrypt

5.57.3.2 Encrypt

```
function Encrypt(const Data: TBytes): TBytes;
```

Description

Use the **Encrypt** method to encrypt data with the certificate key. This method returns an encrypted data.

The maximum block size for PKCS2 padding mode should be 11 bytes less than a key size, and for OAEP padding mode - (2 + 2*HashLength) bytes less than a key size.

Note: If the Ready property is False, the certificate will be automatically loaded.

See also

Decrypt

5.57.3.3 Equals

```
function Equals(Certificate: TScCertificate): Boolean;
```

Description

Use the **Equals** method to compare content of two certificates. If parameters of both certificates coincide, the method returns True.

Note: If the Ready property of either of certificates is False, the certificate will be loaded automatically.

5.57.3.4 ExportTo

```
procedure ExportTo(const FileName: string; const CertEncoding: TScCertificateEncodin
procedure ExportTo(Stream: TStream; const CertEncoding: TScCertificateEncoding = cfP
```

Description

Use this method to export the certificate to file or to stream. The certificate can be stored in different formats.

Parameters:

- `FileName` - specifies the file name in which the certificate will be exported. If the file with the specified name does not exist, in will be created. The existent file will be overwritten.
- `Stream` - pointer to the stream in which the certificate will be exported. Data will be appended to the stream.
- `CertEncoding` - the data format which will be used for storing the certificate.

See also

ImportFrom

5.57.3.5 GetFingerprint

```
procedure GetFingerprint(const HashAlg: TScHashAlgorithm; out
Fingerprint: TBytes); overload;
procedure GetFingerprint(const HashAlg: TScHashAlgorithm; out
```

```
Fingerprint: string); overload;
```

Description

Returns the certificate thumbprint into the `Fingerprint` parameter. The print is formed by using the specified hash algorithm in the `HashAlg` parameter.

See also

Ready

5.57.3.6 GetRawData

```
function GetRawData: TBytes;
```

Description

Returns the raw data for the entire X.509v3 certificate as an array of bytes.

See also

Ready

5.57.3.7 ImportFrom

```
procedure ImportFrom(const FileName: string; const Password: string =
''); overload;
procedure ImportFrom(Stream: TStream; const Password: string = '');
overload;
```

Description

Imports the certificate from the specified file or stream.

The certificate can be stored in different formats. Format is determined automatically when loading the certificate.

Parameters:

- `FileName` - determines the file name from which the certificate will be imported. If the file does not exists, an exception will be raised.
- `Stream` - a pointer to the stream that holds data for importing the certificate.
- `Password` - the password that is used for importing data decryption.

Note: If the certificate is loaded successfully, all properties becomes assigned, and the Ready

property is set to True.

See also

ExportTo

5.57.3.8 Sign

```
function Sign(const Data: TBytes; HashAlg: TScHashAlgorithm = haSHA1;
Padding: TScPaddingMode = pmPKCS1): TBytes;
```

Description

Use the **Sign** method, to sign necessary data by using the private key, which is associated with the certificate. The function returns the signature of the specified data. If the certificate key is not private (Key.IsPrivate = False), the exception will be raised.

The `Padding` parameter can be equal only to pmPKCS1 or pmPSS values.

Use this signature to verify the data integrity. If the data substitution is possible when the data is transferred, it is required to transfer the data signature along with the data itself. In this case the receiver must have the public constituent of the key, that is used to verify the signature.

To verify the signature, use the VerifySign method.

Note: If the Ready property is False, the certificate will be automatically loaded.

See also

VerifySign

5.57.3.9 VerifyCertificate

```
procedure VerifyCertificate(ParentCertificate: TScCertificate; var
StatusSet: TScCertificateStatusSet);
```

Description

Performs a X.509 certificate validation using basic validation policy. The **VerifyCertificate** method verifies that the certificate is valid and is signed by a certificate specified in the `ParentCertificate` parameter. All the errors which occur as a result of the certificate validation are added in the `StatusSet` parameter.

See also

TScCertificateStatusSet

5.57.3.10 VerifySign

```
function VerifySign(const Data, Sign: TBytes; HashAlg: TScHashAlgorithm =
haSHA1; Padding: TScPaddingMode = pmPKCS1): boolean;
```

Description

The **VerifySign** method verifies whether the signature is correct for specified `Data` using the certificate key. If the signature is correct, the function returns True.

The `Padding` parameter can be equal only to pmPKCS1 or pmPSS values.

To get the data signature, the Sign method should be used.

Note: If the Ready property is False, the certificate will be automatically loaded.

See also

Sign

5.58 TScRevokedCertificate

5.58.1 Description

Unit

ScBridge

Description

The **TScRevokedCertificate** class identifies a single revoked certificate.

Certificate revoked by the CA is uniquely identified by the certificate serial number. Also class specifies the date on which the revocation occurred, and additional information in CRL entry extensions.

See Also

TScRevokedCertificates

TScCRL

5.58.2 Properties

5.58.2.1 Extensions

```
property Extensions: TScExtensions;
```

Description

The **Extensions** property contains list of the TScCertificateExtension objects, that specify additional attributes for CRL entry, as Reason Code, Invalidity Date, Certificate Issuer, and other.

See Also

TScCRLCertificateIssuerExtension
TScCRLInvalidityDateExtension
TScCRLReasonCodeExtension

5.58.2.2 RevocationDate

```
property RevocationDate: TDateTime;
```

Description

The **RevocationDate** property specifies the date on which the revocation occurred.

See Also

SerialNumber

5.58.2.3 SerialNumber

```
property SerialNumber: string;
```

Description

The **SerialNumber** property specifies serial number of the certificate, revoked by the CA, that uniquely identifies it.

See Also

TScCertificate.SerialNumber

5.59 TScRevokedCertificates

5.59.1 Description

Unit

ScBridge

Description

TScRevokedCertificates maintains a list of the TScRevokedCertificate objects.

Use **TScRevokedCertificates** to store and maintain a list of objects. **TScRevokedCertificates** provides properties and methods to add, delete, locate, and access objects.
TScRevokedCertificates controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScRevokedCertificates** instance is itself destroyed.

See also

TScRevokedCertificate

TScCRL

5.59.2 Properties

5.59.2.1 RevokedCertificates

```
property RevokedCertificates[Index: integer]: TScRevokedCertificate;
default;
```

Description

Lists the TScRevokedCertificate object references.

Use **RevokedCertificates** to access objects in the list. **RevokedCertificates** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **RevokedCertificates** with the Count property to iterate through the list.

Reassigning an **RevokedCertificates** index frees the object that previously occupied that position in the list.

Note: RevokedCertificates is the default property of TScRevokedCertificates. This means you can omit the property name.

See also

Count

TScRevokedCertificate

5.60 TScCRL

5.60.1 Description

Unit

ScBridge

Description

The **TScCRL** class is used for working with Certificate Revocation List (CRL).

CRLs may be used in a wide range of applications and environments covering a broad spectrum of interoperability goals and an even broader spectrum of operational and assurance requirements. CRL issuers issue CRLs. The CRL issuer is either the certification authority (CA) or an entity that has been authorized by the CA to issue CRLs. CAs publish CRLs to provide status information about the certificates they issued.

A list of the revoked certificates is stored in the RevokedCertificates property.

CRL can be stored in different formats. To import/export a CRL in one of formats, you should use the ImportFrom or ExportTo methods correspondingly. To store a set of CRLs in storage, the CRLList property is used.

See Also

TScCRLList

TScStorage

5.60.2 Properties

5.60.2.1 CRLList

```
property CRLList: TScCRLList;
```

Description

The **CRLList** property is used for automatic loading and storing the Certificate Revocation List (CRL) in Storage. If the CRL was not loaded or imported, and you are trying to invoke functions that use data of the CRL, it is automatically loaded from underlying Storage using CRLName.

See Also

CRLList.Storage

5.60.2.2 CRLName

```
property CRLName: string;
```

Description

The **CRLName** property represents a Certificate Revocation List (CRL) name, that is used for automatic loading and saving the CRL in CRLList.

See Also

TScStorage

5.60.2.3 Extensions

```
property Extensions: TScExtensions;
```

Description

Gets a collection of TScCertificateExtension objects. The extensions defined in the X.509 format allow additional data to be included in the Certificate Revocation List (CRL). The property is set automatically when loading or importing the CRL.

Use **Extensions[Index]** to obtain a pointer to a specific extension. The `Index` parameter indicates the index of the extension. 0 is the index of the first extension.

This property is read-only.

See Also

TScCertificateExtension

5.60.2.4 Issuer

```
property Issuer: string;
```

Description

The **Issuer** property represents a name of the certificate authority (CA) that issued the Certificate Revocation List (CRL). The property is set automatically when loading or importing the CRL.
This property is read-only.

See Also

IssuerName

5.60.2.5 IssuerName

```
property IssuerName: TScDistinguishedName;
```

Description

Gets the distinguished name of the certification authority (CA) that issued the Certificate Revocation List (CRL). The property is set automatically when loading or importing the CRL.

This property is read-only.

See Also

Issuer

5.60.2.6 NextUpdate

```
property NextUpdate: TDateTime;
```

Description

The **NextUpdate** property indicates the date by which the next Certificate Revocation List (CRL) will be issued. The next CRL could be issued before the indicated date, but it will not be issued any later than the indicated date.

The property is set automatically when loading or importing the CRL.

This property is read-only.

See Also

ThisUpdate

5.60.2.7 RevokedCertificates

```
property RevokedCertificates: TScRevokedCertificates;
```

Description

RevokedCertificates maintains a list of the TScRevokedCertificate object references, that indicate information about revoked certificates.

Revoked certificates are listed by their serial numbers. Certificates revoked by the CA are uniquely identified by the certificate serial number.

This property is read-only.

5.60.2.8 Signature

```
property Signature: TBytes;
```

Description

Signature contains the signature of the Certificate Revocation List (CRL) content. The property is set automatically when loading or importing the CRL.

This property is read-only.

See Also

SignatureAlgorithm

5.60.2.9 SignatureAlgorithm

```
property SignatureAlgorithm: TScSignatureAlgorithmIdentifier;
```

Description

SignatureAlgorithm identifies the type of signature algorithm used by the Certificate Revocation List (CRL). The property is set automatically when loading or importing the CRL.

This property is read-only.

See Also

Signature

5.60.2.10 ThisUpdate

```
property ThisUpdate: TDateTime;
```

Description

The **ThisUpdate** property indicates the issue date of this Certificate Revocation List (CRL). The property is set automatically when loading or importing the CRL.

This property is read-only.

See Also

NextUpdate

5.60.2.11 Version

```
property Version: Integer;
```

Description

Gets the X.509 format version of a CRL. Possible values are 1 or 2. The property is set automatically when loading or importing the CRL.

This property is read-only.

5.60.3 Methods

5.60.3.1 CheckCompliance

```
function CheckCompliance(Cert: TScCertificate; CertDistributionPoint:
TScCRLDistributionPoint): boolean; overload;
function CheckCompliance(DeltaCRL: TScCRL): boolean; overload;
```

Description

Use the **CheckCompliance** method, to check compliance of the current Certificate Revocation List (CRL) with specified certificate in the the `Cert` parameter, or with delta CRL, specified in the `DeltaCRL` parameter.

The `CertDistributionPoint` parameter identifies how CRL information for the specified certificate is obtained.

See also

TScCRLDistributionPoint

VerifyCRLChain

5.60.3.2 Equals

```
function Equals(CRL: TScCRL): Boolean;
```

Description

Use the **Equals** method to compare content of two Certificate Revocation Lists (CRLs). If parameters of both CRLs coincide, the method returns True.

Note: If the Ready property of either of CRLs is False, the CRL will be loaded automatically.

5.60.3.3 ExportTo

```
procedure ExportTo(const FileName: string; const CertEncoding: TScCertificateEncodin
procedure ExportTo(Stream: TStream; const CertEncoding: TScCertificateEncoding = cfP
```

Description

Use this method to export the Certificate Revocation List (CRL) to file or to stream. The CRL can be stored in different formats.

Parameters:

- `FileName` - specifies the file name in which the CRL will be exported. If the file with the specified name does not exist, in will be created. The existent file will be overwritten.
- `Stream` - pointer to the stream in which the CRL will be exported. Data will be appended to the stream.
- `CertEncoding` - the data format which will be used for storing the CRL.

See also

ImportFrom

5.60.3.4 FindCertificate

```
function FindCertificate(Cert: TScCertificate; out Reason: TScCRLReason):
boolean;
```

Description

Call **FindCertificate** to determine if a specified certificate is referenced in the RevokedCertificates list. `Cert` is the object reference for which to search. If **FindCertificate** finds an item with a matching certificate, it returns True, and the `Reason` parameter is set to the reason for the certificate revocation. Otherwise the method returns False.

5.60.3.5 ImportFrom

```
procedure ImportFrom(const FileName: string; const Password: string =
''); overload;
procedure ImportFrom(Stream: TStream; const Password: string = '');
overload;
```

Description

Imports the Certificate Revocation List (CRL) from the specified file or stream.

The CRL can be stored in different formats. Format is determined automatically when loading the

CRL.

Parameters:

- `FileName` - determines the file name from which the CRL will be imported. If the file does not exists, an exception will be raised.
- `Stream` - a pointer to the stream that holds data for importing the CRL.
- `Password` - the password that is used for importing data decryption.

Note: If the CRL is loaded successfully, all properties becomes assigned, and the Ready property is set to True.

See also

ExportTo

5.60.3.6 VerifyCRLChain

```
procedure VerifyCRLChain(ParentCertificate: TScCertificate; var
StatusSet: TScCertificateStatusSet);
```

Description

Performs a Certificate Revocation List (CRL) validation using basic validation policy. The **VerifyCRLChain** method verifies that the CRL is valid and is signed by a certificate specified in the `ParentCertificate` parameter. All the errors which occur as a result of the CRL validation are added in the `StatusSet` parameter.

See also

TScCertificateStatusSet

5.61 TScStorageList

5.61.1 Description

Unit

ScBridge

Description

TScStorageList is an abstract class, which determines an interface to access lists of different object types (e.g. keys, users, certificates) stored in a storage.

Use the properties and methods of **TScStorageList** to:

- Add or delete objects from the list.
- Find out how many objects there are.
- Reload the list from the storage.
- Save modified data in the storage.

See also

TScStorage

TScKeyList

TScUserList

TScCertificateList

TScCRLList

5.61.2 Properties

5.61.2.1 Count

```
property Count: Integer;
```

Description

Use **Count** to determine the number of objects in the list.

This property is read-only.

5.61.2.2 Storage

```
property Storage: TScStorage;
```

Description

Check the value of the **Storage** property to determine the storage that is associated with the TScStorageList instance. Applications should not directly assign this property. It is assigned automatically when the instance is created.

5.61.3 Methods

5.61.3.1 Add

```
procedure Add(Item: TScStorageItem);
```

Description

Inserts an item to the end of the list. **Add** places the object after the last item, increments Count and,

if necessary, allocates memory.

At that the Item.StorageList property of the item is replaced to the current instance and the object becomes stored in Storage.

Note: When adding a key, the Ready property of the key must be set to True.

5.61.3.2 Clear

```
procedure Clear;
```

Description

Deletes all items from the list and frees all objects. **Clear** also deletes the objects from the Storage.

Call **Clear** to empty the objects array, set the Count to 0, and free the memory used to store the items array.

5.61.3.3 Flush

```
procedure Flush; virtual;
```

Description

Use this method to force data saving in the physical storage.

5.61.3.4 IndexOf

```
function IndexOf(Item: TScStorageItem): Integer;
```

Description

Call **IndexOf** to get the index for an item in the Items list. Specify the item as the `Item` parameter.

The first item in the array has index 0, the second key has index 1, and so on. If an item is not in the Items array, **IndexOf** returns -1.

5.61.3.5 Refresh

```
procedure Refresh;
```

Description

Reloads object list from the Storage.

Refresh deletes all items from the list and frees all objects. After this, it gets the list of items stored in the Storage, creates a corresponding object for each of them and adds it to the list.

5.61.3.6 Remove

```
procedure Remove(Item: TScStorageItem);
```

Description

Deletes the reference to the `Item` parameter from the Items list and delete the object from the Storage. **Remove** frees the object in addition to removing it from the list.

After the object is removed, all of the items that follow it are moved up in index position and the Count is reduced by one.

5.62 TScKeyList

5.62.1 Description

Unit

ScBridge

Description

TScKeyList is used by a storage to manage the key objects that correspond to keys in the storage.

Use the properties and methods of **TScKeyList** to:

- Access a specific key.
- Add or delete persistent key objects from the list.
- Find out how many keys there are.

See also

TScKey

5.62.2 Properties

5.62.2.1 Count

```
property Count: Integer;
```

Description

Use **Count** to determine the number of keys in the list.

This property is read-only.

See Also

Keys

5.62.2.2 Keys

```
property Keys[Index: Integer]: TScKey; default;
```

Description

Use **Keys** to obtain a TScKey object from the list. The `Index` parameter indicates the index of the key, where 0 is the index of the first key, 1 is the index of the second key, and so on.

Set **Keys** to assign the properties of another key object to one of the keys in the list. Reassigning an **Keys** index does not free the object that previously occupied that position in the list.

Use **Keys** with the Count property to iterate through all of the keys in the list.

See Also

Count

5.62.3 Methods

5.62.3.1 CheckKeyName

```
procedure CheckKeyName(const KeyName: string);
```

Description

Checks if a key name already exists in the list.

CheckKeyName checks for the key specified by `KeyName` in the Keys list. If the key with the specified name is already listed, **CheckKeyName** raises an EScError exception with the duplicate key name error message.

5.62.3.2 FindKey

```
function FindKey(const KeyName: string): TScKey;
```

Description

Call **FindKey** to determine if a specified key is referenced in the Keys list. `KeyName` is the name of the key for which to search. If **FindKey** finds a key with a matching name, it returns the TScKey object for the specified key. Otherwise it returns nil.

Note:

FindKey differs from the KeyByName method only when the named key is not in the list. When the key is not found, **FindKey** returns nil, while KeyByName raises an exception.

See also

KeyByName

5.62.3.3 KeyByName

```
function KeyByName(const KeyName: string): TScKey;
```

Description

Returns a key by specified key name.

Call the **KeyByName** method to retrieve key information for a key when only the key's name is known. **KeyByName** returns the TScKey object for the specified key. If the key can not be found, an exception is raised.

Note:

KeyByName differs from the FindKey method only when the named key is not in the list. When the key is not found, FindKey returns nil, while **KeyByName** raises an exception.

See also

FindKey

5.62.3.4 GetKeyNames

```
procedure GetKeyNames(List: TStrings);
```

Description

Call **GetKeyNames** to fill a list with the key names for all keys in the Keys list. `List` is a TStrings descendant created and maintained by the application.

See also

TScKey

5.62.3.5 Refresh

```
procedure Refresh;
```

Description

Reloads key list in the Keys array from the Storage.

Refresh deletes all items from the Keys list and frees all objects. After this, **Refresh** gets the list of keys stored in the Storage, creates a corresponding TScKey object for each of them and adds it to the list.

See also

Storage.Keys

5.63 TScUserList

5.63.1 Description

Unit

ScBridge

Description

TScUserList is used by a storage to manage the user objects that correspond to users in the storage. This class is used for storing the user list on the server.

Use the properties and methods of **TScUserList** to:

- Access a specific user.
- Add a new user object or delete persistent user objects from the list.
- Find out how many users there are.

See also

TScUser

5.63.2 Properties

5.63.2.1 Count

```
property Count: Integer;
```

Description

Use **Count** to determine the number of users in the list.

This property is read-only.

See Also

Users

5.63.2.2 Users

```
property Users[Index: Integer]: TScUser; default;
```

Description

Use **Users** to obtain a TScUser object from the list. The `Index` parameter indicates the index of the user, where 0 is the index of the first user, 1 is the index of the second user, and so on.

Set **Users** to assign the properties of another user object to one of the users in the list. Reassigning an **Users** index does not free the object that previously occupied that position in the list.

Use **Users** with the Count property to iterate through all of the users in the list.

See Also

Count

5.63.3 Methods

5.63.3.1 CheckUserName

```
procedure CheckUserName(const UserName: string);
```

Description

Checks for the user specified by `UserName` in the Users list. If the user with the specified name is already listed, **CheckUserName** raises an EScError exception with a duplicate user name error message.

See Also

Users

5.63.3.2 FindUser

```
function FindUser(const UserName: string): TScUser;
```

Description

Call **FindUser** to determine if a specified user is referenced in the Users list. `UserName` is the name of the user for which to search. If **FindUser** finds a user with a matching name, it returns the TScUser object for the specified user. Otherwise it returns nil.

Note:

FindUser differs from the UserByName method only when the named user is not in the list. When the user is not found, **FindUser** returns nil, while UserByName raises an exception.

See also

UserByName

5.63.3.3 UserByName

```
function UserByName(const UserName: string): TScUser;
```

Description

Call **UserByName** to determine if a specified user is referenced in the Users list. `UserName` is the name of the user for which to search. If **UserByName** finds a user with a matching name, it returns the TScUser object for the specified user. Otherwise it raises an exception.

Note:

UserByName differs from the FindUser method only when the named user is not in the list. When the user is not found, FindUser returns nil, while **UserByName** raises an exception.

See also

FindUser

5.63.3.4 GetUserNames

```
procedure GetUserNames(List: TStrings);
```

Description

Call **GetUserNames** to fill the `List` with the user names for all users in the Users list. `List` is a TStrings descendant created and maintained by the application.

See also

TScUser

5.63.3.5 Refresh

```
procedure Refresh;
```

Description

Reloads user list in the Users array from the Storage.

Refresh deletes all items from the Users list and frees all objects. After this, **Refresh** gets the list of users stored in the Storage, creates a corresponding TScUser object for each of them and adds it to

the list.

See also

Storage.Users

5.64 TScCertificateList

5.64.1 Description

Unit

ScBridge

Description

TScCertificateList is used by a storage to manage the certificate objects that correspond to certificates in the storage.

Use the properties and methods of **TScCertificateList** to:

- access a specific certificate;
- add a new certificate object or delete persistent certificate objects from the list;
- find out how many certificates there are.

See also

TScCertificate

5.64.2 Properties

5.64.2.1 Count

```
property Count: Integer;
```

Description

Use **Count** to determine the number of certificates in the list.

This property is read-only.

See Also

Certificates

5.64.2.2 Certificates

```
property Certificates[Index: Integer]: TScCertificate; default;
```

Description

Use **Certificates** to obtain a TScCertificate object from the list. The `Index` parameter indicates the index of the certificate, where 0 is the index of the first certificate, 1 is the index of the second certificate, and so on.

Set **Certificates** to assign the properties of another certificate object to one of the certificates in the list. Reassigning an **Certificates** index does not free the object that previously occupied that position in the list.

Use **Certificates** with the Count property to iterate through all of the certificates in the list.

See Also

Count

5.64.3 Methods

5.64.3.1 CertificateByName

```
function CertificateByName(const CertName: string): TScCertificate;
```

Description

Call **CertificateByName** to determine if a specified certificate is referenced in the Certificates list. `CertName` is the name of the certificate for which to search. If **CertificateByName** finds a certificate with a matching name, it returns the TScCertificate object for the specified certificate. Otherwise it raises an exception.

Note:

CertificateByName differs from the FindCertificate method only when the named certificate is not in the list. When the certificate is not found, FindCertificate returns nil, while **CertificateByName** raises an exception.

See also

FindCertificate

5.64.3.2 CheckCertificateName

```
procedure CheckCertificateName(const CertName: string);
```

Description

Checks for the certificate specified by `CertName` in the Certificates list. If the certificate with the specified name is already listed, **CheckCertificateName** raises the EScError exception.

See Also

TScCertificate

EScError

5.64.3.3 FindCertificate

```
function FindCertificate(const CertName: string): TScCertificate;
```

Description

Call **FindCertificate** to determine if a specified certificate appears in the Certificates list. `CertName` is the name of the certificate for which to search. If **FindCertificate** finds a certificate with a matching name, it returns the TScCertificate object for the specified certificate. Otherwise it returns nil.

Note:

FindCertificate differs from the CertificateByName method only when the named certificate is not in the list. When the certificate is not found, **FindCertificate** returns nil, while CertificateByName raises an exception.

See also

CertificateByName

5.64.3.4 GetCertificateNames

```
procedure GetCertificateNames(List: TStrings);
```

Description

Call **GetCertificateNames** to fill the `List` with the certificate names for all certificates in the Certificates list. `List` is a TStrings descendant created and maintained by the application.

See also

TScCertificate

5.64.3.5 Refresh

```
procedure Refresh;
```

Description

Reloads certificate list in the Certificates array from Storage.

Refresh deletes all items from the Certificates list and frees all objects. After this, **Refresh** gets the list of certificates stored in the Storage, creates a corresponding TScCertificate object for each of them and adds it to the list.

See also

Storage.Certificates

5.65 TScCRLList

5.65.1 Description

Unit

ScBridge

Description

TScCRLList is used by a storage to manage the Certificate Revocation List (CRL) objects that correspond to CRLs in the storage.

Use the properties and methods of **TScCRLList** to:

- access a specific CRL;
- add a new CRL object or delete persistent CRL objects from the list;
- find out how many CRLs there are.

See also

TScCRL

5.65.2 Properties

5.65.2.1 Count

```
property Count: Integer;
```

Description

Use **Count** to determine the number of CRLs in the list.

This property is read-only.

See Also

CRLs

5.65.2.2 CRLs

```
property CRLs[Index: Integer]: TScCRL; default;
```

Description

Use **CRLs** to obtain a TScCRL object from the list. The `Index` parameter indicates the index of the CRL, where 0 is the index of the first CRL, 1 is the index of the second CRL, and so on.

Set **CRLs** to assign the properties of another CRL object to one of the CRL in the list. Reassigning an **CRLs** index does not free the object that previously occupied that position in the list.

Use **CRLs** with the Count property to iterate through all of the CRLs in the list.

See Also

Count

5.65.3 Methods

5.65.3.1 CRLByName

```
function CRLByName(const CRLName: string): TScCRL;
```

Description

Call **CRLByName** to determine if a specified CRL is referenced in the CRLs list. `CRLName` is the name of the CRL for which to search. If **CRLByName** finds a CRL with a matching name, it returns the TScCRL object for the specified CRL. Otherwise it raises an exception.

Note:

CRLByName differs from the FindCRL method only when the named CRL is not in the list. When the CRL is not found, FindCRL returns nil, while **CRLByName** raises an exception.

See also

FindCRL

5.65.3.2 CheckCRLName

```
procedure CheckCRLName(const CRLName: string);
```

Description

Checks for the CRL specified by `CRLName` in the CRLs list. If the CRL with the specified name is

already listed, **CheckCRLName** raises the EScError exception.

See Also

TScCRL

EScError

5.65.3.3 FindCRL

```
function FindCRL(const CRLName: string): TScCRL;
```

Description

Call **FindCRL** to determine if a specified CRL appears in the CRLs list. `CRLName` is the name of the CRL for which to search. If **FindCRL** finds a CRL with a matching name, it returns the TScCRL object for the specified CRL. Otherwise it returns nil.

Note:

FindCRL differs from the CRLByName method only when the named CRL is not in the list. When the CRL is not found, **FindCRL** returns nil, while CRLByName raises an exception.

See also

CRLByName

5.65.3.4 GetCRLNames

```
procedure GetCRLNames(List: TStrings);
```

Description

Call **GetCRLNames** to fill the `List` with the CRL names for all CRLs in the CRLs list. `List` is a TStrings descendant created and maintained by the application.

See also

TScCRL

5.65.3.5 Refresh

```
procedure Refresh;
```

Description

Reloads CRL list in the CRLs array from Storage.

Refresh deletes all items from the CRLs list and frees all objects. After this, **Refresh** gets the list of CRLs stored in the Storage, creates a corresponding TScCRL object for each of them and adds it to the list.

See also

Storage.CRLs

5.66 TScStorage

5.66.1 Description

Unit

ScBridge

Description

TScStorage is an abstract class that describes the interface for storing asymmetric keys, certificates, and SSH server users.

TScStorage provides common interface for Memory Storage, File Storage, Registry Storage, and CryptoAPI Storage.

See also

TScMemoryStorage

TScFileStorage

TScRegStorage

TScCryptoAPIStorage

TScCertificate

TScKey

TScUser

5.66.2 Properties

5.66.2.1 Certificates

```
property Certificates: TScCertificateList;
```

Description

Lists all certificate objects of the storage.

Accessing certificates with the **Certificates** property is useful for applications that iterate over some

or all certificates in a storage.

5.66.2.2 CRLs

```
property CRLs: TScCRLList;
```

Description

Lists all Certificate Revocation List (CRL) objects of the storage.

Accessing CRLs with the **CRLs** property is useful for applications that iterate over some or all CRLs in a storage.

5.66.2.3 Keys

```
property Keys: TScKeyList;
```

Description

Lists all key objects of the storage.

Accessing keys with the **Key**s property is useful for applications that iterate over some or all keys in a storage.

5.66.2.4 StoreUserPassword

```
property StoreUserPassword: Boolean default True;
```

Description

The **StoreUserPassword** property determines if password should be saved for user objects. Set this property to True to store password together with the information about user in the plain form. Set this property to False to store only the hash of the password without plain password. This is necessary for security. For example, it is used to protect user passwords if user information was stolen by an intruder.

5.66.2.5 Users

```
property Users: TScUserList;
```

Description

Lists all user objects of the storage.

Accessing users with the **Users** property is useful for applications that iterate over some or all users

in a storage.

5.66.2.6 ReadOnly

```
property ReadOnly: Boolean;
```

Description

Determines whether the storage content can be changed.

Set the **ReadOnly** property to True to forbid removing, adding, and changing objects in the storage. Set the **ReadOnly** property to False to allow edit the storage.

5.66.3 Methods

5.66.3.1 DeleteStorage

```
procedure DeleteStorage; virtual;
```

Description

Physically deletes the storage and all its contents (keys, certificates, users, CRLs list).

5.66.4 Events

5.66.4.1 OnCheckUserPass

```
type
  TScCheckUserPass = procedure(ClientInfo: TScSSHClientInfo; const
    Password: string; var Accept: Boolean) of object;

property OnCheckUserPass: TScCheckUserPass;
```

Description

The **OnCheckUserPass** event arises when the password authentication is demanded on the server. The server previously verifies the given user and password in the storage and specifies the value for the `Accept` parameter. If you want to grant or deny access for the current connection attempt, you should change the `Accept` parameter value.

Parameters:

- `ClientInfo` - the information about the user to be authenticated;
- `Password` - a user password to be verified;
- `Accept` - if `Accept` is set to True, the user is allowed to connect to the server, otherwise the user is

not allowed to connect to the server.

See also

TScUser

5.66.4.2 OnCheckUserKey

type

```
TScCheckUserKey = procedure(ClientInfo: TScSSHClientInfo; Key: TScKey;
  var Accept: Boolean) of object;

property OnCheckUserKey: TScCheckUserKey;
```

Description

The **OnCheckUserKey** event arises when the authentication by the public key is demanded on the server. The server previously verifies the given user and key in the storage and specifies the value for the `Accept` parameter. If you want to grant or deny access for the current connection attempt, you should change the `Accept` parameter value.

Parameters:

- `ClientInfo` - the information about the user to be authenticated;
- `Key` - a user public key to be verified;
- `Accept` - if `Accept` is set to True, the user is allowed to connect to the server, otherwise the user is not allowed to connect to the server.

See also

TScUser

5.67 TScMemoryStorage

5.67.1 Description

Unit

ScBridge

Description

TScMemoryStorage is used to store information about keys, certificates, and users in RAM memory.

On the instance creating the Keys, Certificates and Users properties are empty, and

TScMemoryStorage frees all objects from these lists when the instance is itself destroyed.

Note: The **TScMemoryStorage** component is designed to store certificates and keys only in the RAM of the device. For this reason, certificates and keys are stored in the local memory of the running application at runtime. At design-time, they are stored in the local memory only when you work in the **TScMemoryStorage** property editor: if you close the project, the keys will be removed. You will need to import the keys again after reopening the project.

See Also

TScKey

TScCertificate

TScUser

5.68 TScFileStorage

5.68.1 Description

Unit

ScBridge

Description

TScFileStorage is used to store information about keys, certificates, and users in files.

Use the Path property to specify the path to store files.

The information about keys and users can be stored in encrypted form. Use the Algorithm and Password properties to specify encryption algorithm and password for storing objects in encrypted form.

Objects are loaded automatically when the Keys, Certificates and Users properties are accessed.

See Also

TScKey

TScCertificate

TScUser

5.68.2 Properties

5.68.2.1 Algorithm

```
property Algorithm: TScSymmetricAlgorithm;
```

Description

Information about keys and users can be stored in encrypted form. Use **Algorithm** to specify encrypting algorithm which will be used for encoding and decoding files when saving and loading.

Note: If the Password property is not assigned, files will not be encrypted when saving.

5.68.2.2 Password

```
property Password: string;
```

Description

Information about keys and users can be stored in encrypted form. Use the **Password** property to specify the password which will be used for encoding and decoding files when saving and loading. If **Password** is not assigned, files will not be encrypted when saving.

5.68.2.3 Path

```
property Path: string;
```

Description

Use this property to specify what directory will be used to store files that hold the information about certificates, keys and users. This information is loaded automatically when the Keys, Certificates, and Users properties are accessed.

Default value of the **Path** property is '.'. It means that the files will be stored in your application directory.

5.69 TScRegStorage

5.69.1 Description

Unit

ScBridge

Description

TScRegStorage is used to store information about keys, certificates, and users in the system registry.

Use the KeyPath property to specify the registry key to store the information.

Information about keys and users can be stored in an encrypted form. Use the Algorithm and Password properties to specify encryption algorithm and password for storing objects in the encrypted form.

Objects are loaded automatically when the Keys, Certificates, and Users properties are accessed.

See Also

TScKey

TScCertificate

TScUser

5.69.2 Properties

5.69.2.1 Algorithm

```
property Algorithm: TScSymmetricAlgorithm;
```

Description

Information about keys and users can be stored in an encrypted form. Use **Algorithm** to specify an encrypting algorithm which will be used for encoding and decoding files when saving and loading.

Note: If the Password property is not assigned, files will not be encrypted when saving.

5.69.2.2 KeyPath

```
property KeyPath: string;
```

Description

Use this property to specify what registry key will be used to store registry values that hold the information about certificates, keys and users. This information is loaded automatically when the Keys, Certificates and Users properties are accessed.

Default value of the **KeyPath** property is '\SOFTWARE\SecureBridge\'.

5.69.2.3 Password

```
property Password: string;
```

Description

Information about keys and users can be stored in an encrypted form. Use the **Password** property to specify the password which will be used for encoding and decoding files when saving and loading. If

Password is not assigned, files will not be encrypted when saving.

5.69.2.4 RootKey

```
property RootKey: NativeUInt;
```

Description

Use **RootKey** to determine the hierarchy of subkeys that the storage can access, or to specify the root key for the storage.

By default, **RootKey** is set to HKEY_CURRENT_USER. To change the root key, specify a valid integer value for the **RootKey** property.

5.70 TScCryptoAPIStorage

5.70.1 Description

Unit

ScCryptoAPIStorage

Description

TScCryptoAPIStorage is used to store information about keys and certificates in operation system and external certificate storages. It works through CryptoAPI. CryptoAPI is an application programming interface that can add authentication, encoding, and encryption to OS-based applications.

Use the CertProviderType property to specify the provided type which determines where the certificates will be stored. Keys are stored in system key containers. Each container has an unique system name for each ProviderName.

Objects are loaded automatically when the Certificates and Keys properties are accessed.

TScCryptoAPIStorage does not let storing information about users, therefore an exception will be raised when accessing to the Users property.

See Also

TScCertificate

TScKey

TScUser

5.70.2 Properties

5.70.2.1 CertLocation

type

```
  TScCertLocation = (clCurrentUser, clCurrentUserGroupPolicy,
clLocalMachine, clLocalMachineEnterprise, clLocalMachineGroupPolicy,
clCurrentService, clServices, clUsers);
```

```
property CertLocation: TScCertLocation;
```

Description

Use the **CertLocation** property to specify a system store location. Usage of this property is related on the value of the CertProviderType property.

Default value is clCurrentUser.

5.70.2.2 CertProviderType

type

```
  TScCertProviderType = (ptMemory, ptFile, ptRegistry, ptSystem,
ptSystemRegistry, ptPhysical);
```

```
property CertProviderType: TScCertProviderType;
```

Description

Specifies the provider type. It determines where certificates will be stored. Usage of CertLocation and CertStoreName properties is related to the value of **CertProviderType**.

Default value of this property is ptSystem.

Value	Meaning
ptMemory	Creates a certificate store in cached memory. Typically used to create a temporary store. The CertLocation and CertStoreName properties are not used for this provider type.
ptFile	Initializes the store with certificates from a file. The CertStoreName specifies the patch to the file with data. If the file with specified name does not exists when accessing the storage, the provider will try to create it. If the file exists, the storage will load certificates list from the file into the Certificates property. The CertLocation property is not used. When the storage refers to the underlying file, it opens it and blocks so that other application cannot open it.

`ptRegistry`	Initializes the store with certificates from a registry subkey. A registry key should be specified in the CertStoreName property, where the certificate list is stored. If the specified key does not exist, the provider will try to create it. The CertLocation property indicates the root key for the storage.
`ptSystem`	Initializes the store with certificates from the specified system store. The system store is a logical collection store that consists of one or more physical sibling stores. After the system store is opened, all of the physical stores that are associated with it are also opened and are added to the system store collection. The CertLocation indicates the system store location. The CertStoreName specifies the system store name. For each system store location, the predefined systems stores are: 'MY', 'Root', 'Trust', 'CA'.
`ptSystemRegistry`	Enters into storages set determined by ptSystem. Initializes the store with certificates from a physical registry store. The CertLocation indicates the system store location. The CertStoreName specifies a system store name. For each system store location, the predefined systems stores are: 'MY', 'Root', 'Trust', 'CA'.
`ptPhysical`	Enters into storages set determined by ptSystem. Initializes the store with certificates from a specified physical store. The CertLocation indicates the system store location. The CertStoreName consists of two parts separated with an intervening backslash (\), for example Root \.LocalMachine. Where Root is the name of the system store, and .LocalMachine is the name of the physical store.

See Also
CertLocation
CertStoreName

5.70.2.3 CertStoreName

```
property CertStoreName: string;
```

Description

Use the **CertStoreName** property to specify a store name. Usage of this property depends on the CertProviderType property value.

Default value of this property is 'MY'.

See Also

CertProviderType

5.70.2.4 ProviderName

```
property ProviderName: string;
```

Description

The name of a cryptographic provider. Cryptographic provider - an independent software module that actually performs cryptography algorithms for authentication, encoding, and encryption.

The default provider will be used if the value of this property equals to an empty string. It is recommended to use the default provider.

See also

GetProviderNames

5.70.3 Methods

5.70.3.1 GetProviderNames

```
procedure GetProviderNames(List: TStrings);
```

Description

Call **GetProviderNames** to fill the `List` with the cryptographic service providers available on a computer. `List` is a TStrings descendant created and maintained by the application.

Cryptographic service provider is an independent software module that actually performs cryptography algorithms for authentication, encoding, and encryption.

See also

ProviderName

5.71 TScRandom

5.71.1 Description

Unit

ScRNG

Description

The **TScRandom** class implements functionality of a pseudo-random number generator. It produces a sequence of numbers that meet certain statistical requirements for randomness.

The random number generation starts from a seed value. To set the start value, use the Randomize method. If the same seed is used repeatedly, the same series of numbers is generated. Seed can be generated by using processor step counter, system timer information, information of random mouse movements or pressure of keyboard keys.

Note: Generation of a reliable starting sequence for the random-number generator is required to ensure high security level.

To improve performance, create only one **TScRandom** object to generate many random numbers, instead of creating a new **TScRandom** object to generate one random number.

See also

TScRandom_LFSR

Possible attack types and countermeasures

5.71.2 Methods

5.71.2.1 Randomize

```
procedure Randomize(Seed: Pointer; Count: integer); overload;
procedure Randomize(const Seed: TBytes; const Offset, Count: Integer);
overload; virtual;
procedure Randomize(const Seed: TBytes); overload;
procedure Randomize(const Seed: string); overload;
procedure Randomize(Seed: TStream); overload;
procedure Randomize; overload;
```

Description

Use the **Randomize** method to set a sequence of numbers, that will be used to generate a random-number sequence.

If **Randomize** without parameters is called, `Seed` based on the system timer readout is used.

Parameters:

- `Seed` - the number sequence that is used for calculation of the starting value for a pseudo-random number sequence;
- `Offset` - zero-based byte offset in `Seed`, that points to the beginning of the data location;
- `Count` - data length.

5.71.2.2 Random

```
procedure Random(var buf: TBytes; const Offset, Count: integer); virtual;
```

Description

Fills the elements of a specified array of bytes with random numbers.

To initialize the random number generator, add a call to Randomize before making any calls to **Random**.

Parameters:

- `Buffer` - an array of bytes to keep random numbers;
- `Offset` - zero-based byte offset in `Buffer`, that points to the beginning of the data location to fill;
- `Count` - data length.

5.72 TScRandom_LFSR

5.72.1 Description

Unit

ScRNG

Description

The **TScRandom_LFSR** class is a descendant of the TScRandom class. It implements Linear Feedback Shift Register with variable Period from 2^32 -1 to 2^2032 -1 method for generating random numbers.

Note: Generation of a reliable starting sequence for the random-number generator is required to ensure high security level.

See also

TScRandom

Possible attack types and countermeasures

5.73 TScIdIOHandler

5.73.1 Description

Unit

ScIndy

Description

The **TScIdIOHandler** component is a wrapper for TScSSHChannel with an Indy-compatible interface.

See Also

TScSSHChannel

TScSSHClient

5.73.2 Properties

5.73.2.1 Client

```
property Client: TScSSHClient;
```

Description

Determines secure physical connection between the client and the SSH server that will be used for data transferring.

5.74 TScStorageItem

5.74.1 Description

Unit

ScBridge

Description

The **TScStorageItem** class is an abstract class, which represents an item stored in a storage list.

TScStorageItem is a base class, which is the ancestor for different object types (e.g. keys, users, certificates) stored in a storage. A TScStorageList holds a group of **TScStorageItem** objects. Each **TScStorageItem** has a StorageList property that points to the TScStorageList object to which the item belongs.

See also

TScKey

TScUser

TScCertificate

TScCRL

5.74.2 Properties

5.74.2.1 Ready

```
property Ready: boolean;
```

Description

The **Ready** property determines whether the object is ready to use. Set **Ready** to True, to load data from StorageList automatically. If the StorageList is not assigned, an exception will be raised.

See Also

StorageList

5.74.2.2 StorageList

```
property StorageList: TScStorageList;
```

Description

The **StorageList** property is used for automatic loading and storing objects in Storage. If the object was not loaded, and you are trying to invoke functions that use data of the one, it is automatically loaded from underlying Storage using item name.

See Also

Ready

5.74.3 Methods

5.74.3.1 Assign

```
procedure Assign(Source: TPersistent); override;
```

Description

Copies the contents of another similar object. **Assign** copies properties and other attributes of the specified `Source` object to the current object.

5.74.3.2 Create

```
constructor Create(AStorageList: TScStorageList = nil); virtual;
```

Description

Create **TScStorageItem** instance.

The `AStorageList` parameter points to the TScStorageList object to which the item belongs. The StorageList property is set from the value of this parameter.

5.75 TScKey

5.75.1 Description

Unit

ScBridge

Description

The **TScKey** class is used for working with asymmetric keys of RSA, DSA, or ECC types. The algorithm of the key is determined by the Algorithm property.

In asymmetric encryption two key are used. The first key is used for data decryption and signing (private key), the second key is used for data encrypting (public key).

TScKey lets you working both with private and public keys. The private key contains both parts - private part and public part. You can use the IsPrivate property to determine whether the key is private or public.

The key length is determined by the BitCount property. The more key length, the higher its resistance to breaking.

The **TScKey** class lets you generating new keys by using the Generate method. To store keys, different formats are used. To load or save a key in one of formats, you should use the ImportFrom or ExportTo methods correspondingly. To store a set of keys in storage, the KeyList property is used.

The **TScKey** lets you to sign data with the private key and verify signature with the public key by using the Sign and VerifySign methods. The data signing is used for checking data integrity.

Also **TScKey** can encrypt and decrypt data with Encrypt and Decrypt methods.

See Also

TScKeyList

TScStorage

5.75.2 Properties

5.75.2.1 Algorithm

```
property Algorithm: TScAsymmetricAlgorithm;
```

Description

The **Algorithm** property stores the name of asymmetric algorithm. It is set automatically when loading, importing, or generating the key.

This property is read-only.

See Also

Generate

GenerateEC

5.75.2.2 BitCount

```
property BitCount: integer;
```

Description

The **BitCount** property stores the length of the key. It is set automatically when loading, importing, or generating the key.

This property is read-only.

See Also

Generate

5.75.2.3 DSAData

```
property DSAData: TScDSAData;
```

Description

The **DSAData** property stores the data of the DSA key. This property is set only if the Algorithm property is set to aaDSA.

It is set automatically when loading, importing or generating the key.

This property is read-only.

See Also

Generate

5.75.2.4 ECData

```
property ECData: TScECData;
```

Description

The **ECData** property stores the data of the EC key. This property is set only if the Algorithm property is set to aaEC.

It is set automatically when loading, importing or generating the key.

This property is read-only.

See Also

GenerateEC

5.75.2.5 IsPrivate

```
property IsPrivate: Boolean;
```

Description

The **IsPrivate** property determines whether the key is private or public. **IsPrivate** is set automatically when loading, importing, or generating the key. The private key always contains the public key.

This property is read-only.

See Also

ImportFrom

ExportTo

5.75.2.6 KeyList

```
property KeyList: TScKeyList;
```

Description

The **KeyList** property is used for automatic loading and storing keys in Storage. If the key was not loaded or generated, and you are trying to invoke functions that use data of the key, it is automatically loaded from underlying Storage using KeyName.

See Also

Ready

KeyList.Storage

5.75.2.7 KeyName

```
property KeyName: string;
```

Description

The **KeyName** property represents the key name, that is used for automatic loading and saving the key in KeyList.

See Also

TScStorage

5.75.2.8 OAEPParams

```
property OAEPParams: TScOAEPParams;
```

Description

OAEPParams specifies the OAEP padding parameters to use with RSA encryption or decryption operations.

See Also

Encrypt

Decrypt

TScPaddingMode

5.75.2.9 PSSParams

```
property PSSParams: TScPSSParams;
```

Description

PSSParams specifies the PSS padding parameters to use with RSA signing and verifying signature operations.

See Also

Sign

VerifySign

TScPaddingMode

5.75.2.10 RSAData

```
property RSAData: TScRSAData;
```

Description

The **RSAData** property stores the data of the RSA key. This property is set only if the Algorithm property is set to aaRSA.

It is set automatically when loading, importing or generating the key.

This property is read-only.

See Also

Generate

5.75.2.11 Ready

```
property Ready: Boolean;
```

Description

The **Ready** property determines whether the key is ready to use. Set **Ready** to True, to load data from KeyList automatically. If the KeyList is not assigned, an exception will be raised.

This property is set automatically when the key is generated.

Note: If the key was not loaded or generated, and you are trying to invoke functions that use data of the key, it is automatically loaded from underlying Storage using KeyName.

See Also

KeyName

KeyList

Generate

5.75.3 Methods

5.75.3.1 Decrypt

```
function Decrypt(const Data: TBytes; Padding: TScPaddingMode = pmPKCS2):
```

```
TBytes;
```

Description

Use the **Decrypt** method to decrypt data with the private key by using the specified padding. This method returns the source data that was encrypted by the Encrypt method.

If the key is not private (IsPrivate = False), the exception will be raised.

Note: If the Ready property is False, the key will be automatically loaded.

See also

Encrypt

5.75.3.2 Encrypt

```
function Encrypt(const Data: TBytes; Padding: TScPaddingMode = pmPKCS2):
TBytes;
```

Description

Use the **Encrypt** method to encrypt data with the public key using the specified padding. This method returns an encrypted data.

The `Padding` parameter can be equal only to pmNone, pmPKCS2 or pmOAEP values. The maximum block size for PKCS2 padding mode should be 11 bytes less than a key size, and for OAEP padding mode - (2 + 2*HashLength) bytes less than a key size. The OAEP padding parameters can be specified by the OAEPParams property.

Note: If the Ready property is False, the key will be automatically loaded.

See also

OAEPParams

Decrypt

5.75.3.3 Equals

```
function Equals(Key: TScKey): Boolean;
```

Description

Use the **Equals** method to compare content of two keys. If data of both keys coincide, the function returns True. If either of keys is a public key, only public constituents of keys are compared. If both keys are private, both public and private constituents of keys are compared.

If the Ready property of either of the keys is False, the key will be loaded automatically.

5.75.3.4 ExportTo

```
procedure ExportTo(const FileName: string; const PublicKeyOnly: Boolean;
const Password: string; const Cipher: TScSymmetricAlgorithm =
saTripleDES; const KeyFormat: TScKeyFormat = kfDefault; const Comment:
string = ''); overload;
procedure ExportTo(Stream: TStream; const PublicKeyOnly: Boolean; const
Password: string; const Cipher: TScSymmetricAlgorithm = saTripleDES;
const KeyFormat: TScKeyFormat = kfDefault; const Comment: string = '');
overload;
```

Description

Use this method to export the key to file or to stream.

The key can be exported in different formats. Some formats let store only private keys, other - both private and public. It is possible to store the key in encrypted form to protect it from illegal access. In this case you should specify encryption algorithm and password.

Some formats also let you store additional information about the key except key data.

Parameters:

- `FileName` - specifies file name in which the key will be exported. If the file with specified name does not exist, in will be created. The existent file will be overwritten.
- `Stream` - pointer to the stream in which the key will be exported. Data will be appended to the stream.
- `PublicKeyOnly` - determines, whether only the public key or both public and private keys will be exported. If the current key contains only public constituent, an attempt to save the private key will lead to raising the exception.
- `Password` - determines password that is used by encryption algorithm for storing the key in encrypted form. If the Password is not specified, the key will be stored in open form.
- `Cipher` - encryption algorithm name that is used for storing the key in encrypted form.
- `KeyFormat` - the data format which will be used for storing the key.
- `Comment` - an additional information defined by user.

Note: You can only store the public keys in open form. Therefore, when you try to save a public key with the `Password` parameter assigned, an exception will be raised.

See also

TScKeyFormat

ImportFrom

Generate

5.75.3.5 Generate

```
procedure Generate(const Algorithm: TScAsymmetricAlgorithm; const
BitCount: integer; Random: TScRandom = nil);
```

Description

Generates a new RSA or DSA key, and if the KeyName and KeyList parameters are specified, automatically saves it. If the key is created successfully, the Ready property is set to True.

Random data generated by the specified random number generator is used for generating keys. If Random is nil, the default random number generator is used.

The Algorithm parameter can be set only to the aaRSA or aaDSA values. To generate an Elliptic Curve cryptography key, you should call the GenerateEC method.

Parameters:

- Algorithm - asymmetric algorithm that determines a type of the key to be generated.
- BitCount - key length in bits.
- Random - pointer to the random number generator that is used for getting random data.

Note: The key length specified in the BitCount parameter determines the resistance to breaking. Now for usual tasks the recommended key length is 2048 bits, for crucial tasks - 4096 bits.

See also

GenerateEC

ImportFrom

5.75.3.6 GenerateEC

```
procedure GenerateEC(const ECName: TScECName; Random: TScRandom = nil);
```

Description

Generates a new Elliptic Curve key, and if the KeyName and KeyList parameters are specified, automatically saves it. If the key is created successfully, the Ready property is set to True.

Random data generated by the specified random number generator is used for generating keys. If Random is nil, the default random number generator is used.

To generate RSA or DSA cryptography key, you should call the Generate method.

Parameters:

- ECName - the named Elliptic curve over F_p and F_{2m} cryptography algorithms that describes

parameters of the key to be generated.
- `Random` - pointer to the random number generator used for getting random data.

See also

Generate

ImportFrom

5.75.3.7 GetFingerprint

```
procedure GetFingerprint(const HashAlg: TScHashAlgorithm; out
Fingerprint: TBytes); overload;
procedure GetFingerprint(const HashAlg: TScHashAlgorithm; out
Fingerprint: string); overload;
```

Description

Returns the key print into the `Fingerprint` parameter. The print is formed by using the specified hash algorithm `HashAlg`.

See also

Ready

5.75.3.8 ImportFrom

```
procedure ImportFrom(const FileName: string; const Password: string; out
Comment: string); overload;
procedure ImportFrom(const FileName: string; const Password: string =
''); overload;
procedure ImportFrom(Stream: TStream; const Password: string; out
Comment: string); overload;
procedure ImportFrom(Stream: TStream; const Password: string = '');
overload;
```

Description

Imports the key from the specified file or stream.

The key can be stored in different formats. Format is determined automatically when loading the key. Some formats lets store the key in the encrypted form. If the key was encrypted, it is required to specify the password for decryption.

Parameters:

- `FileName` - determined the file name from which the key will be imported. If the file does not exists, the exception will be raised.

- `Stream` - a pointer to the stream that holds data for importing the key.
- `Password` - the password that is used for data decryption.
- `Comment` - additional information specified by the user when saving the key is returned by this parameter.

Note: If the key is loaded successfully, the Algorithm, BitCount, IsPrivate becomes assigned, and the Ready property is set to True.

See also

ExportTo

Generate

5.75.3.9 Sign

```
function Sign(const Data: TBytes; HashAlg: TScHashAlgorithm = haSHA1;
Padding: TScPaddingMode = pmPKCS1): TBytes;
```

Description

Use the **Sign** method, to sign necessary data by using the private key. The function returns the signature of the specified data.

The `Padding` parameter can be equal only to pmPKCS1 or pmPSS values. The PSS padding parameters can be specified by the PSSParams property.

Use this signature to verify the data integrity. If the data substitution is possible when the data is transferred, it is required to transfer the data signature along with the data itself. In this case the receiver must have the public constituent of the key, that is used to verify the signature.

To verify the signature use the VerifySign method.

Note: If the Ready property is False, the key will be automatically loaded.

Note: If the key is not private (IsPrivate = False), the exception will be raised.

See also

PSSParams

VerifySign

5.75.3.10 VerifySign

```
function VerifySign(const Data, Sign: TBytes; HashAlg: TScHashAlgorithm =
```

```
haSHA1; Padding: TScPaddingMode = pmPKCS1): Boolean;
```

Description

The **VerifySign** method verifies whether the signature is correct for specified `Data` using the public key. If the signature is correct, the function returns True.

The `Padding` parameter can be equal only to pmPKCS1 or pmPSS values.

To get data signature the Sign method should be used.

Note: If the Ready property is False, the key will be automatically loaded.

See also

Sign

5.76 TScUser

5.76.1 Description

Unit

ScBridge

Description

TScUser holds data about a user. This data is used by SSH server when the SSH client authentication is performed.

To store a user list in a the Storage, use the UserList property.

See Also

TScUserList

TScStorage

5.76.2 Properties

5.76.2.1 Authentications

```
property Authentications: TScUserAuthentications;
```

Description

The **Authentications** property contains available authentication methods.

Value	Meaning
`uaPublicKey`	Determines whether the connecting by key is allowed. If the **Authentications** property does not contain this value, the Key property is nil. When this value is added, the Key object is created automatically, when it is removed, the Key is freed.
`uaPassword`	Determines whether the connecting by password is allowed. In this case the password is obtained from the Password property.
`uaOSAuthentication`	Determines whether the connecting by password is allowed. In this case the system password is used. The Password property can be empty.

Note: If both `uaPassword` and `uaOSAuthentication` values are in the set, at first the password from the Password property is used. If the authentication fails, then the system password is used.

5.76.2.2 Domain

```
property Domain: string;
```

Description

The **Domain** property holds the domain in which the user is placed and that is used for OS authentication. The OS authentication is possible if the Authentications property includes the uaOSAuthentication value.

See Also

Authentications

5.76.2.3 ExtData

```
property ExtData: string;
```

Description

The **ExtData** property holds the information to additionally describe a user in any form. **ExtData** has no predefined meaning. A developer sets this property by himself and then can use it in different places for additional user identification.

See Also

Authentications

5.76.2.4 HashPassword

```
property HashPassword: string;
```

Description

The **HashPassword** property holds the password hash that is used for password authentication. The authentication by password is possible if the Authentications property includes the uaPassword value.

This property is read-only and is changed when the Password property is changed or when data is loaded from storage. If this property is not empty and the Password property is empty, only hash of the password is stored in storage.

See Also

Password

TScStorage.StoreUserPassword

5.76.2.5 HomePath

```
property HomePath: string;
```

Description

The **HomePath** property represents path to a root directory used by SFTP server for this user.

See Also

TScSFTPServer.DefaultRootPath

5.76.2.6 Key

```
property Key: TScKey;
```

Description

The **Key** property holds the public key of the user that is used for authentication by key. The authentication by key is possible if the Authentications property includes the uaPublicKey value.

Note: If the Authentications property does not include the uaPublicKey value, the **Key** is nil.

See Also

Authentications

5.76.2.7 Password

```
property Password: string;
```

Description

The **Password** property holds the password that is used for password authentication. The authentication by password is possible if the Authentications property includes the uaPassword value.

If this property is empty and the HashPassword property is not empty, only hash of the password is stored in storage.

See Also

Authentications

TScStorage.StoreUserPassword

5.76.2.8 SSHChannelPermissions

```
property SSHChannelPermissions: TScSSHChannelPermissions;
```

Description

The **SSHChannelPermissions** property contains available user permissions, which he has to establish a new SSH channel.

These rights are used only by SSH server to check a possibility of opening a new SSH channel at the client's request.

SSH client does not use this property.

5.76.2.9 UserList

```
property UserList: TScUserList;
```

Description

UserList is used for automatic loading and saving the information about user in the Storage.

See Also

UserName

UserList.Storage

5.76.2.10 UserName

```
property UserName: string;
```

Description

UserName is used for authentication, and for automatic loading and saving data in the UserList.

See Also

TScStorage

5.77 TScCollectionItem

5.77.1 Description

Unit

ScUtils

Description

The **TScCollectionItem** class is a descendant of the TCollectionItem class, and it represents an item in a collection.

TScCollectionItem is a base class, which is the ancestor for all objects that have capability to represent self as a string. For this it declares the AsString property.

A TScCollection holds a group of **TScCollectionItem** objects. Each **TScCollectionItem** has a Collection property that points to the TScCollection object to which the item belongs.

See Also

TScCollectionItemClass

TScCollection

TScSSHCipherItem

TScSSHHostKeyAlgorithmItem

TScSSHHMacAlgorithmItem

TScSSHKeyExchangeAlgorithmItem

TScSSLCipherSuiteItem

TScSFTPACEItem

5.77.2 Properties

5.77.2.1 AsString

```
property AsString: string;
```

Description

Use **AsString** to represent the item as a string.

See Also

RawData

5.78 TScCollection

5.78.1 Description

Unit

ScUtils

Description

The **TScCollection** class is a descendant of the TCollection class, and it is a container for TScCollectionItem objects.

TScCollection have capability to represent the collection as a string. For this it declares the AsString property.

Each **TScCollection** holds a group of TScCollectionItem descendants. **TScCollection** maintains an index of the collection items in its Items array. The Count property contains the number of items in the collection.

See Also

TScCollectionItem

TScSSHCiphers

TScSSHHostKeyAlgorithms

TScSSHHMacAlgorithms

TScSSHKeyExchangeAlgorithms

TScSSLCipherSuites

TScSFTPACEs

5.78.2 Properties

5.78.2.1 AsString

```
property AsString: string;
```

Description

Use **AsString** to represent the collection as a string. The collection merges all its items into a string, separated by commas.

5.78.3 Methods

5.78.3.1 Create

```
constructor Create(AOwner: TPersistent; ItemClass:
TScCollectionItemClass);
```

Description

Creates and initializes a collection.

`ItemClass` identifies the TScCollectionItem descendants that must be used to represent the items in the collection.

5.78.4 Events

5.78.4.1 OnChanged

```
property OnChanged: TNotifyEvent;
```

Description

Occurs after the changes in a collection are complete.

Whenever items in the collection are added, deleted, moved, or modified, the **OnChanged** event occurs.

5.79 TScSSHCipherItem

5.79.1 Description

Unit

ScSSHUtils

Description

The **TScSSHCipherItem** class is a descendant of the TScCollectionItem class, and it represents a symmetric encryption algorithm in the SSH format.

See Also

TScSSHCiphers

5.79.2 Properties

5.79.2.1 Algorithm

```
property Algorithm: TScSymmetricAlgorithm;
```

Description

Algorithm represents a symmetric encryption algorithm.

See Also

AsString

5.80 TScSSHCiphers

5.80.1 Description

Unit

ScSSHUtils

Description

The **TScSSHCiphers** class is a descendant of the TScCollection class, and it is a container for TScSSHCipherItem objects.

TScSSHCiphers keeps list symmetric encryption algorithms and represents them in the SSH format.

See Also

TScSSHCipherItem

5.81 TScSSHHostKeyAlgorithmItem

5.81.1 Description

Unit

ScSSHUtils

Description

The **TScSSHHostKeyAlgorithmItem** class is a descendant of the TScCollectionItem class, and it represents an asymmetric public key algorithm in the SSH format.

See Also

TScSSHHostKeyAlgorithms

5.81.2 Properties

5.81.2.1 Algorithm

```
property Algorithm: TScAsymmetricAlgorithm;
```

Description

Algorithm represents an asymmetric public key algorithm.

See Also

AsString

5.82 TScSSHHostKeyAlgorithms

5.82.1 Description

Unit

ScSSHUtils

Description

The **TScSSHHostKeyAlgorithms** class is a descendant of the TScCollection class, and it is a container for TScSSHHostKeyAlgorithmItem objects.

TScSSHHostKeyAlgorithms keeps a list of asymmetric public key algorithms and represents them in the SSH format.

See Also

TScSSHHostKeyAlgorithmItem

5.83 TScSSHHMacAlgorithmItem

5.83.1 Description

Unit

ScSSHUtils

Description

The **TScSSHHMacAlgorithmItem** class is a descendant of the TScCollectionItem class, and it represents HMAC algorithm, which can be accepted during the SSH handshake, in the SSH format.

See Also

TScSSHHMacAlgorithms

5.83.2 Properties

5.83.2.1 Algorithm

```
property Algorithm: TScHMACAlgorithm;
```

Description

Algorithm represents an HMAC algorithm which can be accepted during the SSH handshake.

See Also

AsString

5.84 TScSSHHMacAlgorithms

5.84.1 Description

Unit

ScSSHUtils

Description

The **TScSSHHMacAlgorithms** class is a descendant of the TScCollection class, and it is a container for TScSSHHMacAlgorithmItem objects.

TScSSHHMacAlgorithms keeps a list of HMAC algorithms which can be accepted during the SSH handshake, and represents them in the SSH format.

See Also

TScSSHHMacAlgorithmItem

5.85 TScSSHKeyExchangeAlgorithmItem

5.85.1 Description

Unit

ScSSHUtils

Description

The **TScSSHKeyExchangeAlgorithmItem** class is a descendant of the TScCollectionItem class,

and it represents a key exchange algorithm, which can be accepted during the SSH handshake, in the SSH format.

See Also

TScSSHKeyExchangeAlgorithms

5.85.2 Properties

5.85.2.1 Algorithm

```
property Algorithm: TScKeyExchangeAlgorithm;
```

Description

Algorithm represents a key exchange algorithm which can be accepted during the SSH handshake.

See Also

AsString

5.86 TScSSHKeyExchangeAlgorithms

5.86.1 Description

Unit

ScSSHUtils

Description

The **TScSSHKeyExchangeAlgorithms** class is a descendant of the TScCollection class, and it is a container for TScSSHKeyExchangeAlgorithmItem objects.

TScSSHKeyExchangeAlgorithms keeps a list of key exchange algorithms which can be accepted during the SSH handshake, and represents them in the SSH format.

See Also

TScSSHKeyExchangeAlgorithmItem

5.87 THttpOptions

5.87.1 Description

Unit

ScVio

Description

The **THttpOptions** class contains settings for HTTP connection.

For more information on HTTP tunneling refer to the Network Tunneling article.

See Also

Network Tunneling

TScSSHClient.HttpOptions

TScSSLClient.HttpOptions

5.87.2 Properties

5.87.2.1 Enabled

```
property Enabled: boolean; default False;
```

Description

The **Enabled** property indicates whether the HTTP connection is enabled.

See Also

Network Tunneling

5.87.2.2 Password

```
property Password: string;
```

Description

The **Password** property holds the password for HTTP authorization.

See Also

Username

5.87.2.3 ProxyOptions

```
property ProxyOptions: TProxyOptions;
```

Description

The **ProxyOptions** property holds a TProxyOptions object that contains settings for proxy

connection.

If it is necessary to connect to server in another network, sometimes the client can reach it only through proxy. In this case in addition to Url string you have to setup **ProxyOptions**.

5.87.2.4 TrustServerCertificate

```
property TrustServerCertificate: boolean; default False;
```

Description

The **TrustServerCertificate** property specifies if the server SSL certificate will be validated by client.

When **TrustServerCertificate** is set to True, the client will not validate the server SSL certificate.

5.87.2.5 Url

```
property Url: string;
```

Description

The **Url** property holds the url of the tunneling PHP script. For example, if the script is in the server root, the url can be the following: `http://server/tunnel.php`

5.87.2.6 Username

```
property Username: string;
```

Description

The **Username** property holds the user name for HTTP authorization.

See Also

Password

5.87.3 Methods

5.87.3.1 Equals

```
function Equals(HttpOptions: THttpOptions): boolean;
```

Description

Use the **Equals** method to compare content of two THttpOptions objects. If both values coincide, the method returns True.

5.88 TProxyOptions

5.88.1 Description

Unit

ScVio

Description

The **TProxyOptions** class is used when connecting through a proxy server.

A proxy connection can be established using the HTTP or SOCKS protocol. To enable a SOCKS connection, use the SocksVersion property.

For more information on HTTP tunneling, see Network Tunneling.

See Also

Network Tunneling

THttpOptions.ProxyOptions

5.88.2 Properties

5.88.2.1 Hostname

```
property Hostname: string;
```

Description

The **Hostname** property holds the host name or IP address to connect to proxy server.

5.88.2.2 Password

```
property Password: string;
```

Description

The **Password** property holds the password for the proxy server account.

See Also

Username

5.88.2.3 Port

```
property Port: integer;
```

Description

Use the **Port** property to specify the port number for TCP/IP connection with proxy server.

5.88.2.4 ResolveDNS

```
property ResolveDNS: boolean; default True;
```

Description

The **ResolveDSN** property specifies whether to resolve the DNS of the target server to an IP address when passing it to the SOCKS server.

When **ResolveDSN** is True, the DNS will be resolved to an IP address and then passed to the proxy server. Otherwise, the unmodified DNS of the target server will be passed to the proxy server.

The property has effect only when used with SOCKS5. The SOCKS version is specified in the SocksVersion property.

The default value is True.

See Also

SocksVersion

5.88.2.5 SocksVersion

```
property SocksVersion: TCRSocksVersion; default svNoSocks;
```

Description

The **SocksVersion** property holds the SOCKS protocol version. The SOCKS protocol is intended to exchange network packages between the client and server by routing traffic to the actual server through a firewall.

By default SOCKS is not used (the value is `svNoSocks`)and a connection to the proxy server will be established through HTTP.

See Also

ResolveDNS

5.88.2.6 Username

```
property Username: string;
```

Description

The **Username** property holds the proxy server account name.

See Also

Password

5.89 TScSSHCustomChannel

5.89.1 Description

Unit

ScSSHChannel

Description

TScSSHCustomChannel is an abstract class that ensures the logical connection creation via the SSH tunnel and gives an interface for data exchange. Physically a secure connection is provided by an SSH client that can be assigned to the Client property.

To exchange data, you can use ReadBuffer, ReadString and WriteBuffer, WriteString methods.

See Also

TScSSHChannel

TScSSHShell

TScSSHClient

5.89.2 Properties

5.89.2.1 ChannelInfo

```
property ChannelInfo: TScSSHChannelInfo;
```

Description

The **ChannelInfo** property holds the information about the SSH channel. **ChannelInfo** is initialized

after the channel is opened.

5.89.2.2 Client

```
property Client: TScSSHClient;
```

Description

The **Client** property determines the physical connection between SSH client and SSH server. This connection is used to exchange data. To create a logical connection, the **Client** property must be set.

This property can be set at design time by selecting a TScSSHClient object from the provided list.

At runtime, set the **Client** property to reference of an existent TScSSHClient object.

5.89.2.3 Connected

```
property Connected: Boolean;
```

Description

Connected determines, whether the connection is established. Switch it to True, to establish a logical connection to an SSH server. Switch it to False, to close a logical connection to the SSH server.

Note: To establish a connection, it is required to set the Client property, that provides secure physical connection to the SSH server. After the connection is established, the information is transferred through the secure channel.

See Also

Client

5.89.2.4 EventsCallMode

```
property EventsCallMode: TScEventCallMode; default ecAsynchronous;
```

Description

The **EventsCallMode** property determines how the OnAsyncError and OnAsyncReceive event handlers will be called. The thing is that data coming from the server is processed in a separate thread of a SSH connection. And the call of the event handlers can occur in a different way to synchronize with a main thread of the application.

The default value is the `ecAsynchronous` mode, events are added to a queue and then asynchronously synchronized from this queue with the main thread. This allows not slowing down the thread in which the events occur and at the same time calling the event handlers in the main thread.

When setting the property to the `ecSynchronous` value, the event call will be immediately synchronized with the main thread.

When setting the property to the `ecDirectly` value, there is no synchronization with the main thread.

Default value is the `ecAsynchronous` mode.

See Also

OnAsyncError

OnAsyncReceive

5.89.2.5 InCount

```
property InCount: integer;
```

Description

Determines data size received from the server. This data can be obtained using the ReadBuffer method.

See Also

OnAsyncReceive

ReadBuffer

5.89.2.6 Timeout

```
property Timeout: integer; default 15;
```

Description

Determines amount of time during which the client makes attempts to obtain data from the server.

It is measured in seconds. Default value is 15 seconds.

See Also

ReadBuffer

5.89.2.7 UseUnicode

```
property UseUnicode: boolean; default False;
```

Description

The **UseUnicode** property specifies, whether UTF8 conversion will be used by the channel when data

is transferred to/from the stream.

If **UseUnicode** equals False, data will be converted to ASCII format, otherwise, it will be converted from/to UTF8 format.

The default value is False.

See Also

ReadString

WriteString

5.89.3 Methods

5.89.3.1 Connect

```
procedure Connect;
```

Description

Call **Connect** to establish a logical connection through an SSH tunnel. **Connect** sets the Connected property to True.

See Also

Connected

5.89.3.2 Disconnect

```
procedure Disconnect;
```

Description

Call **Disconnect** to close a logical connection through an SSH tunnel. **Disconnect** sets the Connected property to False.

See Also

Connected

5.89.3.3 ReadBuffer

```
function ReadBuffer(var Buffer; const Count: integer): integer; overload;
function ReadBuffer(var Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Call **ReadBuffer** to read `Count` bytes from the stream into `Buffer`. **ReadBuffer** returns bytes count that was actually read.

If size of the received data is less than `Count` bytes, **ReadBuffer** waits during amount of time specified in Timeout, and then returns control.

See Also

OnAsyncReceive

ReadNoWait

ReadString

5.89.3.4 ReadNoWait

```
function ReadNoWait(var Buffer; const Count: integer): integer; overload;
function ReadNoWait(var Buffer: TBytes; const Offset, Count: integer):
integer; overload;
function ReadNoWait(Stream: TStream): integer; overload;
```

Description

Call **ReadNoWait** to read `Count` bytes from the channel stream into `Buffer` or `Stream`. **ReadNoWait** returns bytes count that was actually read.

If the size of the received data is equal to 0, **ReadNoWait** waits for receiving at least one byte, but no longer than the amount of the time specified in Timeout, and then returns control.

If the channel contain at least one received byte, then, unlike the ReadBuffer method, it immediately returns the available number of bytes.

See Also

OnAsyncReceive

ReadBuffer

ReadString

5.89.3.5 ReadString

```
function ReadString: string;
```

Description

The **ReadString** method reads all data from the stream and returns it as a string. If the size of the received data is equal to 0, **ReadString** returns an empty string.

The UseUnicode property specifies, whether UTF8 conversion from the stream to the string is to be used.

See Also

OnAsyncReceive

ReadBuffer

ReadNoWait

UseUnicode

5.89.3.6 SkipBuffer

```
function SkipBuffer(Count: integer): integer;
```

Description

Call **SkipBuffer** to delete `Count` bytes from the stream. **SkipBuffer** returns bytes count that was actually skipped.

See Also

OnAsyncReceive

ReadBuffer

ReadNoWait

5.89.3.7 WriteBuffer

```
function WriteBuffer(const Buffer; const Count: integer): integer;
overload;
function WriteBuffer(const Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Call **WriteBuffer** to transfer `Count` bytes from `Buffer` through an existent connection. The function returns bytes count that was actually transferred.

See Also

ReadBuffer

WriteString

5.89.3.8 WriteString

```
procedure WriteString(const Buffer: string);
```

Description

Call **WriteString** to transfer the string `Buffer` through an existing connection.

The UseUnicode property specifies whether UTF8 conversion from a string to a byte array is to be used.

See Also

ReadString

WriteBuffer

UseUnicode

5.89.4 Events

5.89.4.1 OnAsyncError

```
type
  TScErrorEvent = procedure (Sender: TObject; E: Exception) of object;

property OnAsyncError: TScErrorEvent;
```

Description

Occurs when an exception is raised during asynchronous data receiving.

`Sender` is the object that raised the exception. `E` is the exception object that describes the exception.

See Also

EventsCallMode

5.89.4.2 OnAsyncReceive

```
type
  TScAsyncReceiveEvent = procedure(Sender: TObject) of object;

property OnAsyncReceive: TScAsyncReceiveEvent;
```

Description

Occurs when data is received from the server.

Note: data coming from the server is processed in a separate SSH connection stream. Therefore, the event handler call can occur in a different way to synchronize with the main thread of the application.

The EventsCallMode property determines how this event handler will be called.

The InCount property indicates the size of the received data. The data can be read with the ReadBuffer method.

See Also

EventsCallMode

InCount

5.89.4.3 OnConnect

```
property OnConnect: TNotifyEvent;
```

Description

Occurs before establishing logical connection through an SSH tunnel.

See Also

OnDisconnect

5.89.4.4 OnDisconnect

```
property OnDisconnect: TNotifyEvent;
```

Description

Occurs after the logical connection to an SSH server becomes closed.

See Also

OnConnect

5.90 TScSSHChannel

5.90.1 Description

Unit

ScSSHChannel

Description

TScSSHChannel is a component that serves as a data exchange interface for creating a logical connection through an SSH tunnel. Physically the secure connection is provided by an SSH client that can be assigned to the Client property.

When the Direct property is set to True, a connection to the specified DestHost and DestPort is established through a secure channel. To exchange data, you should use the ReadBuffer and WriteBuffer methods.

If the Direct property is not set, the component lets you forward data from one machine to another through an encrypted SSH tunnel. In this case the component is listening on the SourcePort on the local or remote host (depending on the Remote property) and forwards the received data to the specified DestHost and DestPort.

TScSSHChannel also supports dynamic port forwarding, i.e. obtaining the destination address dynamically at runtime. To enable dynamic port forwarding, set the Dynamic property to True.

See Also

Connected

TScSSHClient

Step-by-step tutorial

SSH-tunnel principles

5.90.2 Properties

5.90.2.1 Connected

```
property Connected: Boolean;
```

Description

If the Direct property is set to True, **Connected** determines whether the connection to the specified DestHost is established. Otherwise, it determines whether the port forwarding is running.

If Direct is False, and you are setting **Connected** to True, the component starts listening SourcePort on the local or remote host (on which the SSH server is located) depending on the Remote property

value. If someone is connected to this port, the logical connection to the specified DestHost and DestPort is established.

When setting **Connected** to False, all open channels are closed and the listener thread is terminated and freed.

Note: To establish a connection, it is required to set the Client property with a component that provides secure physical connection to the SSH server. After the connection is established, the information will be transferred through the secure channel.

See Also

GatewayPorts

5.90.2.2 DestHost

```
property DestHost: string;
```

Description

A host name to which the connection will be established.

See Also

DestPort

Connected

5.90.2.3 DestPort

```
property DestPort: integer;
```

Description

Use **DestPort** to specify the port number on DestHost for TCP/IP connection.

See Also

DestHost

Connected

5.90.2.4 Direct

```
property Direct: Boolean; default False;
```

Description

It **Direct** is True, the direct connection to the specified DestHost and DestPort will be created.

Otherwise the component will forward data from one machine to another through an encrypted SSH channel. In this case the component is listening the SourcePort on the local or remote host (depending on the Remote property) and forwards the received data to specified DestHost and DestPort.

See Also

Connected

DestHost

DestPort

SourcePort

5.90.2.5 Dynamic

```
property Dynamic: Boolean; default False;
```

Description

The **Dynamic** property specifies whether to use dynamic port forwarding. This option lets you forward data from one machine to another through an encrypted SSH tunnel.

When **Dynamic** is not set, the component listens on the SourcePort of the local host and forwards the received data to the specified DestHost and DestPort.

When **Dynamic** is set to True, the component listens on the SourcePort of the local host; however, the destination address is obtained dynamically at runtime. TScSSHChannel determines where to forward connections based on the SOCKS protocol. The component creates a SOCKS server which acts as a proxy that can be used in other applications.

Thus, when the **Dynamic** property is set, the client should connect to the SourcePort as to the SOCKS proxy server and specify the destination address to forward the data to during negotiation.

Note: This property has effect only when the Direct and Remote properties are set to False.

See Also

Connected

Direct

SourcePort

5.90.2.6 GatewayPorts

```
property GatewayPorts: Boolean; default False;
```

Description

Specifies whether remote hosts are allowed to connect to forwarded SourcePort. If **GatewayPorts** is False (default value), remote hosts are not allowed to connect to forwarded ports.

5.90.2.7 Remote

```
property Remote: Boolean; default False;
```

Description

Determines on which side the SourcePort will be listened when port forwarding. If this property is False, port on the localhost will be listened, if **Remote** is True - port on the remote host (on which the SSH server is located) will be listened.

Note: This property has sense only if the Direct property is set to False.

See Also

Connected

Direct

5.90.2.8 SourcePort

```
property SourcePort: integer;
```

Description

Use the **SourcePort** property to specify the port number that will be listened on the local or remote host for port forwarding.

Note: This property has sense only if the Direct property is set to False.

See Also

Connected

Direct

GatewayPorts

5.90.2.9 SSHStream

```
property SSHStream: TScSSHStream;
```

Description

Use the **SSHStream** property to obtain the TScSSHStream object that lets working with an SSH channel through the TStream interface.

5.90.3 Methods

5.90.3.1 ReadBuffer

```
function ReadBuffer(var Buffer; const Count: integer): integer; overload;
function ReadBuffer(var Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Call **ReadBuffer** to read `Count` bytes from the stream into `Buffer`. **ReadBuffer** returns bytes count that were actually read.

If size of the received data is less than `Count` bytes, **ReadBuffer** waits during amount of time specified in Timeout, and then returns control.

Note:

This method works only if the Direct property is set to True. Otherwise, an exception is raised.

See Also

OnAsyncReceive

ReadNoWait

ReadString

5.90.3.2 WriteBuffer

```
function WriteBuffer(const Buffer; const Count: integer): integer;
overload;
function WriteBuffer(const Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Call **WriteBuffer** to transfer `Count` bytes from `Buffer` through an existent connection. The function returns bytes count that was actually transferred.

Note:

This method works only if the Direct property is set to True. Otherwise, an exception is raised.

See Also

ReadBuffer

WriteString

5.90.4 Events

5.90.4.1 OnError

```
type
  TScErrorEvent = procedure(Sender: TObject; E: Exception) of object;

property OnError: TScErrorEvent;
```

Description

Occurs with local port forwarding if an error arose in the listening thread.

`Sender` is the object that raised the exception. `E` is the exception object that describes the exception.

See Also

Connected

5.90.4.2 OnSocketConnect

```
type
  TScSocketEvent = procedure(Sender: TObject; const SockAddr: TSockAddr)
of object;

property OnSocketConnect: TScSocketEvent;
```

Description

This event occurs if someone tries to connect to the SourcePort when local port forwarding.

`Sender` is the object that raised the event. `SockAddr` it the object that describes the socket for data exchange.

See Also

OnSocketDisconnect

Connected

5.90.4.3 OnSocketDisconnect

```
type
  TScSocketEvent = procedure(Sender: TObject; const SockAddr: TSockAddr)
of object;

property OnSocketDisconnect: TScSocketEvent;
```

Description

This event occurs if the socket which the connection was established with, become closed or broken.

`Sender` is the object that raised the event. `SockAddr` it the object that describes the socket for data exchange.

See Also

OnSocketConnect

Connected

5.91 TScSSHShell

5.91.1 Description

Unit

ScSSHChannel

Description

TScSSHShell is responsible for opening the shell on remote side. Usually, there is only one shell logical connection established during secure session. Physically secure connection is provided by SSH client that can be assigned to the Client property.

There are two ways to use this component. The first way is to execute a command with the ExecuteCommand method.

The second way is connection in NonBlocking mode. Set NonBlocking to True, Connect to the server, send commands to the server using the WriteString method. The OnAsyncReceive event notifies that some data from the server was received. Use the ReadString method to read it.

See Also

TScSSHClient

5.91.2 Properties

5.91.2.1 Environment

```
property Environment: TStrings;
```

Description

The **Environment** property should contain a list of environment variables in format of «Variable=Value». These variables are sent to the server on connect.

5.91.2.2 NonBlocking

```
property NonBlocking: Boolean;
```

Description

Use this property to determine what data transferring mode will be used: synchronous or asynchronous. If **NonBlocking** is True, the ReadBuffer method will not block the execution of other code in the application. The data is transferred in asynchronous mode.

When data is received from the server, the OnAsyncReceive event will arise.

See Also

InCount

OnAsyncReceive

ReadBuffer

WriteBuffer

5.91.2.3 TerminalInfo

```
property TerminalInfo: TScTerminalInfo;
```

Description

The **TerminalInfo** property represents information about pseudo-terminal, which is created on the server side for correct displaying results of the command execution via TScSSHShell.

This information is sent to the server on connect.

See also

TScTerminalInfo

5.91.3 Methods

5.91.3.1 ExecuteCommand

```
function ExecuteCommand(const Command: string): string;
```

Description

Executes `Command` on the server. **ExecuteCommand** establishes the logical connection to the SSH server and sends a command execution inquiry.

If NonBlocking mode is not enabled, the method waits until the command is executed, and then returns a result. After the result is obtained, connection to the server is closed. So, this method can execute only one command within a connection.

In NonBlocking mode the method immediately returns an empty string, and does not wait until the command is executed. The command execution result can be obtained with the ReadString or ReadBuffer method.

The result of the **ExecuteCommand** method is tightly related to the SSH server that executes the command.

An alternative way to execute commands remotely is calling the WriteString and WriteBuffer methods.

See Also

NonBlocking

ReadString

WriteString

5.91.3.2 ReadString

```
function ReadString: string;
```

Description

The **ReadString** method reads the result of a command executed by the WriteString method.

See Also

ReadBuffer

WriteString

5.91.3.3 WriteString

```
procedure WriteString(const Buffer: string);
```

Description

Use the **WriteString** method to send a command with parameters to the server. The command is passed through an existent connection and executed remotely. The line feed symbol must conclude the command.

The result of the command execution can be obtained by the ReadString and ReadBuffer methods.

See Also

ReadString

WriteBuffer

5.92 TScSSHStream

5.92.1 Description

Unit

ScSSHChannel

Description

The **TScSSHStream** class is a descendant of TStream and lets read and write data through the protected channel. Use **TScSSHStream** to get access to an SSH channel through the TStream interface.

See Also

TScSSHChannel.SSHStream

5.92.2 Methods

5.92.2.1 Create

```
constructor Create(Channel: TScSSHChannel);
```

Description

Create **TScSSHStream** instance.

The `Channel` parameter is an object that represents the protected channel to reading and writing data through it. The Channel.Direct property must be set to True. Otherwise, an exception is raised.

See Also

TScSSHChannel

5.93 TScSSHClient

5.93.1 Description

Unit

ScSSHClient

Description

TScSSHClient is a component that implements functionality of SSH client. **TScSSHClient** unites several logical server connections in one physical secure connection. Logical connections can exist in different threads. It connects to the SSH server to which point the HostName and Port properties.

To connect to an SSH server, you can use the following parameters:

- authentication method Authentication that will be used by the server to authenticate the client.
- asymmetric encrypting algorithms HostKeyAlgorithms and server public key HostKeyName are used by the client to authenticate for the SSH server;
- symmetric encrypting algorithms CiphersClient and CiphersServer to encrypt transferred data;
- information about user: User, Password, PrivateKeyName.

See Also

Connected

Step-by-step tutorial

SSH-tunnel destination

5.93.2 Properties

5.93.2.1 Authentication

```
property Authentication: TScSSHAuthentication; default atPassword;
```

Description

The **Authentication** property determines what authentication method will be used by server to authenticate the client.

The default is the authentication by password.

See Also

Connected

TScSSHClient.OnAuthenticationPrompt

5.93.2.2 CiphersClient

```
property CiphersClient: TScSSHCiphers;
```

Description

The **CiphersClient** property holds a list of the acceptable symmetric algorithms that can be used for encrypting data that is passed from client to server.

The algorithms are stored in order of preference.

See Also

Connected

5.93.2.3 CiphersServer

```
property CiphersServer: TScSSHCiphers;
```

Description

The **CiphersServer** property holds a list of the acceptable symmetric algorithms that can be used for encrypting data that is passed from server to client.

The algorithms are stored in order of preference.

See Also

Connected

5.93.2.4 ClientInfo

```
property ClientInfo: TScSSHClientInfo;
```

Description

Holds information about the current connection. **ClientInfo** is initialized after the client is authenticated by server.

See Also

Connected

5.93.2.5 CompressionClient

```
property CompressionClient: TScCompression; default csAllowed;
```

Description

The **CompressionClient** property indicates how data compression should be used for data that is passed from client to server.

Compression is allowed by default.

See Also

Connected

5.93.2.6 CompressionServer

```
property CompressionServer: TScCompression; default csAllowed;
```

Description

The **CompressionServer** property indicates how data compression should be used for data that is passed from server to client.

Compression is allowed by default.

See Also

Connected

5.93.2.7 Connected

```
property Connected: Boolean;
```

Description

Determines whether the connection to SSH server is established. Switch **Connected** to True, to establish connection to SSH server. Switch **Connected** to False, to close the connection to SSH server.

See Also

Connect

Disconnect

5.93.2.8 HMACAlgorithms

```
property HMACAlgorithms: TScSSHHMacAlgorithms;
```

Description

The **HMACAlgorithms** property holds a list of the acceptable HMAC algorithms, which can be used during the SSH handshake.

The algorithms are stored in order of preference.

See Also

HostKeyName

Connected

5.93.2.9 HostKeyAlgorithms

```
property HostKeyAlgorithms: TScSSHHostKeyAlgorithms;
```

Description

The **HostKeyAlgorithms** property holds the list of the algorithms supported for the server host key.

Specify the asymmetric algorithms, for what the client have a server public key, or want to obtain this key. This key used by client to authenticate the server.

The algorithms are stored in order of preference.

See Also

HostKeyName

Connected

5.93.2.10 HostKeyName

```
property HostKeyName: string;
```

Description

Determines name of the server public key that is stored in KeyStorage.

The public key received from the server is compared with the key from KeyStorage when server is authenticating. If the keys does not coincide, or the corresponding key is not found in the

KeyStorage, the OnServerKeyValidate event is raised. If the keys coincide, the server is considered valid.

Note: If **HostKeyName** is not specified, the key is searched by HostName.

See Also

OnServerKeyValidate

HostKeyAlgorithms

5.93.2.11 HostName

```
property HostName: string;
```

Description

Specifies the host name or the IP addres to connect to the SSH server.

The Connect method uses values in the **HostName** and Port properties to establish a connection for the SSH session.

See Also

Port

Connected

5.93.2.12 HttpOptions

```
property HttpOptions: THttpOptions;
```

Description

The **HttpOptions** property holds a THttpOptions object that contains settings for HTTP connection.

For more information on HTTP tunneling refer to the Network Tunneling article.

See Also

Connected

5.93.2.13 KeyExchangeAlgorithms

```
property KeyExchangeAlgorithms: TScSSHKeyExchangeAlgorithms;
```

Description

The **KeyExchangeAlgorithms** property holds a list of the acceptable key exchange algorithms, which can be used during the SSH handshake.

The algorithms are stored in order of preference.

See Also

HostKeyName

Connected

5.93.2.14 KeyStorage

```
property KeyStorage: TScStorage;
```

Description

KeyStorage is used to access key list in storage. If **KeyStorage** is not assigned, an exception will be raised when attempting to connect.

See Also

HostKeyName

PrivateKeyName

5.93.2.15 Options

```
property Options: TScSSHClientOptions;
```

Description

Options determines behaviour of the SSH client.

See Also

TScSSHClientOptions

Connected

5.93.2.16 Password

```
property Password: string;
```

Description

Password is used to connect to the server when user is authenticated by password.

See Also

Authentication

Connected

5.93.2.17 Port

```
property Port: integer; default 22;
```

Description

Use the **Port** property to specify a port number for TCP/IP connection with the SSH server.

The Connect method uses values in the HostName and **Port** properties to establish a connection for the SSH session.

The default value is 22 port number.

See Also

HostName

Connected

5.93.2.18 PrivateKeyName

```
property PrivateKeyName: string;
```

Description

Specifies private key name that is stored in KeyStorage.

If the authentication by key is used, the user must have his pair of keys. The public key should be transferred to the server, while the private key will be used by the client to sign data, that will be used by server to authenticate the user.

Note: If **PrivateKeyName** is not specified, the key will be searched by the name in User.

See Also

Authentication

Keys transferring

5.93.2.19 ServerVersion

```
property ServerVersion: string;
```

Description

Holds the version of the current SSH server. **ServerVersion** is initialized after the client is authenticated by the server.

5.93.2.20 Timeout

```
property Timeout: integer; default 15;
```

Description

Determines the time interval in seconds during which the client will try to obtain data from the server when authenticating. If data is not obtained, the connection becomes closed.

The default value is 15 seconds.

See Also

Connected

5.93.2.21 User

```
property User: string;
```

Description

User name that is used to connect to the server.

See Also

Password

Connected

5.93.3 Methods

5.93.3.1 Connect

```
procedure Connect;
```

Description

Establishes connection to SSH server. **Connect** sets the Connected property to True.

See Also

Disconnect
AfterConnect
BeforeConnect

5.93.3.2 Disconnect

```
procedure Disconnect;
```

Description

Closes an existent connection to SSH server. **Disconnect** sets the Connected property to False.

See Also

Connect
AfterDisconnect
BeforeDisconnect

5.93.4 Events

5.93.4.1 AfterConnect

```
property AfterConnect: TNotifyEvent;
```

Description

Occurs after a connection to an SSH server is established.

See Also

AfterDisconnect
BeforeConnect
BeforeDisconnect
Connected

5.93.4.2 AfterDisconnect

```
property AfterDisconnect: TNotifyEvent;
```

Description

Occurs after the connection to an SSH server becomes closed.

See Also

AfterConnect
BeforeConnect
BeforeDisconnect
Connected

5.93.4.3 BeforeConnect

```
property BeforeConnect: TNotifyEvent;
```

Description

Occurs immediately before establishing a connection to an SSH server.

See Also

AfterConnect
AfterDisconnect
BeforeDisconnect
Connected

5.93.4.4 BeforeDisconnect

```
property BeforeDisconnect: TNotifyEvent;
```

Description

Occurs immediately before the connection to an SSH server becomes closed.

See Also

AfterConnect
AfterDisconnect
BeforeConnect
Connected

5.93.4.5 OnBanner

```
type
  TBannerEvent = procedure(Sender: TObject; const Banner: string) of
```

```
object;

property OnBanner: TBannerEvent;
```

Description

Occurs if SSH server returns a banner when authenticating. The `Banner` hold received banner.

The banner may contain a warning message or any other information message.

See Also

Connected

5.93.4.6 OnAuthenticationPrompt

```
type
  TScAuthenticationPromptEvent = procedure(Sender: TObject; const Name,
Instruction: string; const Prompts: TStringDynArray; var Responses:
TStringDynArray) of object;

property OnAuthenticationPrompt: TScAuthenticationPromptEvent;
```

Description

The **OnAuthenticationPrompt** event occurs when performing keyboard-interactive authentication. This event may be called several times during authentication process to request corresponding user information.

The server sends a request concerning information that should be obtained from the user. The amount of requested information can be learned by defining the length of the `Prompts` array. Developer should provide an interface for the user to enter requested information. The received information should be written to the `Responses` array.

Parameters:

- `Sender` - the object that raised the event;
- `Name` - the query name;
- `Instruction` - extra data for explanation;
- `Prompts` - an array of strings, each one of which holds a request to user concerning necessary information. The length of the array can be 0;
- `Responses` - an array of strings a user should fill in with the corresponding information.

See Also

TScSSHClient.Authentication

5.93.4.7 OnServerKeyValidate

```
type
  TScServerKeyValidationEvent = procedure(Sender: TObject; NewServerKey:
TScKey; var Accept: boolean) of object;

property OnServerKeyValidate: TScServerKeyValidationEvent;
```

Description

Occurs if the key received from the server and the key specified in HostKeyName does not coincide.

If the client connects to the server for the first time and does not have the server public key, it is possible to accept the key received from the server. This key will be stored in Storage. It will be used to authenticate the server in the future. But in this case to provide safety, you ought to verify in any way (e.g. by phone) the key print. If you trust the server, set the `Accept` to True to establish the connection.

To get the key print, use the GetFingerprint method.

To save a key to the Storage, specify the key name (NewServerKey.KeyName) and invoke KeyStorage.Keys.Add(NewServerKey).

Parameters:

- `Sender` - the object that raised the event;
- `NewServerKey` - the public key received from the server;
- `Accept` - when `Accept` is set to True, the server is considered valid, and the server authentication is successful. When `Accept` is set to False, the server is considered invalid and the connection is closed.

See Also

HostKeyName

Connected

5.94 TScSSHClientOptions

5.94.1 Description

Unit

ScSSHClient

Description

The **TScSSHClientOptions** class determines behaviour of an SSH client.

See also

TScSSHClient.Options

5.94.2 Properties

5.94.2.1 BindAddress

```
property BindAddress: string;
```

Description

Determines the TCP/IP address on the local machine as the source address of the connection. Only useful on systems with more than one TCP/IP address.

5.94.2.2 ClientVersion

```
property ClientVersion: string;
```

Description

Determines the version of TScSSHClient. The default value is `'SSH-2.0-Devart-8.0'`.

5.94.2.3 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is `ivIPv4`.

See also

TIPVersion

5.94.2.4 MsgIgnoreRate

```
property MsgIgnoreRate: Integer; default 0;
```

Description

Determines probability of sending a packet that will be ignore by the server (SSH_MSG_IGNORE packets) after each data packet. These ignore packets are intended for increasing data protection level against cracking by traffic analyzing. The value of the **MsgIgnoreRate** property can vary from 0 to 100. 0 means that no ignore packages will be sent. 100 means that one ignore package will be sent after the each data package.

Note: The traffic is increased when you increase the value of this option.

5.94.2.5 RekeyLimit

```
property RekeyLimit: string;
```

Description

The **RekeyLimit** property determines how much data can be transferred before the session key is renegotiated. You can specify a number with a prefix that indicates unit (K - Kilobytes, M - Megabytes, G - Gigabytes).

5.94.2.6 ServerAliveCountMax

```
property ServerAliveCountMax: integer; default 3;
```

Description

Determines how many keep-alive messages may be sent to server before a response from the server is received. If the value of this property is reached, SSH client will disconnect from the server.

The default value is 3.

See also

ServerAliveInterval

5.94.2.7 ServerAliveInterval

```
property ServerAliveInterval: integer; default 0;
```

Description

Determines a timeout interval in seconds after which SSH client will send a keep-alive message through the encrypted channel to request a response from the server if no data has been received from the server.

The default value is 0. It means that these messages will not be sent to the server.

See also

ServerAliveCountMax

5.94.2.8 SocketReceiveBufferSize

```
property SocketReceiveBufferSize: integer; default 32768;
```

Description

Use the **SocketReceiveBufferSize** property to determine the total per-socket buffer space reserved for receives. This value is set by the OS functions to the socket.

Use this property to increase the application performance.

The default value is 32768.

See also

SocketSendBufferSize

5.94.2.9 SocketSendBufferSize

```
property SocketSendBufferSize: integer; default 32768;
```

Description

Use the **SocketSendBufferSize** property to determine the total per-socket buffer space reserved for sends.This value is set by the OS functions to the socket.

Use this property to increase the application performance.

The default value is 32768.

See also

SocketReceiveBufferSize

5.94.2.10 TCPKeepAlive

```
property TCPKeepAlive: boolean; default True;
```

Description

The **TCPKeepAlive** property specifies whether the system should send TCP keep alive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed.

The default value is True.

5.95 TScSSHConnectionInfo

5.95.1 Description

Unit

ScSSHUtils

Description

The **TScSSHConnectionInfo** class holds information about an SSH connection.

See also

TScSSHClientInfo

TScSSHClient.ClientInfo

5.95.2 Properties

5.95.2.1 CipherClient

```
property CipherClient: TScSymmetricAlgorithm;
```

Description

The **CipherClient** property represents the symmetric algorithm used in the current connection to encrypt data transferred from the client to the server.

This property is read-only.

5.95.2.2 CiphersClient

```
property CiphersClient: TScSSHCiphers;
```

Description

The **CiphersClient** property holds list of acceptable symmetric algorithms for encryption data passed from the client to the server. The algorithms are stored in order of preference.

This property is read-only.

5.95.2.3 CipherServer

```
property CipherServer: TScSymmetricAlgorithm;
```

Description

The **CipherServer** property represents the symmetric algorithm used in the current connection to encrypt data transferred from the server to the client.

This property is read-only.

5.95.2.4 CiphersServer

```
property CiphersServer: TScSSHCiphers;
```

Description

The **CiphersServer** property holds list of acceptable symmetric algorithms for encryption data passed from the server to the client. The algorithms are stored in order of preference.

This property is read-only.

5.95.2.5 CompressionClient

```
property CompressionClient: TScCompressionAlgorithm;
```

Description

The **CompressionClient** property represents the algorithm used in the current connection to compress data transferred from the client to the server.

This property is read-only.

5.95.2.6 CompressionServer

```
property CompressionServer: TScCompressionAlgorithm;
```

Description

The **CompressionServer** property represents the algorithm used in the current connection to compress data transferred from the server to the client.

This property is read-only.

5.95.2.7 Domain

```
property Domain: string;
```

Description

The **Domain** property holds the domain in which the user, initiated the connection, is placed and that is used for OS authentication.

This property is read-only.

5.95.2.8 HMACAlgorithms

```
property HMACAlgorithms: TScSSHHMacAlgorithms;
```

Description

The **HMACAlgorithms** property holds list of acceptable HMAC algorithms used in the current connection for verifying integrity of data that is transferred between an SSH client and an SSH server. The algorithms are stored in order of preference.

This property is read-only.

5.95.2.9 HMACClient

```
property HMACClient: TScHMACAlgorithm;
```

Description

The **HMACClient** property holds the HMAC algorithm used in the current connection for verifying data integrity that is transferred from the client to the server.

This property is read-only.

5.95.2.10 HMACServer

```
property HMACServer: TScHMACAlgorithm;
```

Description

The **HMACServer** property holds the HMAC algorithm used in the current connection for verifying data integrity that is transferred from the server to the client.

This property is read-only.

5.95.2.11 HostKeyAlgorithm

```
property HostKeyAlgorithm: TScAsymmetricAlgorithm;
```

Description

The **HostKeyAlgorithm** property represents the asymmetric encryption algorithm for the server host key, used in the current connection.

This property is read-only.

5.95.2.12 KeyExchangeAlgorithm

```
property KeyExchangeAlgorithm: TScKeyExchangeAlgorithm;
```

Description

The **KeyExchangeAlgorithm** property represents the key exchange algorithm which was accepted during SSH handshake of the current connection.

This property is read-only.

5.95.2.13 LocalSockAddr

```
property LocalSockAddr: PSockAddr;
```

Description

The **LocalSockAddr** property represents the information in the WinSocket format about the local socket used for data exchange.

This property is read-only.

5.95.2.14 SockAddr

```
property SockAddr: PSockAddr;
```

Description

The **SockAddr** property represents the information in the WinSocket format about the socket used for data exchange.

This property is read-only.

5.95.2.15 User

```
property User: string;
```

Description

The **User** property represents the user name initiated the connection.

This property is read-only.

5.95.2.16 UserExtData

```
property UserExtData: string;
```

Description

UserExtData has no predefined meaning. The **UserExtData** property is provided for the convenience of developers. It can be used for storing an additional information.

This property is read-only.

5.95.2.17 Version

```
property Version: string;
```

Description

The **Version** property represents the version of the another side (the server version for the client, the client version for the server).

This property is read-only.

5.96 TScSSHClientInfo

5.96.1 Description

Unit

ScSSHUtils

Description

The **TScSSHClientInfo** class is a descendant of the TScSSHConnectionInfo class, that holds information about an SSH connection, and it adds the Data property for the convenience of developers.

See also

TScSSHClient.ClientInfo

TScSSHServer.ClientInfos

5.96.2 Properties

5.96.2.1 Data

```
property Data: TObject;
```

Description

Data has no predefined meaning. The **Data** property is provided for the convenience of developers. It can be used for storing an additional object.

5.97 TScSSHChannelInfo

5.97.1 Description

Unit

ScSSHUtils

Description

The **TScSSHChannelInfo** contains the information about an SSH channel.

See also

TScSSHCustomChannel.ChannelInfo

TScSSHServer.ChannelInfos

TScSSHClientInfo

5.97.2 Properties

5.97.2.1 Client

```
property Client: TScSSHClientInfo;
```

Description

The **Client** property holds information about the current SSH connection.

This property is read-only.

5.97.2.2 Data

```
property Data: TObject;
```

Description

Data has no predefined meaning. The **Data** property is provided for the convenience of developers. It can be used for storing an additional object.

5.97.2.3 DestHost

```
property DestHost: string;
```

Description

The **DestHost** property represents the name of the host, which is established connection to.

This property is read-only.

See also

DestPort

5.97.2.4 DestPort

```
property DestPort: integer;
```

Description

The **DestPort** property represents the port number at DestHost for TCP/IP connection.

This property is read-only.

See also

DestHost

5.97.2.5 Direct

```
property Direct: boolean;
```

Description

The **Direct** property determines in what way data received from this client will be handled at the server. If **Direct** is True, data received from the SSH client is passed through a socket to the host specified in DestHost.

If **Direct** is False, the information received from the SSH client is not passed ahead automatically. In this case the input information must be handled in a handler of the OnDataFromClient event.

This property is read-only.

5.97.2.6 IsSession

```
property IsSession: boolean;
```

Description

The **IsSession** property determines the type of the current SSH channel. If **IsSession** is True, channel is the shell session. If **IsSession** is False, the information received from an SSH client is passed ahead.

This property is read-only.

5.97.2.7 Remote

```
property Remote: boolean;
```

Description

The **Remote** property determines from which side of the tunnel is the connection initiator located. If **Remote** is True, the initiator is on the side of an SSH server, if **Remote** is False, the initiator is on the side of an SSH client.

This property is read-only.

5.98 TScSSHServerOptions

5.98.1 Description

Unit

ScSSHServer

Description

The **TScSSHServerOptions** class determines behaviour of an SSH server.

See also

TScSSHServer.Options

5.98.2 Properties

5.98.2.1 AllowEmptyPassword

```
property AllowEmptyPassword: boolean; default False;
```

Description

Determines whether connection with an empty user's password is allowed.

5.98.2.2 Banner

```
property Banner: string;
```

Description

Holds a warning message that is sent to a client before authentication is allowed.

5.98.2.3 ClientAliveCountMax

```
property ClientAliveCountMax: integer; default 3;
```

Description

Determines how many keep-alive messages may be sent to a client before a response from the client is received. If the value of this property is reached, SSH server will close connection with this client.

The default value is 3.

5.98.2.4 ClientAliveInterval

```
property ClientAliveInterval: integer; default 0;
```

Description

Determines a timeout interval in seconds after which SSH server will send a keep-alive message through the encrypted channel to request a response from the client if no data has been received from the client.

The default value is 0. It means that these messages will not be sent to the client.

5.98.2.5 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is `ivIPv4`.

See also

TIPVersion

5.98.2.6 ListenAddress

```
property ListenAddress: string;
```

Description

Specifies the local address which SSH server should listen to. If there are several netcards installed on the computer, and **ListenAddress** is not assigned, the "0.0.0.0" address will be listened. It means

that it is possible to connect to any of the installed netcards.

5.98.2.7 ListenBacklog

```
property ListenBacklog: integer; default 5;
```

Description

Specifies the maximum number of queued connection requests that can be pending.

ListenBacklog is a socket-level property that describes the number of "pending accept" requests to be queued. If the listen backlog queue fills up, new socket requests will be rejected.

The default value is 5.

5.98.2.8 MaxConnections

```
property MaxConnections: integer; default 0;
```

Description

Specifies the maximum number of the established connections to the SSH server. Additional connections will be dropped until any of the opened connections is not completed.

0 value means an unlimited number of the opened connections.

The default value is 0.

5.98.2.9 MaxStartups

```
property MaxStartups: integer; default 20;
```

Description

Specifies the maximum number of concurrent unauthenticated connections to the SSH server. Additional connections will be dropped until authentication succeeds.

The default value is 20.

5.98.2.10 RekeyLimit

```
property RekeyLimit: string;
```

Description

The **RekeyLimit** property determines how much data can be transferred before the session key is renegotiated. You can specify a number with a prefix that indicates unit (K - Kilobytes, M - Megabytes, G - Gigabytes).

5.98.2.11 ServerVersion

```
property ServerVersion: string;
```

Description

Determines the version of TScSSHServer. The default value is `'SSH-2.0-Devart-8.0'`.

5.98.2.12 TCPKeepAlive

```
property TCPKeepAlive: boolean; default True;
```

Description

The **TCPKeepAlive** property specifies whether the system should send TCP keep alive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed.

The default value is True.

5.99 TScSSHServer

5.99.1 Description

Unit

ScSSHServer

Description

The **TScSSHServer** component implements functionality of SSH server.

TScSSHServer listens to the TCP/IP port specified in the Port property, and if an SSH client tries to connect to this port, **TScSSHServer** authenticates the client. If authentication is successful, it will establish connection. After that the **TScSSHServer** carries out client queries.

Storage.Users holds the list of the users that are allowed to connect to the SSH server.

To support for the SFTP protocol set the SFTPServer property to reference of a TScSFTPServer object.

See Also

Active

TScSFTPServer

TScSSHClient

Step-by-step tutorial

5.99.2 Properties

5.99.2.1 Active

```
property Active: Boolean;
```

Description

Indicates whether the SSH server is running.

Set **Active** to True to run the SSH server. After the server is activated, it starts listening the TCP/IP specified in the Port property.

Set **Active** to False to stop the SSH server.

See Also

BeforeClientConnect

Port

5.99.2.2 AllowCompression

```
property AllowCompression: boolean; default True;
```

Description

The **AllowCompression** property indicates if data, that is passed between the SSH server and an SSH client, can be compressed.

The default value is True.

5.99.2.3 Authentications

```
property Authentications: TScSSHAuthentications; default [atPublicKey,
atPassword];
```

Description

The **Authentications** property holds a set of acceptable authentication methods, that can be used by the SSH server to authenticate a client.

TScSSHServer supports only the authentication by the user's public key and the authentication by password.

5.99.2.4 ChannelInfoCount

```
property ChannelInfoCount: Integer;
```

Description

The **ChannelInfoCount** property represents a quantity of opened at this moment SSH-channels. The information about channels can be accessed via the ChannelInfos property.

See also

ChannelInfos

5.99.2.5 ChannelInfos

```
property ChannelInfos[Index: Integer]: TScSSHChannelInfo;
```

Description

The **ChannelInfos** property contains information about logical connections - SSH-channels, which are opened at this moment on the server.

The quantity of channels can be found via the ChannelInfoCount property.

See also

ChannelInfoCount

ClientInfos

TScSSHChannelInfo

5.99.2.6 Ciphers

```
property Ciphers: TScSymmetricAlgorithms; default [saTripleDES_cbc,
         saBlowfish_cbc, saAES128_cbc, saAES192_cbc, saAES256_cbc,
         saCast128_cbc, saTripleDES_ctr, saBlowfish_ctr, saAES128_ctr,
         saAES192_ctr, saAES256_ctr, saCast128_ctr];
```

Description

The **Ciphers** property holds a set of the acceptable symmetric encryption algorithms, that are used for encrypting transferred data.

The default value is list of following algorithms:

saTripleDES_cbc, saBlowfish_cbc, saAES128_cbc, saAES192_cbc, saAES256_cbc, saCast128_cbc, saTripleDES_ctr, saBlowfish_ctr, saAES128_ctr, saAES192_ctr, saAES256_ctr, saCast128_ctr.

5.99.2.7 ClientInfoCount

```
property ClientInfoCount: Integer;
```

Description

The **ClientInfoCount** property represents a quantity of clients, which are connected to the server at this moment.

The information about clients can be accessed by the ClientInfos property.

See also

ClientInfos

5.99.2.8 ClientInfos

```
property ClientInfos[Index: Integer]: TScSSHClientInfo;
```

Description

The **ClientInfos** property represents information about clients, which are connected to the server at this time. The information about client is added to the list after successful connecting. It is deleted after connection closing.

The quantity of the clients can be found via the ClientInfoCount property.

See also

ChannelInfos

ClientInfoCount

TScSSHClientInfo

5.99.2.9 HMACs

```
property HMACs: TScHMACAlgorithms; default [hmacSHA1, hmacSHA2_256,
hmacSHA2_512, hmacSHA2_224, hmacSHA2_384];
```

Description

The **HMACs** property holds a set of the acceptable HMAC algorithms that are used during the SSH handshake.

The default value is list of following algorithms:

hmacSHA1, hmacSHA2_256, hmacSHA2_512, hmacSHA2_224, hmacSHA2_384;

See Also

KeyExchangeAlgorithms

5.99.2.10 HostKeyAlgorithms

```
property HostKeyAlgorithms: TScAsymmetricAlgorithms; default [aaRSA,
aaEC];
```

Description

The **HostKeyAlgorithms** property holds set of the algorithms supported for the host key.

Indicate the asymmetric algorithms, for which server has private key that. This key is used by client to authenticate the server.

The default value is the RSA algorithm.

See Also

KeyNameRSA

KeyNameDSA

5.99.2.11 KeyExchangeAlgorithms

```
property KeyExchangeAlgorithms: TScKeyExchangeAlgorithms; default
        [keDHGroup1Sha1, keDHGroup14Sha1, keDHExchSha1, keDHExchSha256,
        keECDHSha2Nistp256, keECDHSha2Nistp384, keECDHSha2Nistp521,
        keCurve25519Sha256];
```

Description

The **KeyExchangeAlgorithms** property holds a set of the acceptable key exchange algorithms that are used during the SSH handshake.

The default value is list of following algorithms:

keDHGroup1Sha1, keDHGroup14Sha1, keDHExchSha1, keDHExchSha256, keECDHSha2Nistp256, keECDHSha2Nistp384, keECDHSha2Nistp521, keCurve25519Sha256.

See Also

HMACs

5.99.2.12 KeyNameDSA

```
property KeyNameDSA: string;
```

Description

Determines name of the private DSA key that is stored in Storage.

The server must have one or more couples of keys so that clients are able to authenticate the server. The public key should be passed to the client. The private key will be used by the server when authenticating.

Note: If **KeyNameDSA** is not specified, the key is searched by the name 'ssh-dss'.

See Also

KeyNameRSA

HostKeyAlgorithms

TScSSHClient.HostKeyName

Keys transferring

5.99.2.13 KeyNameRSA

```
property KeyNameRSA: string;
```

Description

Determines name of the private RSA key that is stored in Storage.

The server must have one or more couples of keys so that clients are able to authenticate the server. The public key should be passed to the client. The private key will be used by the server when authenticating.

Note: If **KeyNameRSA** is not specified, the key is searched by the name 'ssh-rsa'.

See Also

KeyNameDSA

HostKeyAlgorithms

TScSSHClient.HostKeyName

Keys transferring

5.99.2.14 Options

```
property Options: TScSSHServerOptions;
```

Description

Options determines behaviour of the SSH server.

See Also

Active

TScSSHServerOptions

5.99.2.15 Port

```
property Port: integer; default 22;
```

Description

Use the **Port** property to specify what TCP/IP port number will the SSH server listen on.

The default value is 22 port number.

See Also

Active

BeforeClientConnect

5.99.2.16 ServerVersion

```
property ServerVersion: string;
```

Description

The **ServerVersion** property specifies the version of the SSH server.

This property is read-only. The default value is `'SSH-2.0-Devart-7.0'`.

See Also

Active

5.99.2.17 SFTPServer

```
property SFTPServer: TScSFTPServer;
```

Description

Use the **SFTPServer** property to specify support for the SFTP protocol.

To support for the SFTP protocol, **SFTPServer** must be set. This property can be set at design time by selecting a TScSFTPServer object from the provided list. At runtime, set the **SFTPServer** property to reference of an existent TScSFTPServer object.

See Also

Active

TScSFTPServer

5.99.2.18 Storage

```
property Storage: TScStorage;
```

Description

Use the **Storage** property to store keys and user list in storage.

The **Storage**.Users holds the user list that can connect to the server.

5.99.2.19 Timeout

```
property Timeout: integer; default 60;
```

Description

Determines time interval in seconds during which the server will be trying to obtain data from the client when authenticating. If the data is not received, server closes this connection.

The default value is 60 seconds.

5.99.3 Methods

5.99.3.1 SendToClient

```
procedure SendToClient(ChannelInfo: TScSSHChannelInfo; const Buffer;
const Count: integer); overload;
procedure SendToClient(ChannelInfo: TScSSHChannelInfo; const Buffer:
TBytes; const Offset, Count: integer); overload;
```

Description

Call **SendToClient** to send data to an SSH client.

Use this method if for the channel specified in `ChannelInfo` the Direct mode is used. To handle data received from the client, the OnDataFromClient and OnDataToClient events are used.

Parameters:

- `ChannelInfo` - holds the information about SSH channel;

- `Buffer` - points to the buffer that contains data to be transferred;
- `Offset` - zero-based byte offset in `Buffer` that indicates location of the data to transfer;
- `Count` - length of the data to be transferred.

See also

TScSSHChannelInfo.Direct

OnDataFromClient

OnDataToClient

5.99.4 Events

5.99.4.1 AfterChannelDisconnect

```
type
  TScAfterChannelDisconnect = procedure(Sender: TObject; ChannelInfo:
    TScSSHChannelInfo) of object;

property AfterChannelDisconnect: TScAfterChannelDisconnect;
```

Description

Occurs after an SSH channel is disconnected.

Parameters:

- `Sender` - the object whose event handler is called;
- `ChannelInfo` - holds the information about the current SSH channel.

5.99.4.2 AfterClientConnect

```
type
  TScClientEvent = procedure(Sender: TObject; ClientInfo:
    TScSSHClientInfo) of object;

property AfterClientConnect: TScClientEvent;
```

Description

This event occurs after the connection with an SSH client is established.

Parameters:

- `Sender` - the SSH server to which the client connects;

- `ClientInfo` - holds the information about the current state.

See Also

BeforeClientConnect

5.99.4.3 AfterClientDisconnect

type

```
  TScClientEvent = procedure(Sender: TObject; ClientInfo:
    TScSSHClientInfo) of object;

property AfterClientDisconnect: TScClientEvent;
```

Description

Occurs after an SSH client is disconnected, or if connection to the client is lost.

Parameters:

- `Sender` - the object whose event handler is called;
- `ClientInfo` - holds the information about the current state.

5.99.4.4 AfterShellDisconnect

type

```
  TScAfterShellDisconnect = procedure(Sender: TObject; ClientInfo:
TScSSHClientInfo) of object;

property AfterShellDisconnect: TScAfterShellDisconnect;
```

Description

Occurs after an SSH shell session is disconnected.

Parameters:

- `Sender` - the object whose event handler is called;
- `ClientInfo` - holds information about the current connection state.

See Also

TScSSHShell

5.99.4.5 BeforeChannelConnect

```
type
  TScBeforeChannelConnect = procedure(Sender: TObject; ChannelInfo:
    TScSSHChannelInfo; var Direct: Boolean) of object;

property BeforeChannelConnect: TScBeforeChannelConnect;
```

Description

Occurs on opening a new SSH channel.

If you set the `Direct` parameter to True, the data obtained from the client will not be transferred anywhere. It is required to add a handler for the OnDataFromClient event to handle the data obtained from client.

Parameters:

- `Sender` - the object whose event handler is called;
- `ChannelInfo` - holds the information about the current SSH channel;
- `Direct` - determines whether the data obtained from the client will be transferred to the host and port specified by user. Set `Direct` to True if you want to process the data received from the client yourself.

See Also

OnDataFromClient

OnDataToClient

5.99.4.6 BeforeClientConnect

```
type
  TScBeforeClientConnectEvent = procedure(Sender: TObject; const
    SockAddr: PSockAddr; var Cancel: boolean) of object;

property BeforeClientConnect: TScBeforeClientConnectEvent;
```

Description

Occurs before establishing a new SSH connection, when there is a socket connection to the port listened by the SSH server.

To cancel a connection, set the `Cancel` parameter to True. This can be done for both a particular IP by checking the `SockAddr` parameter and for any other reason.

Parameters:

- `Sender` - the object whose event handler is called;
- `SockAddr` - holds the information in the WinSocket format about the socket that is trying to connect to the SSH server;
- `Cancel` - determines whether an SSH connection to the specified socket will be established. Set `Cancel` to True if you want to cancel establishing a connection.

See Also

AfterClientConnect

5.99.4.7 BeforeShellConnect

```
type
  TScBeforeShellConnect = procedure(Sender: TObject; ClientInfo:
    TScSSHClientInfo) of object;

property BeforeShellConnect: TScBeforeShellConnect;
```

Description

Occurs before opening a new shell session.

Parameters:

- `Sender` - the object whose event handler is called;
- `ClientInfo` - holds information about the current connection state.

See Also

TScSSHShell

5.99.4.8 OnCancelRemotePortForwardingRequest

```
type
  TScOnCancelRemotePortForwardingRequest = procedure(Sender: TObject;
 ClientInfo: TScSSHClientInfo; const Host: string; const Port: Integer) of
 object;

property OnCancelRemotePortForwardingRequest:
TScOnCancelRemotePortForwardingRequest;
```

Description

The **OnCancelRemotePortForwardingRequest** event occurs when an SSH client side requests to

cancel remote port forwarding that was started earlier.

Parameters:

- `Sender` - the object whose event handler is called;
- `ClientInfo` - holds information about the current connection;
- `Host` - points to the host from which data is forwarded;
- `Port` - specifies the number of the port that is listened to for data forwarding.

See Also

OnRemotePortForwardingRequest

5.99.4.9 OnChannelError

```
type
  TScChannelError = procedure(Sender: TObject; ChannelInfo:
    TScSSHChannelInfo; E: Exception) of object;

property OnChannelError: TScChannelError;
```

Description

The **OnChannelError** event occurs on errors that arise in an SSH channel thread.

The event handler is called in the thread in which the Exception arose.

Parameters:

- `Sender` - the object whose event handler is called;
- `ChannelInfo` - holds the information about the current SSH channel;
- `E` - the object that describes the exception.

5.99.4.10 OnClientError

```
type
  TScClientError = procedure(Sender: TObject; ClientInfo:
    TScSSHClientInfo; E: Exception) of object;

property OnClientError: TScClientError;
```

Description

The **OnClientError** event occurs on errors that arise in an SSH client thread.

The event handler is called in the thread in which the Exception arose.

Parameters:

- `Sender` - the object whose event handler is called;
- `ClientInfo` - holds the information about the current connection;
- `E` - the object that describes the exception.

5.99.4.11 OnDataFromClient

```
type
  TScData = procedure(Sender: TObject; ChannelInfo: TScSSHChannelInfo;
    const Buffer: TBytes; const Offset, Count: integer) of object;

property OnDataFromClient: TScData;
```

Description

Occurs when a new data chunk from an SSH client is received and decrypted.

Parameters:

- `Sender` - the object whose event handler is called;
- `ChannelInfo` - holds the information about the current SSH channel;
- `Buffer` - points to the buffer that contains received data;
- `Offset` - zero-based byte offset in `Buffer` that indicates location of the received data;
- `Count` - length of the received data.

See Also

OnDataToClient

5.99.4.12 OnDataToClient

```
type
  TScData = procedure(Sender: TObject; ChannelInfo: TScSSHChannelInfo;
    const Buffer: TBytes; const Offset, Count: integer) of object;

property OnDataToClient: TScData;
```

Description

Occurs after a data chunk is received from the host with which connection is established in the current channel, and before the data will be encrypted and sent to the SSH client.

Parameters:

- `Sender` - the object whose event handler is called;
- `ChannelInfo` - holds the information about the current SSH channel;
- `Buffer` - points to the buffer that contains data to be transferred;
- `Offset` - zero-based byte offset in `Buffer` that indicates location of the data to be encrypted and transferred;
- `Count` - length of the data to be transferred.

See Also

OnDataFromClient

5.99.4.13 OnError

```
type
  TScErrorEvent = procedure(Sender: TObject; E: Exception) of object;

property OnError: TScErrorEvent;
```

Description

This event occurs, if an error arise in the main thread of the SSH server.

Event handler is called in the same thread where the exception arose.

Parameters:

- `Sender` - the object whose event handler is called;
- `E` - the object that describes the exception.

See Also

Active

5.99.4.14 OnRemotePortForwardingRequest

```
type
  TScOnRemotePortForwardingRequest = procedure(Sender: TObject;
    ClientInfo: TScSSHClientInfo; const Host: string; const Port:
    Integer; var Allow: Boolean) of object;

property OnRemotePortForwardingRequest: TScOnRemotePortForwardingRequest;
```

Description

The **OnRemotePortForwardingRequest** event occurs when remote port forwarding is requested from an SSH client side.

Parameters:

- `Sender` - the object whose event handler is called;
- `ClientInfo` - holds information about the current connection;
- `Host` - points to the host from which data should be forwarded;
- `Port` - specifies the number of the port to listen and to forward data to;
- `Allow` - set this parameter to True if you want to allow remote port forwarding from the specified host and port. Set `Allow` to False to disable port forwarding.

See Also

OnCancelRemotePortForwardingRequest

5.100 TScSFTPSessionInfo

5.100.1 Description

Unit

ScSSHUtils

Description

The **TScSFTPSessionInfo** contains the information about an SFTP session.

See also

TScSSHClientInfo

5.100.2 Properties

5.100.2.1 Client

```
property Client: TScSSHClientInfo;
```

Description

The **Client** property holds information about the current SSH connection.

This property is read-only.

5.100.2.2 Data

```
property Data: TObject;
```

Description

Data has no predefined meaning. The **Data** property is provided for the convenience of developers. It can be used for storing an additional object.

5.100.2.3 EOL

```
property EOL: string;
```

Description

The **EOL** property defines value of the end-of-line marker. **EOL** is newline sequence used on an SFTP server. It is used in order to process text files in a cross platform compatible way correctly.

The server sends the **EOL** value to an SFTP client when establishing a connection. Therefore, it can be changed by user from a default value only in the TScSFTPServer.OnOpen event handler.

The default value is '#13#10' for Windows platform and '#10' for non-Widows platforms.

5.100.2.4 HomePath

```
property HomePath: string;
```

Description

The **HomePath** property represents path to a root directory used by SFTP server for the current session.

When opening a new SFTP session, the following order of setting a root directory for the session is used:

At first, the TScUser.HomePath property value of the current user is checked. If this field is not empty, its value is taken. If it is empty, then the root directory for the session is set from the TScSFTPServer.DefaultRootPath property. Otherwise, the root directory for the session is set as the name of the current directory.

See Also

TScSFTPServer.DefaultRootPath

5.100.2.5 UseUnicode

```
property UseUnicode: boolean;
```

Description

The **UseUnicode** property specifies, whether UTF8 conversion is to be used by the server when parsing file names. **UseUnicode** is set automatically according to protocol flow, but user could also set it to the desired value.

The default value is True.

5.100.2.6 Version

```
property Version: integer;
```

Description

The **Version** property represents the version of the SFTP protocol.

This property is read-only.

5.101 TScHandle

5.101.1 Description

Unit

ScSFTPServer

Description

The **TScHandle** contains the full name and the operating system handle to a file or a directory.

See also

TScSFTPServer

5.101.2 Properties

5.101.2.1 FullFileName

```
property FullFileName: string;
```

Description

The **FullFileName** property holds the path and name to the referenced file or directory.

This property is read-only.

5.101.2.2 Handle

```
property Handle: THandle;
```

Description

The **Handle** property holds the operating system handle to the referenced file or directory.

5.102 TScSearchRec

5.102.1 Description

Unit

ScSFTPServer

Description

The **TScSearchRec** contains reference to the TSearchRec record that defines information about the file or directory.

See also

TScSFTPServer

5.102.2 Properties

5.102.2.1 SearchRec

```
property SearchRec: TSearchRec;
```

Description

The **SearchRec** property reperesents the TSearchRec record that defines information about the file or directory.

5.103 TScSFTPServer

5.103.1 Description

Unit

ScSFTPServer

Description

The **TScSFTPServer** component implements functionality of SFTP server.

To start SFTP server, it is enough to create the **TScSFTPServer** object and assign it to the TScSSHServer.SFTPServer property.

See Also

TScSSHServer.SFTPServer

5.103.2 Properties

5.103.2.1 DefaultRootPath

```
property DefaultRootPath: string;
```

Description

The **DefaultRootPath** property represents a path to the root directory used by SFTP server by default.

When opening a new SFTP session, the following order of setting a root directory for the session is used:

At first, the TScUser.HomePath property value of the current user is checked. If this field is not empty, its value is taken. If it is empty, then the root directory for the session is set from the **DefaultRootPath** property. Otherwise, the root directory for the session is set as the name of the current directory.

See Also

TScUser.HomePath

TScSFTPSessionInfo.HomePath

5.103.2.2 UseUnicode

```
property UseUnicode: boolean; default True;
```

Description

The **UseUnicode** property specifies, whether UTF8 conversion is to be used by the server when parsing file names.

The default value is True.

5.103.3 Methods

5.103.3.1 DefaultBlockFile

```
procedure DefaultBlockFile(SFTPSessionInfo: TScSFTPSessionInfo; Data:
TObject; const Offset, Len: Int64; const BlockModes: TScSFTPBlockModes;
var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultBlockFile** method to create a byte-range lock on a file specified by the `Data` object. The lock can be either mandatory (the server enforces that no other process or client can perform operations violating the lock) or advisory (no other processes can obtain a conflicting lock, but the server does not enforce that no operation violates the lock).

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a blocking file as the TScHandle object. Usually this object is previously returned by the DefaultOpenFile method.
- `Offset` - zero-based byte offset in the file that indicates the beginning of the byte-range to lock.
- `Len` - the number of bytes in the range to lock. The special value 0 means a lock from `Offset` to the end of the file.
- `BlockModes` - the blocking mode.
- `Error` - returns the information about an error that can arise when blocking a file.

See also

OnBlockFile

5.103.3.2 DefaultCloseFile

```
procedure DefaultCloseFile(SFTPSessionInfo: TScSFTPSessionInfo; Data:
TObject; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultCloseFile** method to close an opened handle of a file or directory specified by the `Data` object.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a closing file or directory as the TScHandle or TScSearchRec object. Usually this object is previously returned by the DefaultOpenFile or DefaultOpenDirectory methods.
- `Error` - returns the information about an error that can arise when closing a file.

See also

OnCloseFile

5.103.3.3 DefaultCreateLink

```
procedure DefaultCreateLink(SFTPSessionInfo: TScSFTPSessionInfo; const
LinkPath, TargetPath: string; Symbolic: boolean; var Error:
TScSFTPError); virtual;
```

Description

Call the **DefaultCreateLink** method to create either hard or symbolic link.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `LinkPath` - specifies the path name of the new link to create.
- `TargetPath` - specifies the path of an existing file system object to which the new-link-path will refer.
- `Symbolic` - determines if the link will be a symbolic link, or a special file that redirects file system parsing to the resulting path. If `Symbolic` is false, the link will be a hard link, or a second directory entry referring to the same file or directory object.
- `Error` - returns the information about an error that can arise when creating a link.

See also

OnCreateLink

5.103.3.4 DefaultGetAbsolutePath

```
procedure DefaultGetAbsolutePath(SFTPSessionInfo: TScSFTPSessionInfo;
const Path: string; const Control: TScSFTPRealpathControl; ComposePath:
TStringList; var AbsolutePath: string; out Error: TScSFTPError); virtual;
```

Description

Call the **DefaultGetAbsolutePath** method to canonize the given path name to the absolute canonical one. **DefaultGetAbsolutePath** converts path names containing ".." components or relative path names without a leading slash into absolute paths.

To get an absolute path from a relative one, the SFTPSessionInfo.HomePath property value is added at the beginning of the relative path.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.

- `Path` - original path which should be resolved into an absolute canonical path.
- `Control` - the parameters of identifying the absolute path.
- `ComposePath` - specifies multiple elements, in which case the method will build the resulting path by applying each compose path to the accumulated result until all elements have been applied.
- `AbsolutePath` - returns the resolved absolute path.
- `Error` - returns the information about an error that can arise when resolving an absolute path.

See also

OnGetAbsolutePath

5.103.3.5 DefaultGetFullPath

```
procedure DefaultGetFullPath(SFTPSessionInfo: TScSFTPSessionInfo; var
Path: string);
```

Description

Call the **DefaultGetFullPath** method to canonize the given path name to the absolute canonical one. **DefaultGetFullPath** converts path name containing ".." components or relative path name without a leading slash into the absolute path.

To get an absolute path from a relative one, the SFTPSessionInfo.HomePath property value is added at the beginning of the relative path.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - original path which should be resolved into the absolute canonical path.

See also

OnGetFullPath

TScSFTPSessionInfo.HomePath

DefaultRootPath

5.103.3.6 DefaultMakeDirectory

```
procedure DefaultMakeDirectory(SFTPSessionInfo: TScSFTPSessionInfo; const
Path: string; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultMakeDirectory** method to create a new directory.

Parameters:

- SFTPSessionInfo - contains the information about the current SFTP session.
- Path - specifies the directory to be created.
- Error - returns the information about an error that can arise when creating a directory.

See also

OnMakeDirectory

5.103.3.7 DefaultOpenDirectory

```
procedure DefaultOpenDirectory(SFTPSessionInfo: TScSFTPSessionInfo; const
Path: string; out Data: TObject; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultOpenDirectory** method to open an existing directory for reading.

The obtained Data object may be used in other methods, for example, in DefaultReadDirectory, DefaultCloseFile.

Parameters:

- SFTPSessionInfo - contains the information about the current SFTP session.
- Path - is the path name of the directory to be listed (without any trailing slash). If Path does not refer to a directory, the method returns an error.
- Data - returns the information about an opened directory as the TScSearchRec object.
- Error - returns the information about an error that can arise when opening a directory.

See also

OnOpenDirectory

5.103.3.8 DefaultOpenFile

```
procedure DefaultOpenFile(SFTPSessionInfo: TScSFTPSessionInfo; const
FileName: string; const OpenAttributes: TScSFTPFileOpenAttributes; out
Data: TObject; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultOpenFile** method to open or create a file.

The obtained Data object may be used in other methods, for example, in DefaultReadFile, DefaultWriteFile, DefaultCloseFile.

Parameters:

- SFTPSessionInfo - contains the information about the current SFTP session.

- `FileName` - the name of the file that is being opened. If `FileName` is the name of a directory, an error will be raised.
- `OpenAttributes` - contains attributes for the file opening.
- `Data` - returns the information about an opened file as the TScHandle object.
- `Error` - returns the information about an error that can arise when opening a file.

See also

OnOpenFile

5.103.3.9 DefaultReadDirectory

```
procedure DefaultReadDirectory(SFTPSessionInfo: TScSFTPSessionInfo; Data:
TObject; FileInfo: TScSFTPFileInfo; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultReadDirectory** method to search the next file in a directory specified by the `Data` object and retrieve the information about this file in the `FileInfo` object. If a file is not found, the erEof error is returned.

In order to obtain a complete directory listing, the user must call this method until the erEof error will be returned.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about the reading directory as the TScSearchRec object. Usually this object is previously returned by the DefaultOpenDirectory method.
- `FileInfo` - the object that will contain the information about the found file.
- `Error` - returns the information about an error that can arise when reading a directory. The erEof error is returned if file is not found.

See also

OnReadDirectory

5.103.3.10 DefaultReadFile

```
procedure DefaultReadFile(SFTPSessionInfo: TScSFTPSessionInfo; Data:
TObject; Offset: Int64; Count: cardinal; var Buffer: TBytes; var Read:
cardinal; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultReadFile** method to read data of file specified by the `Data` object.

DefaultReadFile returns the erEof error if the end of file was reached.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a reading file as the TScHandle object. Usually this object is previously returned by the DefaultOpenFile method.
- `Offset` - the offset in bytes relative to the beginning of the file that the read starts at. This parameter is ignored if TEXT MODE was specified during the open.
- `Count` - the maximum number of bytes to read.
- `Buffer` - the buffer to which the data will be read.
- `Read` - returns the amount of read data.
- `Error` - returns the information about an error that can arise when reading a file. The erEof error is returned if the end of file was reached.

See also

OnReadFile

5.103.3.11 DefaultReadSymbolicLink

```
procedure DefaultReadSymbolicLink(SFTPSessionInfo: TScSFTPSessionInfo;
const Path: string; out SymbolicName: string; var Error: TScSFTPError);
virtual;
```

Description

Call the **DefaultReadSymbolicLink** method to read the target of a symbolic link.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - the path name of the symbolic link to be read.
- `SymbolicName` - returns the target of the link.
- `Error` - returns the information about an error that can arise on the operation execution.

See also

OnReadSymbolicLink

5.103.3.12 DefaultRemoveDirectory

```
procedure DefaultRemoveDirectory(SFTPSessionInfo: TScSFTPSessionInfo;
const Path: string; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultRemoveDirectory** method to remove a directory. This method cannot be used to remove a file.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - specifies the directory to be removed.
- `Error` - returns the information about an error that can arise when removing a directory.

See also

OnRemoveDirectory

5.103.3.13 DefaultRemoveFile

```
procedure DefaultRemoveFile(SFTPSessionInfo: TScSFTPSessionInfo; const
FileName: string; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultRemoveFile** method to remove a file. This method cannot be used to remove directories.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `FileName` - specifies the name of the file to be removed.
- `Error` - returns the information about an error that can arise when removing a file.

See also

OnRemoveFile

5.103.3.14 DefaultRenameFile

```
procedure DefaultRenameFile(SFTPSessionInfo: TScSFTPSessionInfo; const
OldName, NewName: string; const Flags: TScSFTPRenameFlags; var Error:
TScSFTPError); virtual;
```

Description

Call the **DefaultRenameFile** method to rename file or directory.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `OldName` - the name of an existing file or directory.

- NewName - the new name for the file or directory.
- Flags - the renaming parameters.
- Error - returns the information about an error that can arise when renaming a file.

See also

OnRenameFile

5.103.3.15 DefaultRetrieveAttributes

```
procedure DefaultRetrieveAttributes(SFTPSessionInfo: TScSFTPSessionInfo;
const Path: string; FollowSymLink: boolean; const ReqAttrs:
TScSFTPAttributes; Attributes: TScSFTPFileAttributes; var Error:
TScSFTPError); virtual;
```

Description

Call the **DefaultRetrieveAttributes** method to retrieve the attributes for a named file.

DefaultRetrieveAttributes receives the file attributes and writes them to the Attributes object.

Parameters:

- SFTPSessionInfo - contains the information about the current SFTP session.
- Path - specifies the file system object for which attributes should be returned.
- FollowSymLink - specifies if the file follows symbolic links.
- ReqAttrs - specifies the file attributes which should be retrieving.
- Attributes - an object to which the attributes of the requested file will be written.
- Error - returns the information about an error that can arise when retrieving file attributes.

See also

OnRetrieveAttributes

5.103.3.16 DefaultRetrieveAttributesByHandle

```
procedure DefaultRetrieveAttributesByHandle(SFTPSessionInfo:
TScSFTPSessionInfo; Data: TObject; const ReqAttrs: TScSFTPAttributes;
Attributes: TScSFTPFileAttributes; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultRetrieveAttributesByHandle** method to retrieve the attributes for a file specified by the Data object.

DefaultRetrieveAttributesByHandle receives the file attributes and writes them to the Attributes object.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a requested file or directory as the TScHandle or TScSearchRec object. Usually this object is previously returned by the DefaultOpenFile or DefaultOpenDirectory methods.
- `ReqAttrs` - specifies the file attributes which should be retrieving.
- `Attributes` - an object to which the attributes of the requested file will be written.
- `Error` - returns the information about an error that can arise when retrieving file attributes.

See also

OnRetrieveAttributesByHandle

5.103.3.17 DefaultSetAttributes

```
procedure DefaultSetAttributes(SFTPSessionInfo: TScSFTPSessionInfo; const
Path: string; Attributes: TScSFTPFileAttributes; var Error:
TScSFTPError); virtual;
```

Description

Call the **DefaultSetAttributes** method to set the attributes for a named file.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - the file system object (e.g. file or directory) whose attributes are to be modified. If this object does not exist, or the user does not have sufficient access to write the attributes, the method returns an error.
- `Attributes` - object, that specifies the modified attributes to be applied.
- `Error` - returns the information about an error that can arise when setting file attributes.

See also

OnSetAttributes

5.103.3.18 DefaultSetAttributesByHandle

```
procedure DefaultSetAttributesByHandle(SFTPSessionInfo:
TScSFTPSessionInfo; Data: TObject; Attributes: TScSFTPFileAttributes; var
Error: TScSFTPError); virtual;
```

Description

Call the **DefaultSetAttributesByHandle** method to set the attributes for a file specified by the `Data`

object.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a modifying file or directory as the TScHandle or TScSearchRec object. Usually this object is previously returned by the DefaultOpenFile or DefaultOpenDirectory methods.
- `Attributes` - object, that specifies the modified attributes to be applied.
- `Error` - returns the information about an error that can arise when setting file attributes.

See also

OnSetAttributesByHandle

5.103.3.19 DefaultUnBlockFile

```
procedure DefaultUnBlockFile(SFTPSessionInfo: TScSFTPSessionInfo; Data:
TObject; const Offset, Len: Int64; var Error: TScSFTPError); virtual;
```

Description

Call the **DefaultUnBlockFile** method to remove a previously acquired byte-range lock on the file specified by the `Data` object.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about an unblocking file as the TScHandle object. Usually this object is previously returned by the DefaultOpenFile method.
- `Offset` - the beginning of the byte-range to lock.
- `Len` - the number of bytes in the range to lock. The special value 0 means lock from `Offset` to the end of the file.
- `Error` - returns the information about an error that can arise when unblocking a file.

See also

OnUnBlockFile

5.103.3.20 DefaultWriteFile

```
procedure DefaultWriteFile(SFTPSessionInfo: TScSFTPSessionInfo; Data:
TObject; Offset: Int64; const Buffer: TBytes; Count: integer; var Error:
TScSFTPError); virtual;
```

Description

Call the **DefaultWriteFile** method to write data to the file specified by the `Data` object.

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a writing file as the TScHandle object. Usually this object is previously returned by the DefaultOpenFile method.
- `Offset` - the offset in bytes relative to the beginning of the file that the writing started at. This field is ignored if TEXT MODE was specified during the opening.
- `Buffer` - the sequence of bytes that should be written to the file.
- `Count` - the number of bytes to write.
- `Error` - returns the information about an error that can arise when writing a file.

See also

OnWriteFile

5.103.3.21 GetCanonicalPath

```
function GetCanonicalPath(const Path: string): string; virtual;
```

Description

Call the **GetCanonicalPath** method to canonize the given path name to the canonical one. **GetCanonicalPath** converts path names containing ".." components or relative path names without a leading slash into canonical paths.

Parameters:

- `Path` - original path which should be resolved into the canonical path.
- `Result` - returns the resolved canonical path.

See also

GetFullPath

5.103.3.22 GetFullPath

```
function GetFullPath(SFTPSessionInfo: TScSFTPSessionInfo; const Path:
string): string; virtual;
```

Description

Call the **GetFullPath** method to canonize the given path name to the absolute canonical one. **GetFullPath** converts path name containing ".." components or relative path name without a leading slash into the absolute path.

To get an absolute path from a relative one, the SFTPSessionInfo.HomePath property value is added at the beginning of the relative path.

GetFullPath calls the OnGetFullPath event handler, if it is set, or, otherwise, the DefaultGetFullPath method to execute this operation

Parameters:

- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - original path which should be resolved into the absolute canonical path.
- `Result` - returns the resolved absolute path.

See also

OnGetFullPath

TScSFTPSessionInfo.HomePath

DefaultRootPath

5.103.4 Events

5.103.4.1 OnBlockFile

```
type
  TScSFTPServerBlockFileEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; const Offset,
     Len: Int64; const BlockModes: TScSFTPBlockModes; var Error:
     TScSFTPError) of object;

property OnBlockFile: TScSFTPServerBlockFileEvent;
```

Description

The **OnBlockFile** event occurs on request from an SFTP client to create a byte-range lock on a file specified by the `Data` object. The lock can be either mandatory (the server enforces that no other process or client can perform operations violating the lock) or advisory (no other processes can obtain a conflicting lock, but the server does not enforce that no operation violates the lock).

You can call the DefaultBlockFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a blocking file as the TScHandle object or any user's object. Usually this object is previously returned by the DefaultOpenFile method or the OnOpenFile event handler.
- `Offset` - zero-based byte offset in the file that indicates the beginning of the byte-range to lock.

- `Len` - the number of bytes in the range to lock. The special value 0 means a lock from `Offset` to the end of the file.
- `BlockModes` - the blocking mode.
- `Error` - a parameter to pass the information about an error that can arise when blocking a file.

See also

DefaultBlockFile

5.103.4.2 OnClose

```
type
  TScSFTPServerCloseEvent = procedure(Sender: TObject; SFTPSessionInfo:
     TScSFTPSessionInfo) of object;

property OnClose: TScSFTPServerCloseEvent;
```

Description

The **OnClose** event occurs after an SFTP session is closed.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.

5.103.4.3 OnCloseFile

```
type
  TScSFTPServerCloseFileEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; var Error:
     TScSFTPError) of object;

property OnCloseFile: TScSFTPServerCloseFileEvent;
```

Description

The **OnCloseFile** event occurs on request from an SFTP client to close an opened handle of a file or directory specified by the `Data` object.

You can call the DefaultCloseFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.

- SFTPSessionInfo - contains the information about the current SFTP session.
- Data - specifies the information about a closing file or directory as the TScHandle or TScSearchRec object or any user's object. Usually this object is previously returned by the DefaultOpenFile or DefaultOpenDirectory methods, or the OnOpenFile or OnOpenDirectory event handlers.
- Error - a parameter to pass the information about an error that can arise when closing a file.

See also

DefaultCloseFile

5.103.4.4 OnCreateLink

```
type
  TScSFTPServerCreateLinkEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; const LinkPath, TargetPath:
     string; Symbolic: boolean; var Error: TScSFTPError) of object;

property OnCreateLink: TScSFTPServerCreateLinkEvent;
```

Description

The **OnCreateLink** event occurs on request from an SFTP client to create either hard or symbolic link.

You can call the DefaultCreateLink method to execute this operation or write your own implementation.

Parameters:

- Sender - the object whose event handler is called.
- SFTPSessionInfo - contains the information about the current SFTP session.
- LinkPath - specifies the path name of the new link to create. To get an absolute file path, use the GetFullPath method.
- TargetPath - specifies the path of an existing file system object to which the new-link-path should refer.
- Symbolic - determines if the link will be a symbolic link, or a special file that redirects file system parsing to the resulting path. If Symbolic is false, the link should be a hard link, or a second directory entry referring to the same file or directory object.
- Error - a parameter to pass the information about an error that can arise when creating a link.

See also

DefaultCreateLink

5.103.4.5 OnGetAbsolutePath

```
type
  TScSFTPServerGetAbsolutePathEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; const Path: string; const
     Control: TScSFTPRealpathControl; ComposePath: TStringList; out
     AbsolutePath: string; var Error: TScSFTPError) of object;

property OnGetAbsolutePath: TScSFTPServerGetAbsolutePathEvent;
```

Description

The **OnGetAbsolutePath** event occurs on request from an SFTP client to canonize the given path name to the absolute canonical one. The **OnGetAbsolutePath** event handler should convert path names containing ".." components or relative path names without a leading slash into absolute paths. To get an absolute path from a relative one, the SFTPSessionInfo.HomePath property value should be added at the beginning of the relative path.

You can call the DefaultGetAbsolutePath method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - original path which should be resolved into an absolute canonical path.
- `Control` - the parameters of identifying the absolute path.
- `ComposePath` - specifies multiple elements, in which case an event handler should build the resulting path by applying each compose path to the accumulated result until all elements have been applied.
- `AbsolutePath` - a parameter to pass the resolved absolute path.
- `Error` - a parameter to pass the information about an error that can arise when resolving an absolute path.

See also

DefaultGetAbsolutePath

TScSFTPSessionInfo.HomePath

5.103.4.6 OnGetFullPath

```
type
  TScSFTPServerGetFullPathEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; var Path: string) of object;
```

```
property OnGetFullPath: TScSFTPServerGetFullPathEvent;
```

Description

The **OnGetFullPath** event occurs when calling the GetFullPath method. **OnGetFullPath** occurs after the absolute canonical path is resolved. The **OnGetFullPath** event handler can change this value at its own discretion.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - the resolved absolute canonical path. It is a variable parameter and it can be changed.

See also

GetFullPath

5.103.4.7 OnMakeDirectory

```
type
  TScSFTPServerMakeDirectoryEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; const Path: string; var Error:
     TScSFTPError) of object;

property OnMakeDirectory: TScSFTPServerMakeDirectoryEvent;
```

Description

The **OnMakeDirectory** event occurs on request from an SFTP client to create a new directory.

You can call the DefaultMakeDirectory method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - specifies the directory to be created. To get an absolute directory path, use the GetFullPath method.
- `Error` - a parameter to pass the information about an error that can arise when creating a directory.

See also

DefaultMakeDirectory

5.103.4.8 OnOpen

```
type
  TScSFTPServerOpenEvent = procedure(Sender: TObject; SFTPSessionInfo:
     TScSFTPSessionInfo) of object;

property OnOpen: TScSFTPServerOpenEvent;
```

Description

The **OnOpen** event occurs before opening a new SFTP session.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.

5.103.4.9 OnOpenDirectory

```
type
  TScSFTPServerOpenDirectoryEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; const Path: string; out Data:
     TObject; var Error: TScSFTPError) of object;

property OnOpenDirectory: TScSFTPServerOpenDirectoryEvent;
```

Description

The **OnOpenDirectory** event occurs on request from an SFTP client to open an existing directory for reading. The returned `Data` object may be used in other event handlers, for example, in OnReadDirectory, OnCloseFile.

You can call the DefaultOpenDirectory method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - is the path name of the directory to be listed (without any trailing slash). If `Path` does not refer to a directory, an event handler should return an error. To get an absolute directory path, use the GetFullPath method.
- `Data` - a parameter to pass the information about an opened directory as the TScSearchRec object or any user's object.
- `Error` - a parameter to pass the information about an error that can arise when opening a

directory.

See also

DefaultOpenDirectory

DefaultRootPath

5.103.4.10 OnOpenFile

type

```
TScSFTPServerOpenFileEvent = procedure(Sender: TObject;
   SFTPSessionInfo: TScSFTPSessionInfo; const FileName: string; const
   OpenAttributes: TScSFTPFileOpenAttributes; out Data: TObject; var
   Error: TScSFTPError) of object;

property OnOpenFile: TScSFTPServerOpenFileEvent;
```

Description

The **OnOpenFile** event occurs on request from an SFTP client to open or create a file. The returned `Data` object may be used in other event handlers, for example, in OnReadFile, OnWriteFile, OnCloseFile.

You can call the DefaultOpenFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `FileName` - the name of the file that is being opened. If `FileName` is the name of a directory, an event handler should return an error. To get an absolute file path, use the GetFullPath method.
- `OpenAttributes` - contains attributes for the file opening.
- `Data` - a parameter to pass the information about an opened file as the TScHandle object or any user's object.
- `Error` - a parameter to pass the information about an error that can arise when opening a file.

See also

DefaultOpenFile

GetFullPath

5.103.4.11 OnReadDirectory

type

```
TScSFTPServerReadDirectoryEvent = procedure(Sender: TObject;
   SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; FileInfo:
```

```
    TScSFTPFileInfo; var Error: TScSFTPError) of object;

property OnReadDirectory: TScSFTPServerReadDirectoryEvent;
```

Description

The **OnReadDirectory** event occurs on request from an SFTP client to search the next file in a directory specified by the `Data` object and retrieve the information about this file. If a file is not found, the erEof error should be returned.

In order to obtain a complete directory listing, the user can process this request until the erEof error will be returned.

You can call the DefaultReadDirectory method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a reading directory as the TScSearchRec object or any user's object. Usually this object is previously returned by the DefaultOpenDirectory method or the OnOpenDirectory event handler.
- `FileInfo` - the object that will contain the information about the found file.
- `Error` - a parameter to pass the information about an error that can arise when reading a directory. The erEof error should be returned if file is not found.

See also

DefaultReadDirectory

5.103.4.11 OnReadFile

```
type
  TScSFTPServerReadFileEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; Offset: Int64;
     Count: cardinal; var Buffer: TBytes; var Read: cardinal; var Error:
     TScSFTPError) of object;

property OnReadFile: TScSFTPServerReadFileEvent;
```

Description

The **OnReadFile** event occurs on request from an SFTP client to read data of file specified by the `Data` object. The **OnReadFile** event handler should return the erEof error if the end of file was reached.

You can call the DefaultReadFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a reading file as the TScHandle object or any user's object. Usually this object is previously returned by the DefaultOpenFile method or the OnOpenFile event handler.
- `Offset` - the offset in bytes relative to the beginning of the file that the read starts at. This parameter is ignored if TEXT MODE was specified during the open.
- `Count` - the maximum number of bytes to read.
- `Buffer` - the buffer to which the data should be read.
- `Read` - a parameter to pass the amount of read data.
- `Error` - a parameter to pass the information about an error that can arise when reading a file. The erEof error should be returned if the end of file was reached.

See also

DefaultReadFile

5.103.4.13 OnReadSymbolicLink

type

```
TScSFTPServerReadSymbolicLinkEvent = procedure(Sender: TObject;
   SFTPSessionInfo: TScSFTPSessionInfo; const Path: string; out
   SymbolicName: string; var Error: TScSFTPError) of object;

property OnReadSymbolicLink: TScSFTPServerReadSymbolicLinkEvent;
```

Description

The **OnReadSymbolicLink** event occurs on request from an SFTP client to read the target of a symbolic link.

You can call the DefaultReadSymbolicLink method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - the path name of the symbolic link to be read. To get an absolute file path, use the GetFullPath method.
- `SymbolicName` - a parameter to pass the target of the link.
- `Error` - a parameter to pass the information about an error that can arise on the operation execution.

See also

DefaultReadSymbolicLink

5.103.4.14 OnRemoveDirectory

```
type
  TScSFTPServerRemoveDirectoryEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; const Path: string; var Error:
     TScSFTPError) of object;

property OnRemoveDirectory: TScSFTPServerRemoveDirectoryEvent;
```

Description

The **OnRemoveDirectory** event occurs on request from an SFTP client to remove a directory.

You can call the DefaultRemoveDirectory method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - specifies the directory to be removed. To get an absolute directory path, use the GetFullPath method.
- `Error` - a parameter to pass the information about an error that can arise when removing a directory.

See also

DefaultRemoveDirectory

5.103.4.15 OnRemoveFile

```
type
  TScSFTPServerRemoveFileEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; const FileName: string; var
     Error: TScSFTPError) of object;

property OnRemoveFile: TScSFTPServerRemoveFileEvent;
```

Description

The **OnRemoveFile** event occurs on request from an SFTP client to remove a file.

You can call the DefaultRemoveFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `FileName` - specifies the name of the file to be removed. To get an absolute file path, use the GetFullPath method.
- `Error` - a parameter to pass the information about an error that can arise when removing a file.

See also

DefaultRemoveFile

5.103.4.16 OnRenameFile

type

```
TScSFTPServerRenameFileEvent = procedure(Sender: TObject;
   SFTPSessionInfo: TScSFTPSessionInfo; const OldName, NewName: string;
   const Flags: TScSFTPRenameFlags; var Error: TScSFTPError) of object;

property OnRenameFile: TScSFTPServerRenameFileEvent;
```

Description

The **OnRenameFile** event occurs on request from an SFTP client to rename file or directory.

You can call the DefaultRenameFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `OldName` - the name of an existing file or directory. To get an absolute file path, use the GetFullPath method.
- `NewName` - the new name for the file or directory.
- `Flags` - the renaming parameters.
- `Error` - a parameter to pass the information about an error that can arise when renaming a file.

See also

DefaultRenameFile

5.103.4.17 OnRequestFileSecurityAttributes

type

```
  TScOnRequestFileSecurityAttributes = procedure(Sender: TObject;
     Attributes: TScSFTPFileAttributes; const Path: string;
     SecurityDescriptor: Pointer) of object;

property OnRequestFileSecurityAttributes:
TScOnRequestFileSecurityAttributes;
```

Description

The **OnRequestFileSecurityAttributes** event occurs on request from an SFTP client to read a directory.

This event occurs in the DefaultReadDirectory method to retrieve the security attributes for the file. The **OnRequestFileSecurityAttributes** event handler should write the file permissions attribute (TScSFTPFileAttributes.Permissions) and the file ACL and ACE attributes (TScSFTPFileAttributes.AclFlags, TScSFTPFileAttributes.ACEs) to the `Attributes` object.

Parameters:

- `Sender` - the object whose event handler is called.
- `Attributes` - an object to which the security attributes of the requested file should be written.
- `Path` - specifies the file system object for which security attributes should be returned.
- `SecurityDescriptor` - a pointer to a buffer that contains the security descriptor of the requested file. This descriptor is returned by the GetFileSecurity function of Windows OS.

5.103.4.11 OnRetrieveAttributes

```
type
  TScSFTPServerRetrieveAttributesEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; const Path: string;
     FollowSymLink: boolean; const ReqAttrs: TScSFTPAttributes;
     Attributes: TScSFTPFileAttributes; var Error: TScSFTPError) of
     object;

property OnRetrieveAttributes: TScSFTPServerRetrieveAttributesEvent;
```

Description

The **OnRetrieveAttributes** event occurs on request from an SFTP client to retrieve the attributes for a named file. The **OnRetrieveAttributes** event handler should receive the file attributes and write them to the `Attributes` object.

You can call the DefaultRetrieveAttributes method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - specifies the file system object for which attributes should be returned. To get an absolute file path, use the GetFullPath method.
- `FollowSymLink` - specifies if the file follows symbolic links.
- `ReqAttrs` - specifies the file attributes which should be retrieving.
- `Attributes` - an object to which the attributes of the requested file should be written.
- `Error` - a parameter to pass the information about an error that can arise when retrieving file attributes.

See also

DefaultRetrieveAttributes

5.103.4.15 OnRetrieveAttributesByHandle

```
type
  TScSFTPServerRetrieveAttributesByHandleEvent = procedure(Sender:
     TObject; SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; const
     ReqAttrs: TScSFTPAttributes; Attributes: TScSFTPFileAttributes; var
     Error: TScSFTPError) of object;

property OnRetrieveAttributesByHandle:
TScSFTPServerRetrieveAttributesByHandleEvent;
```

Description

The **OnRetrieveAttributesByHandle** event occurs on request from an SFTP client to retrieve the attributes for a file specified by the `Data` object. The **OnRetrieveAttributesByHandle** event handler should receive the file attributes and write them to the `Attributes` object.

You can call the DefaultRetrieveAttributesByHandle method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a requested file or directory as the TScHandle or TScSearchRec object or any user's object. Usually this object is previously returned by the DefaultOpenFile or DefaultOpenDirectory methods, or the OnOpenFile or OnOpenDirectory event handlers.
- `ReqAttrs` - specifies the file attributes which should be retrieving.
- `Attributes` - an object to which the attributes of the requested file should be written.
- `Error` - a parameter to pass the information about an error that can arise when retrieving file attributes.

See also

DefaultRetrieveAttributesByHandle

5.103.4.2 OnSetAttributes

type

```
TScSFTPServerSetAttributesEvent = procedure(Sender: TObject;
   SFTPSessionInfo: TScSFTPSessionInfo; const Path: string;
   Attributes: TScSFTPFileAttributes; var Error: TScSFTPError) of
   object;
```

property OnSetAttributes: TScSFTPServerSetAttributesEvent;

Description

The **OnSetAttributes** event occurs on request from an SFTP client to set the attributes for a named file.

You can call the DefaultSetAttributes method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Path` - the file system object (e.g. file or directory) whose attributes are to be modified. If this object does not exist, or the user does not have sufficient access to write the attributes, an event handler should return an error. To get an absolute file path, use the GetFullPath method.
- `Attributes` - an object, that specifies the modified attributes to be applied.
- `Error` - a parameter to pass the information about an error that can arise when setting file attributes.

See also

DefaultSetAttributes

5.103.4.2 OnSetAttributesByHandle

type

```
TScSFTPServerSetAttributesByHandleEvent = procedure(Sender: TObject;
   SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; Attributes:
   TScSFTPFileAttributes; var Error: TScSFTPError) of object;
```

property OnSetAttributesByHandle:
TScSFTPServerSetAttributesByHandleEvent;

Description

The **OnSetAttributesByHandle** event occurs on request from an SFTP client to set the attributes for a file specified by the `Data` object.

You can call the DefaultSetAttributesByHandle method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a modifying file or directory as the TScHandle or TScSearchRec object or any user's object. Usually this object is previously returned by the DefaultOpenFile or DefaultOpenDirectory methods, or the OnOpenFile or OnOpenDirectory event handlers.
- `Attributes` - an object, that specifies the modified attributes to be applied.
- `Error` - a parameter to pass the information about an error that can arise when setting file attributes.

See also

DefaultSetAttributesByHandle

5.103.4.21 OnUnBlockFile

type

```
TScSFTPServerUnBlockFileEvent = procedure(Sender: TObject;
   SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; const Offset,
   Len: Int64; var Error: TScSFTPError) of object;

property OnUnBlockFile: TScSFTPServerUnBlockFileEvent;
```

Description

The **OnUnBlockFile** event occurs on request from an SFTP client to remove a previously acquired byte-range lock on the file specified by the `Data` object.

You can call the DefaultUnBlockFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about an unblocking file as the TScHandle object or any user's object. Usually this object is previously returned by the DefaultOpenFile method or the OnOpenFile event handler.

- `Offset` - the beginning of the byte-range to lock.
- `Len` - the number of bytes in the range to lock. The special value 0 means lock from `Offset` to the end of the file.
- `Error` - a parameter to pass the information about an error that can arise when unblocking a file.

See also

DefaultUnBlockFile

5.103.4.23 OnWriteFile

```
type
  TScSFTPServerWriteFileEvent = procedure(Sender: TObject;
     SFTPSessionInfo: TScSFTPSessionInfo; Data: TObject; Offset: Int64;
     const Buffer: TBytes; Count: integer; var Error: TScSFTPError) of
     object;

property OnWriteFile: TScSFTPServerWriteFileEvent;
```

Description

The **OnWriteFile** event occurs on request from an SFTP client to write data to the file specified by the `Data` object.

You can call the DefaultWriteFile method to execute this operation or write your own implementation.

Parameters:

- `Sender` - the object whose event handler is called.
- `SFTPSessionInfo` - contains the information about the current SFTP session.
- `Data` - specifies the information about a writing file as the TScHandle object or any user's object. Usually this object is previously returned by the DefaultOpenFile method or the OnOpenFile event handler.
- `Offset` - the offset in bytes relative to the beginning of the file that the writing started at. This field is ignored if TEXT MODE was specified during the opening.
- `Buffer` - the sequence of bytes that should be written to the file.
- `Count` - the number of bytes to write.
- `Error` - a parameter to pass the information about an error that can arise when writing a file.

See also

DefaultWriteFile

5.104 TScSFTPClient

5.104.1 Description

Unit

ScSFTPClient

Description

The **TScSFTPClient** component implements functionality of SFTP client.

SFTP protocol provides secure file transfer (and more generally file system access). It is used to implement secure remote file system service, as well as secure file transfer service.

SFTP client runs over secure channel using the SSH protocol. On that the SFTP client authentication is performed on the SSH protocol level. The secure connection is provided by an SSH client that can be assigned to the SSHClient property.

Use the ReadBlockSize and WriteBlockSize properties to increase the performance.

The **TScSFTPClient** class throws an EScSFTPError exception when error occur during execution of any command to the SFTP server. The EScSFTPError.ErrorCode property contains a value that indicates the source of the error.

5.104.2 Properties

5.104.2.1 Active

```
property Active: boolean;
```

Description

Use the **Active** property to determine whether the connection to SFTP server is established.

This property is read-only.

5.104.2.2 EventsCallMode

```
property EventsCallMode: TScEventCallMode; default ecAsynchronous;
```

Description

The **EventsCallMode** property determines how the event handlers will be called. The thing is that data coming from the server is processed in a separate thread of the SSH connection. And the call of the event handlers can occur in a different way for synchronization with the main thread of the application.

The default value is the `ecAsynchronous` mode when the events are added to a queue and then

asynchronously synchronized from this queue with the main thread. This allows not slowing down the thread in which events occur and at the same calling the event handlers in the main thread.

When setting the property to the `ecSynchronous` value, the event call will be immediately synchronized with the main thread.

When setting the property to the `ecDirectly` value, there is no synchronization with the main thread.

Default value is the `ecAsynchronous` mode.

See Also

NonBlocking

5.104.2.3 NonBlocking

```
property NonBlocking: boolean; default False;
```

Description

Use the **NonBlocking** property to determine the data transferring mode to use: synchronous or asynchronous.

If **NonBlocking** is True, then all commands to the SFTP server will not block execution of other code in the application. Data is transferred in the asynchronous mode. The result of the command execution can be received only by processing corresponding event (for example, OnSuccess and OnError). If **NonBlocking** is False, then the result of command execution is returned by the method when returning control.

The default value is False.

5.104.2.4 PipelineLength

```
property PipelineLength: integer; default 32;
```

Description

Use the **PipelineLength** property to indicate the amount of upload or download requests, which are sent before waiting for all requests to be completed.

The transfer speed increases, in the case when more requests are sent. However, if there is an error, all requests are discarded. In addition, the memory consumption depends on the number of pending requests. You should set **PipelineLength** to 1 if the speed is not essential and the memory consumption is.

The default value is 32 requests.

5.104.2.5 ReadBlockSize

```
property ReadBlockSize: integer; default 65536;
```

Description

Use the **ReadBlockSize** property to determine the maximum size of the data block that will be sent as one query to the SFTP server when reading a file. Use this property to increase the application performance.

The default value is 65536.

See Also

ReadFile

WriteBlockSize

5.104.2.6 ServerProperties

```
property ServerProperties: TScSFTPServerProperties;
```

Description

The **ServerProperties** property holds the detailed information about the current SFTP server that the server may send when establishing a connection.

This property is read-only.

See Also

TScSFTPServerProperties

5.104.2.7 ServerVersion

```
property ServerVersion: TScSFTPVersion;
```

Description

The **ServerVersion** property holds the version of the SFTP protocol that is used in the current connection. It is set when establishing a connection to the SFTP server (on the Initialize method call).

This property is read-only.

See Also

Initialize

5.104.2.8 SSHClient

```
property SSHClient: TScSSHClient;
```

Description

Use the **SSHClient** property to determine the secure connection between an SSH client and the SSH server. This connection is used to exchange data. To create an SFTP connection, the **SSHClient** property should be set.

This property can be set at design time by selecting a TScSSHClient object from the provided list.

At runtime, set the **SSHClient** property to reference an existing TScSSHClient object.

5.104.2.9 Timeout

```
property Timeout: integer; default 15;
```

Description

Use the **Timeout** property to determine the amount of time during which the client makes attempts to obtain data from the server. It is measured in seconds.

The default value is 15 seconds.

5.104.2.10 UseUnicode

```
property UseUnicode: boolean; default False;
```

Description

The **UseUnicode** property specifies, whether UTF8 conversion is to be used by the client when parsing file names.

The default value is False.

5.104.2.11 Version

```
property Version: TScSFTPVersion;
```

Description

The **Version** property holds the version of the SFTP protocol the client is going to use.

If the client wants to interoperate with servers that support discontinued versions of the SFTP

protocol, it should set this property to vSFTP3, and then use the OnVersionSelect event handler.

The default value is vSFTP3.

See Also

Initialize

OnVersionSelect

5.104.2.11 WriteBlockSize

```
property WriteBlockSize: boolean; default 65536;
```

Description

Use the **WriteBlockSize** property to determine the maximum size of the data block that will be sent to the SFTP server as one query when writing to file. Use this property to increase the application performance.

The default value is 65536.

See Also

WriteFile

ReadBlockSize

5.104.3 Methods

5.104.3.1 Block

```
procedure Block(const Handle: TScSFTPFileHandle; Offset, Count: Int64;
BlockModes: TScSFTPBlockModes);
```

Description

Call the **Block** method to create a byte-range lock on the file specified by the handle. The lock can be either mandatory (the server enforces that no other process or client can perform operations violating the lock) or advisory (no other processes can obtain a conflicting lock, but the server does not enforce that no operation violates the lock).

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Note: This operation is supported starting with the version 6 of the SFTP protocol.

Parameters:

- `Handle` - a handle returned by OpenFile or OpenDirectory methods. Note that some servers may return the SSH_FX_OP_UNSUPPORTED error if the handle is a directory handle.

- `Offset` - the beginning of the byte-range to lock.
- `Count` - the number of bytes in the range to lock. The special value 0 means a lock from `Offset` to the end of the file.
- `BlockModes` - the blocking mode.

See also

OnError

OnSuccess

OpenFile

UnBlock

5.104.3.2 CheckFile

```
procedure CheckFile(const FileName: string; StartOffset, Length: Int64;
BlockSize: Integer; ReplyExtension: TScCheckFileReplyExtension = nil);
```

Description

Call the **CheckFile** method to check if a file (or its part) that client already has matches the one that is on the server.

If the NonBlocking property is False, **CheckFile** returns control after receiving an answer from the server and writing the results to the `ReplyExtension` object. Otherwise the result is written to the `ReplyExtension` object on executing the OnReplyCheckFile event. If the server returns an error, the OnError event is generated.

Note: this request is not supported by all SFTP servers.

Parameters:

- `FileName` - the path to the file to check. If `FileName` is a directory, an error will be returned. If `FileName` refers to a symbolic link, the target will be opened.
- `StartOffset` - the starting offset of the data to include to the hash.
- `Length` - the length of data to include to the hash. If the length is zero, all data from `StartOffset` to the end-of-file should be included.
- `BlockSize` - an independent hash that will be computed over every block in the file. The size of blocks is specified by `BlockSize`. The `BlockSize` must not be smaller than 256 bytes. If the block-size is 0, then only one hash over the entire range will be made.
- `ReplyExtension` - an object to which the computed hashes will be written. If this parameter is set to nil or is not set at all, then the OnReplyCheckFile event should be processed. If the object is specified, it will be returned in the OnReplyCheckFile event handler.

See also

CheckFileByHandle

OnError

OnReplyCheckFile

5.104.3.3 CheckFileByHandle

```
procedure CheckFileByHandle(const Handle: TScSFTPFileHandle; StartOffset,
Length: Int64; BlockSize: Integer; ReplyExtension:
TScCheckFileReplyExtension = nil);
```

Description

Call the **CheckFileByHandle** method to check if a file (or its part) that client already has matches the one that is on the server.

If the NonBlocking property is False,**CheckFileByHandle** returns control after receiving an answer from the server and writing the results to the `ReplyExtension` object. Otherwise the result is written to the `ReplyExtension` object on executing the OnReplyCheckFile event. If the server returns an error, the OnError event is generated.

Note: this request is not supported by all SFTP servers.

Parameters:

- `Handle` - a handle previously returned in the response to OpenFile.
- `StartOffset` - the starting offset of the data to include to the hash.
- `Length` - the length of data to include to the hash. If the length is zero, all data from `StartOffset` to the end-of-file should be included.
- `BlockSize` - an independent hash that will be computed over every block in the file. The size of blocks is specified by `BlockSize`. The `BlockSize` must not be smaller than 256 bytes. If the block-size is 0, then only one hash over the entire range will be made.
- `ReplyExtension` - an object to which the computed hashes will be written. If this parameter is set to nil or is not set at all, then the OnReplyCheckFile event should be processed. If the object is specified, it will be returned in the OnReplyCheckFile event handler.

See also

CheckFile

OnError

OnReplyCheckFile

5.104.3.4 CloseHandle

```
procedure CloseHandle(const Handle: TScSFTPFileHandle);
```

Description

Call the **CloseHandle** method to close an opened file handle.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by processing the OnSuccess and OnError events.

Parameters:

- `Handle` - a handle previously returned in the response to OpenFile or OpenDirectory. The handle becomes invalid immediately after this command was sent.

See Also

OnError

OnSuccess

OpenDirectory

OpenFile

5.104.3.5 CopyRemoteFile

```
procedure CopyRemoteFile(const Source, Destination: string; Overwrite:
Boolean);
```

Description

Call the **CopyRemoteFile** method to copy a file from one location to another on the server.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `Source` - holds the initial path to the file that is being copied.
- `Destination` - holds the destination path to copy the file to.
- `Overwrite` - specifies whether to overwrite the file with the same name if it exists.

See Also

OnError

OnSuccess

5.104.3.6 CreateLink

```
procedure CreateLink(const LinkPath, TargetPath: string; Symbolic:
```

```
boolean = True);
```

Description

Call the **CreateLink** method to create either hard or symbolic link on the server.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `LinkPath` - specifies the path name of the new link to create.
- `TargetPath` - specifies the path of an existing file system object to which the new-link-path will refer.
- `Symbolic` - the link should be a symbolic link, or a special file that redirects file system parsing to the resulting path. If `Symbolic` is false, the link should be a hard link, or a second directory entry referring to the same file or directory object. This parameter is supported starting with the version 4 of the SFTP protocol.

See Also

OnError

OnSuccess

5.104.3.7 Disconnect

```
procedure Disconnect;
```

Description

Call the **Disconnect** method to close an existing connection to the SFTP server. **Disconnect** sets the Active property to False.

See Also

Active

Initialize

5.104.3.8 DownloadFile

```
procedure DownloadFile(const Source, Destination: string; Overwrite:
Boolean; FileOffset: Int64 = 0);
```

Description

Call the **DownloadFile** method to copy a file from remote machine to the local.

To create local resulting file with the required attributes (for example, with the attributes

corresponding to the ones that the file on the server has) process the OnCreateLocalFile event. By default file with default attributes is created.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `Source` - holds the initial path to the file that is being copied.
- `Destination` - holds the destination path to copy the file to.
- `Overwrite` - specifies whether to overwrite the file with the same name if it exists.
- `FileOffset` - the offset in bytes relative to the beginning of the file that the downloading starts at.

See Also

OnCreateLocalFile

OnError

OnSuccess

5.104.3.9 DownloadToStream

```
procedure DownloadToStream(const SourceName: string; Destination:
TStream; FileOffset: Int64 = 0);
```

Description

Call the **DownloadToStream** method to copy a file from a remote machine to a local one.

To create a local resulting file with the required attributes (for example, with the attributes corresponding to the ones that the file on the server has) handle the OnCreateLocalFile event. By default, a file with the default attributes is created.

If the NonBlocking property is set to True, then the control is returned immediately and you can see the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `SourceName` - holds the initial path to the file that is being copied.
- `Destination` - holds the destination data stream to copy the file to.
- `FileOffset` - the offset in bytes relative to the beginning of the file where downloading starts at.

See Also

OnCreateLocalFile

OnError

OnSuccess

5.104.3.1 EOF

```
function EOF: boolean; overload;
function EOF(const Handle: TScSFTPFileHandle): boolean; overload;
```

Description

Use the **EOF** method to determine that an attempt to read past or required the end-of-file was made or that there are no more directory entries to return. This method has sense only when NonBlocking = False.

To find out the end-of-file state for a specific file, use the overloaded method, in which you should specify the required file in the `Handle` parameter.

See Also

NonBlocking

ReadDirectory

ReadFile

TextSeek

5.104.3.1 Initialize

```
procedure Initialize;
```

Description

Call the **Initialize** method to establish a connection to the SFTP server. **Initialize** sets the Active property to True.

If the Version property was set to the version 3 of the SFTP server before establishing a connection, and the server supports higher versions of the SFTP protocol, then the OnVersionSelect event may be raised. At that user can choose the required version of the SFTP protocol. After the connection was established **Initialize** sets the ServerVersion and ServerProperties properties.

See Also

Active

Disconnect

OnVersionSelect

ServerProperties

ServerVersion

5.104.3.11 MakeDirectory

```
procedure MakeDirectory(const Path: string; Attributes:
TScSFTPFileAttributes = nil);
```

Description

Call the **MakeDirectory** method to create new directory.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `Path` - specifies the directory to be created.
- `Attributes` - specifies the attributes that should be applied to it upon creation (refer to TScSFTPFileAttributes).

See Also

OnError

OnSuccess

TScSFTPFileAttributes

5.104.3.12 OpenDirectory

```
function OpenDirectory(const Path: string): TScSFTPFileHandle;
```

Description

Call the **OpenDirectory** method to open an existing directory on the server for enumeration.

If NonBlocking is False,**OpenDirectory** returns the directory handle. Otherwise it returns nil, and to obtain the directory handle the OnOpenFile event should be processed. If the server returns an error, the OnError event is generated.

The obtained directory handle may be used in other operations, for example, in ReadDirectory.

When enumeration is complete, the handle must be closed using the CloseHandle method.

Parameters:

- `Path` - the path name of the directory to be listed (without any trailing slash). If `Path` does not refer to a directory, the server returns an error.

See Also

CloseHandle

OnError

OnOpenFile

OpenFile

5.104.3.14 OpenFile

```
function OpenFile(const FileName: string; Modes: TScSFTPFileOpenModes;
Attributes: TScSFTPFileAttributes = nil): TScSFTPFileHandle; overload;
function OpenFile(const FileName: string; Mode: TScSFTPFileOpenMode;
Flags: TScSFTPFileOpenFlags = []; BlockModes: TScSFTPBlockModes = [];
Access: TScSFTPDesiredAccess = []; Attributes: TScSFTPFileAttributes =
nil): TScSFTPFileHandle; overload;
```

Description

Call the **OpenFile** method to open or create a remote file.

If the NonBlocking property is set to False, **OpenFile** returns file handle. Otherwise it returns nil, and to receive the file handle the OnOpenFile event should be processed. If the server returns an error, the OnError event is generated.

The file handle that was received may be used in other operations like ReadFile, WriteFile etc. After the work with the file handle was finished, you should close it by calling the CloseHandle method.

Parameters:

- `FileName` - the name of the file that is being opened. If `FileName` is the name of a directory, an error will be raised.
- `Modes` - flags for the file opening (refer to TScSFTPFileOpenModes).
- `Mode` - the mode of the file opening (refer to TScSFTPFileOpenMode).
- `Flags` - the set of flags for the file opening (refer to TScSFTPFileOpenFlags).
- `BlockModes` - the blocking mode of the file that is being opened (refer to TScSFTPBlockModes).
- `Access` - the rights for the file access that are a combination of values of the ace-mask flags (refer to TScSFTPDesiredAccess). If the server cannot grant the access desired, it returns the SSH_FX_PERMISSION_DENIED error.
- `Attributes` - specifies the initial attributes for the file. Parameter is ignored if an existing file is opened.

Note: It is preferable to use the first overload method with the version 4 of the SFTP protocol or lower, and the second - with the version 5 or higher (that's why it has more parameters, and, as a result, has enhanced functionality).

See Also

Block

CloseHandle

NonBlocking

OnError

OnOpenFile

OpenDirectory

ReadFile

WriteFile

TScSFTPServerProperties.Newline

5.104.3.15 QueryAvailableSpace

```
procedure QueryAvailableSpace(const Path: string; ReplyExtension:
TScSpaceAvailableReplyExtension = nil);
```

Description

Call the **QueryAvailableSpace** method to learn the amount of available space for an arbitrary path.

If the NonBlocking property is False, **QueryAvailableSpace** returns control after receiving an answer from the server and writing the result to the `ReplyExtension` object. Otherwise the result is written to `ReplyExtension` on executing the OnReplySpaceAvailable event. If the server returns an error, the OnError event is generated.

Note: this request is not supported by all SFTP servers.

Parameters:

- `Path` - the path for which the available space should be reported.
- `ReplyExtension` - an object to which the data about available space will be written. If this parameter is set to nil or not set at all, the OnReplySpaceAvailable event should be processed in order to get the result. If the object is specified, it will be returned in the OnReplySpaceAvailable event handler.

See Also

OnError

OnReplySpaceAvailable

5.104.3.16 QueryUserHomeDirectory

```
function QueryUserHomeDirectory(const Username: string): string;
```

Description

Call the **QueryUserHomeDirectory** method to request user home directory for the specified `Username`. An empty string implies the current user.

Many users are used typing '~' as an alias for their home directory, or ~username as an alias for another user's home directory. To support this feature use this method.

If the NonBlocking property is False, **QueryUserHomeDirectory** returns user home directory. Otherwise it returns an empty string, and in order to get the directory the OnFileName event should be processed. If the server returns an error, the OnError event is generated.

Note: this request is not supported by all SFTP servers.

See Also

OnError

OnFileName

5.104.3.17 ReadDirectory

```
procedure ReadDirectory(const Handle: TScSFTPFileHandle);
```

Description

Call the **ReadDirectory** method to retrieve a directory listing. In order to obtain a complete directory listing, the client must call this method until the EOF property is set to True. For every received file or directory name the OnDirectoryList event is generated. If the server returns an error, the OnError event is generated.

Parameters:

- `Handle` - a handle previously returned in the response to OpenDirectory. If `Handle` is an ordinary file handle returned by OpenFile, the server returns error.

See Also

EOF

OnDirectoryList

OnError

OpenDirectory

ReadDirectoryToList

5.104.3.18 ReadDirectoryToList

```
procedure ReadDirectoryToList(const Handle: TScSFTPFileHandle; List:
TCRObjectList);
```

Description

Call the **ReadDirectoryToList** method to retrieve a directory listing. For every received file or directory name the TScSFTPFileInfo object is created and added to the specified `List`. If the server returns an error, the OnError event is generated.

Unlike the ReadDirectory method, to get a complete directory listing, the **ReadDirectoryToList** method must be called only once; the OnDirectoryList event is not generated.

Parameters:

- `Handle` - a handle returned previously in the response to OpenDirectory. If `Handle` is an ordinary file handle returned by OpenFile, the server returns an error.
- `List` - the TScSFTPFileInfo objects, which describe every received file or directory name, will be assigned to this object. If `List` is nil, an exception will be raised.

See Also

OnError

OpenDirectory

ReadDirectory

5.104.3.19 ReadFile

```
function ReadFile(const Handle: TScSFTPFileHandle; FileOffset: Int64; var
Buffer; Count: integer): integer; overload;
function ReadFile(const Handle: TScSFTPFileHandle; FileOffset: Int64; var
Buffer: TBytes; Offset, Count: integer): integer; overload;
```

Description

Call the **ReadFile** method to read remote file data.

If the NonBlocking property is set to False **ReadFile** returns the amount of data read. Otherwise it returns 0 and the data can be read from the buffer on processing the OnData event. OnData may occur several times for a single call of **ReadFile**, if the amount of requested data exceeds the possible amount of data the server can return during one request (refer to the ReadBlockSize property). If the servers returns an error, the OnError event is generated.

ReadFile sets the EOF property to True if the end of file was reached.

Parameters:

- `Handle` - a handle previously returned in the response to OpenFile.
- `FileOffset` - the offset in bytes relative to the beginning of the file that the read starts at. This parameter is ignored if TEXT MODE was specified during the open.
- `Buffer` - the buffer to which the data will be read. If the buffer is specified, it will be returned by the OnData event handler. If NonBlocking is True, the parameter can have the nil value.
- `Offset` - the position in the buffer from which to start writing the data.
- `Count` - the maximum number of bytes to read.

See Also

OnData

OnError

OpenFile

ReadBlockSize

5.104.3.20 ReadSymbolicLink

```
function ReadSymbolicLink(const Path: string): string;
```

Description

Call the **ReadSymbolicLink** method to read the target of a symbolic link. `Path` specifies the path name of the symbolic link to be read.

If NonBlocking is False, **ReadSymbolicLink** returns the target of the link. Otherwise it returns an empty string, and in order to obtain the result, the OnFileName event should be processed. If the server returns an error, the OnError event is generated.

See Also

OnError

OnFileName

5.104.3.21 RemoveDirectory

```
procedure RemoveDirectory(const Path: string);
```

Description

Call the **RemoveDirectory** method to remove a directory. The `Path` parameter specifies the directory to be removed. This request cannot be used to remove a file.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

See Also

OnError

OnSuccess

5.104.3.21 RemoveFile

```
procedure RemoveFile(const FileName: string);
```

Description

Call the **RemoveFile** method to remove a file. `FileName` is the name of the file to be removed. This request cannot be used to remove directories.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

See Also

OnError

OnSuccess

5.104.3.22 RenameFile

```
procedure RenameFile(const OldPath, NewPath: string; Flags:
TScSFTPRenameFlags = []);
```

Description

Call the **RenameFile** method to rename or move file or directory.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `OldPath` - the name of an existing file or directory.
- `NewPath` - the new name for the file or directory.
- `Flags` - the renaming parameters. This parameter is supported only since the version 5 of the SFTP protocol.

See Also

OnError

OnSuccess

5.104.3.23 RequestExtension

```
procedure RequestExtension(const ExtName: string; const ExtData: TBytes;
ReplyExtension: TScSFTPExtension = nil); overload;
```

```
procedure RequestExtension(Extension: TScSFTPExtension; ReplyExtension:
TScSFTPExtension = nil); overload;
```

Description

Call the **RequestExtension** method to send an extended request to the server.

Extensions allow clients to query the server for additional information which may not be widely supported, but all the same can be implemented by some servers.

If the NonBlocking property is False, **RequestExtension** returns control after receiving an answer from the server and writing it to the `ReplyExtension` object. Otherwise the result is written to the `ReplyExtension` object on executing the OnReplyExtension event. If the server returns an error, the OnError event is generated.

Note: this option is supported starting from the version 3 of the SFTP protocol.

Parameters:

- `ExtName` - string that holds the extension name.
- `ExtData` - the extension data, that is specified by the specific extension.
- `Extension` - the object the Name and Data properties of which specify the extension parameters.
- `ReplyExtension` - the object to which the replied data will be written. If this parameter is set to nil or is not set at all, then the OnReplyExtension event should be processed.If the object is specified, it will be returned in the OnReplyExtension event handler.

See Also

OnError

OnReplyExtension

TScSFTPExtension

5.104.3.2 RetrieveAbsolutePath

```
function RetrieveAbsolutePath(const Path: string; Control:
TScSFTPRealpathControl = rcNoCheck; ComposePath: TStringList = nil):
string;
```

Description

Call the **RetrieveAbsolutePath** method to have the server canonize any given path name to an absolute path. This is useful for converting path names containing ".." components or relative path names without a leading slash into absolute paths.

If NonBlocking is False, **RetrieveAbsolutePath** returns absolute path. Otherwise it returns an empty string, and in order to obtain absolute path the OnFileName event should be processed. If the server returns an error, the OnError event is generated.

Parameters:

- `Path` - original path which the client wants to be resolved into an absolute canonical path.
- `Control` - the parameters of identifying the absolute path. This parameter is supported starting with the version 6 of the SFTP protocol.
- `ComposePath` - the client may specify multiple elements, in which case the server should build the resulting path by applying each compose path to the accumulated result until all elements have been applied. This parameter is supported starting with the version 6 of the SFTP protocol.

See Also

OnError

OnFileName

5.104.3.26 RetrieveAttributes

```
procedure RetrieveAttributes(Attrs: TScSFTPFileAttributes; const Path:
string; SymbolicLinks: boolean = False; const Flags: TScSFTPAttributes =
[]);
```

Description

Call the **RetrieveAttributes** method to retrieve the attributes for a named file.

If NonBlocking is False, then **RetrieveAttributes** returns control after receiving attributes from the server and writing them to the `Attrs` object. Otherwise the attributes are set to `Attrs` on executing the OnFileAttributes event. If the server returns an error, the OnError event is generated.

Parameters:

- `Attrs` - an object to which the attributes of the requested file will be written.
- `Path` - specifies the file system object for which attributes should be returned.
- `SymbolicLinks` - specifies if the server follows symbolic links.
- `Flags` - specifies the attribute flags in which the client has particular interest. This parameter is supported starting with the version 4 of the SFTP protocol.

See Also

OnError

OnFileAttributes

RetrieveAttributesByHandle

TScSFTPFileAttributes

5.104.3.27 RetrieveAttributesByHandle

```
procedure RetrieveAttributesByHandle(Attrs: TScSFTPFileAttributes; const
Handle: TScSFTPFileHandle; const Flags: TScSFTPAttributes = []);
```

Description

Call the **RetrieveAttributesByHandle** method to retrieve the attributes for a previously opened file.

If NonBlocking is False, then **RetrieveAttributesByHandle** returns control after receiving attributes from the server and writing them to the `Attrs` object. Otherwise the attributes are set to `Attrs` on executing the OnFileAttributes event. If the server returns an error, the OnError event is generated.

Parameters:

- `Attrs` - an object to which the attributes of the requested file will be written.
- `Handle` - a handle previously returned in the response to OpenFile or OpenDirectory.
- `Flags` - specifies the attribute flags in which the client has particular interest. This parameter is supported starting with the version 4 of the SFTP protocol.

See Also

OnError

OnFileAttributes

RetrieveAttributes

TScSFTPFileAttributes

5.104.3.2 SetAttributes

```
procedure SetAttributes(const Path: string; Attributes:
TScSFTPFileAttributes);
```

Description

Call the **SetAttributes** method to set the attributes for a named file.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `Path` - the file system object (e.g. file or directory) whose attributes are to be modified. If this object does not exist, or the user does not have sufficient access to write the attributes, the server will return an error.
- `Attributes` - object, that specifies the modified attributes to be applied.

See Also

OnError

OnSuccess

OpenDirectory

OpenFile

SetAttributesByHandle

5.104.3.2 SetAttributesByHandle

```
procedure SetAttributesByHandle(const Handle: TScSFTPFileHandle;
Attributes: TScSFTPFileAttributes);
```

Description

Call the **SetAttributesByHandle** method to set the attributes for a previously opened file.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `Handle` - a handle previously returned in the response to OpenFile or OpenDirectory.
- `Attributes` - object, that specifies the modified attributes to be applied.

See Also

OnError

OnSuccess

OpenDirectory

OpenFile

SetAttributes

5.104.3.3 TextSeek

```
function TextSeek(const Handle: TScSFTPFileHandle; LineNumber: Int64):
Boolean;
```

Description

Call the **TextSeek** method to support seek on text file.

If the NonBlocking property is False, **TextSeek** returns True, if the requested line was found, and False, if the end of file was reached.

If the NonBlocking property is True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Note: this request is not supported by all SFTP servers.

Parameters:

- `Handle` - a handle returned by the OpenFile method.
- `LineNumber` - the index of the line number to look for, where byte 0 in the file is the line number 0, and the byte directly following the first newline sequence in the file is the line number 1 and so on.

See Also

OnError

OnSuccess

5.104.3.33 UnBlock

```
procedure UnBlock(const Handle: TScSFTPFileHandle; Offset, Count: Int64);
```

Description

Call the **UnBlock** method to remove a previously acquired byte-range lock on the specified handle.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Note: This operation is supported starting with the version 6 of the SFTP protocol.

Parameters:

- `Handle` - a handle on which a Block request has previously been.
- `Offset` - the beginning of the byte-range to lock.
- `Count` - the number of bytes in the range to lock. The special value 0 means lock from `Offset` to the end of the file.

See also

Block

OnError

OnSuccess

5.104.3.34 UploadFile

```
procedure UploadFile(const Source, Destination: string; Overwrite:
Boolean; FileOffset: Int64 = 0);
```

Description

Call the **UploadFile** method to copy a file from the local machine to the remote one.

To create a resulting file on the server with the required attributes (for example, the attributes should

correspond to the ones that the local file has) process the OnSetRemoteFileAttributes event.

If the NonBlocking property is set to True, then control is returned at once and you can learn the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `Source` - holds the initial path to the file that is being copied.
- `Destination` - holds the destination path to copy the file to.
- `Overwrite` - specifies whether to overwrite the file with the same name if it exists.
- `FileOffset` - the offset in bytes relative to the beginning of the file that the uploading started at.

See Also

OnError

OnSuccess

OnSetRemoteFileAttributes

5.104.3.3 UploadFromStream

```
procedure UploadFromStream(Source: TStream; const DestinationName:
string; Overwrite: boolean; FileOffset: Int64 = 0);
```

Description

Call the **UploadFromStream** method to copy a file from the local machine to the remote one.

To create a resulting file on the server with the required attributes (for example, the attributes should correspond to the ones that the local file has) process the OnSetRemoteFileAttributes event.

If the NonBlocking property is set to True, then the control is returned immediately and you can see the result of the operation execution by handling the OnSuccess and OnError events.

Parameters:

- `Source` - holds the source data stream that is being copied.
- `Destination` - holds the destination path to copy the file to.
- `Overwrite` - specifies whether to overwrite the file with the same name if it exists.
- `FileOffset` - the offset in bytes relative to the beginning of the file that the uploading started at.

See Also

OnError

OnSuccess

OnSetRemoteFileAttributes

5.104.3.3 WriteFile

```
procedure WriteFile(const Handle: TScSFTPFileHandle; FileOffset: Int64;
const Buffer; Count: integer); overload;
procedure WriteFile(const Handle: TScSFTPFileHandle; FileOffset: Int64;
const Buffer: TBytes; Offset, Count: integer); overload;
```

Description

Call the **WriteFile** method to write data to the remote file.

If the server returns an error, the OnError event is generated.

Parameters:

- `Handle` - a handle previously returned as a response to OpenFile.
- `FileOffset` - the offset in bytes relative to the beginning of the file that the writing started at. This field is ignored if TEXT MODE was specified during the opening.
- `Buffer` - the sequence of bytes that should be written to the file.
- `Offset` - the position in the buffer from which to start reading the data.
- `Count` - the number of bytes to write.

See Also

OpenFile

5.104.4 Events

5.104.4.1 AfterWriteData

```
type
  TScSFTPDataEvent = procedure(Sender: TObject; const FileName: string;
const Handle: TScSFTPFileHandle; FileOffset: Int64; const Buffer: TBytes;
Offset, Count: Integer; EOF: Boolean) of object;

property AfterWriteData: TScSFTPDataEvent;
```

Description

The **AfterWriteData** event occurs after data is written to a remote file when executing the WriteFile method. TScSFTPClient sends data by pieces of WriteBlockSize size on uploading file, therefore the **AfterWriteData** event may occur several times during one call of this method.

Parameters:

- `Sender` - the object that raised the event.

- FileName - the name of the file in which the data is being written.
- Handle - handle of the file, in which data was written.
- FileOffset - the offset in bytes relative to the beginning of the file that the writing started at.
- Buffer - the buffer which holds the data write to the file. It can hold the nil value if EOF is True.
- Offset - the position in the buffer that indicates the beginning of the written data.
- Count - the amount of data sent in bytes.
- EOF - indicates that the end of file was reached. If EOF value is True, the end of file was reached. Otherwise it was not reached.

Note, when using the **AfterWriteData** event handler, the onSuccess event is not triggered on Operation = opWritingFile.

5.104.4.2 BeforeWriteData

```
type
  TScSFTPDataChangeEvent = procedure(Sender: TObject; const FileName:
string; const Handle: TScSFTPFileHandle; FileOffset: Int64; var Buffer:
TBytes; var Offset, Count: Integer; EOF: Boolean) of object;

property BeforeWriteData: TScSFTPDataChangeEvent;
```

Description

The **BeforeWriteData** event occurs before data has been written to a remote file when executing the WriteFile method. TScSFTPClient sends data by pieces of WriteBlockSize size on uploading file, therefore the **BeforeWriteData** event may occur several times during one call of this method.

Parameters:

- Sender - the object that raised the event.
- FileName - the name of the file in which the data is being written.
- Handle - handle of the file, in which data was written.
- FileOffset - the offset in bytes relative to the beginning of the file that the writing started at.
- Buffer - the buffer which holds the data write to the file. It can hold the nil value if EOF is True.
- Offset - the position in the buffer that indicates the beginning of the written data.
- Count - the amount of data sent in bytes.
- EOF - indicates that the end of file was reached. If EOF value is True, the end of file was reached. Otherwise it was not reached.

5.104.4.3 OnConnect

```
property OnConnect: TNotifyEvent;
```

Description

The **OnConnect** event occurs before establishing secure logical connection through an SSH tunnel.

5.104.4.4 OnCreateLocalFile

```
type
  TScSFTPCreateLocalFileEvent = procedure(Sender: TObject; const
  LocalFileName, RemoteFileName: string; Attrs: TScSFTPFileAttributes; var
  Handle: THandle) of object;

property OnCreateLocalFile: TScSFTPCreateLocalFileEvent;
```

Description

The **OnCreateLocalFile** event occurs when copying a file from remote machine to the local during the DownloadFile method call. When processing **OnCreateLocalFile** event, you should create a file and set required attributes for it. The handle of the created file should be specified in the `Handle` parameter.

Parameters:

- `Sender` - the object that raised the event.
- `LocalFileName` - the local path to copy the file to.
- `RemoteFileName` - the path to the file (that should be copied) on the server.
- `Attrs` - the object that holds the attributes of the file that is being copied (the original file).
- `Handle` - set the handle of the created file as a value of this variable.

See Also

DownloadFile

5.104.4.5 OnData

```
type
  TScSFTPDataEvent = procedure(Sender: TObject; const FileName: string;
  const Handle: TScSFTPFileHandle; FileOffset: Int64; const Buffer:
  TBytes; Offset, Count: Integer; EOF: Boolean) of object;

property OnData: TScSFTPDataEvent;
```

Description

The **OnData** event occurs when reading data from a remote file when executing the ReadFile method.

This event may occur several times during one call of this method.

Parameters:

- `Sender` - the object that raised the event.
- `FileName` - the name of the file from which the data is being read.
- `Handle` - the file handle for the file that was sent to the ReadFile method.
- `FileOffset` - the offset in bytes relative to the beginning of the file that the read starts at.
- `Buffer` - the buffer which holds the data read from the file. It can hold the nil value if `EOF` is True.
- `Offset` - the position in the buffer that indicates the beginning of the written data.
- `Count` - the amount of data received in bytes.
- `EOF` - indicates that the end of file was reached. If `EOF` value is True, the end of file was reached. Otherwise it was not reached.

See Also

ReadFile

5.104.4.6 OnDirectoryList

```
type
  TScSFTPDirectoryListEvent = procedure(Sender: TObject; const Path:
  string; const Handle: TScSFTPFileHandle; FileInfo: TScSFTPFileInfo; EOF:
  Boolean) of object;

property OnDirectoryList: TScSFTPDirectoryListEvent;
```

Description

The **OnDirectoryList** event occurs when receiving directory listing during the ReadDirectory method call. The **OnDirectoryList** event is generated for every name of a file or directory that was received.

Parameters:

- `Sender` - the object that raised the event.
- `Path` - the path from which the directory listing is retrieved.
- `Handle` - a handle of this directory, that was sent to the ReadDirectory method.
- `FileInfo` - the object that holds information on the file. Can be of the nil value if `EOF` is True.
- `EOF` - determines if there are more files or directories in the specified directory. If `EOF` value is True, this file is the last file in this directory and it does not contain any other files. Otherwise there are more files in the directory.

See Also

TScSFTPFileInfo

ReadDirectory

5.104.4.7 OnDisconnect

```
property OnDisconnect: TNotifyEvent;
```

Description

The **OnDisconnect** event occurs after the logical connection through an SSH server is closed.

5.104.4.8 OnError

```
type
  TScSFTPErrorEvent = procedure(Sender: TObject; Operation:
  TScSFTPOperation; const Filename: string; const Handle:
  TScSFTPFileHandle; ErrorCode: integer; const ErrorMessage: string; var
  Fail: Boolean) of object;

property OnError: TScSFTPErrorEven;
```

Description

The **OnError** event occurs when the server returns an error when executing some operation.

Parameters:

- `Sender` - the object that raised the event.
- `Operation` - determines during which operation the error occurred.
- `Filename` - the name of the file or directory with which the operation was performed.
- `Handle` - the handle of the file with which the operation was executed. May be of the nil value.
- `ErrorCode` - holds the error code. To learn the error codes, refer to the EScSFTPError topic.
- `ErrorMessage` - holds the readable description of the error.
- `Fail` - if the NonBlocking property is False, then set the `Fail` parameter to False to prevent raising an exception, and set this parameter to True to raise the EScSFTPError exception. If the NonBlocking property is True, this parameter is ignored.

See Also

TScSFTPOperation

EScSFTPError

5.104.4.9 OnFileAttributes

```
type
  TScSFTPFileAttributesEvent = procedure(Sender: TObject; const FileName:
  string; const Handle: TScSFTPFileHandle; FileAttributes:
  TScSFTPFileAttributes) of object;

property OnFileAttributes: TScSFTPFileAttributesEvent;
```

Description

The **OnFileAttributes** event occurs when requesting file attributes during the RetrieveAttributes or RetrieveAttributesByHandle method call.

Parameters:

- `Sender` - the object that raised the event.
- `FileName` - the name of the file for which attributes are returned.
- `Handle` - a file handle for this file. `Handle` is set only if the RetrieveAttributesByHandle method is called. Otherwise it has nil value.
- `FileAttributes` - the object that holds the attributes of the requested file.

See Also

TScSFTPFileAttributes

RetrieveAttributes

RetrieveAttributesByHandle

5.104.4.10 OnFileName

```
type
  TScSFTPFileNameEvent = procedure(Sender: TObject; const SrcFileName,
  DestFileName: string) of object;

property OnFileName: TScSFTPFileNameEvent;
```

Description

The **OnFileName** event occurs during the call to the ReadSymbolicLink, RetrieveAbsolutePath, or QueryUserHomeDirectory method.

If the ReadSymbolicLink method was called, the `SrcFileName` parameter holds the name of the symbolic link, and the `DestFileName` parameter holds the target of this symbolic link.

During the RetrieveAbsolutePath method call `SrcFileName` holds the original path and `DestFileName` holds the absolute path.

During the QueryUserHomeDirectory method call `SrcFileName` holds the specified user name, and `DestFileName` - home directory for this user name.

See Also

ReadSymbolicLink

RetrieveAbsolutePath

QueryUserHomeDirectory

5.104.4.11 OnOpenFile

type

```
TScSFTPOpenFileEvent = procedure(Sender: TObject; const FileName:
string; const Handle: TScSFTPFileHandle) of object;
```

```
property OnOpenFile: TScSFTPOpenFileEvent;
```

Description

The **OnOpenFile** event occurs when opening file or directory when executing OpenFile or OpenDirectory methods.

Parameters:

- `Sender` - the object that raised the event.
- `FileName` - the name of the file or directory that is being opened.
- `Handle` - the file handle that was received from the server for the file or directory that is being opened. This handle may be used in other operations like ReadFile, WriteFile etc.

See Also

OpenFile

OpenDirectory

5.104.4.12 OnReplyCheckFile

type

```
TScSFTPReplyCheckFileEvent = procedure(Sender: TObject; const FileName:
string; const Handle: TScSFTPFileHandle; CheckFileReplyExtension:
TScCheckFileReplyExtension) of object;
```

```
property OnReplyCheckFile: TScSFTPReplyCheckFileEvent;
```

Description

The **OnReplyCheckFile** event occurs when requesting file check during the CheckFile or CheckFileByHandle method call.

Parameters:

- `Sender` - the object that raised the event.
- `FileName` - the path to the file to check.
- `Handle` - a file hanle for this file. This parameter is set only when calling the CheckFileByHandle method. Otherwise its value is nil.
- `CheckFileReplyExtension` - the object that holds received computed hashes.

See Also

TScCheckFileReplyExtension

CheckFile

CheckFileByHandle

5.104.4.11 OnReplyExtension

```
type
  TScSFTPReplyExtensionEvent = procedure(Sender: TObject; const ExtName:
  string; Extension: TScSFTPExtension) of object;

property OnReplyExtension: TScSFTPReplyExtensionEvent;
```

Description

The **OnReplyExtension** event occurs when the server answers the extended request during the RequestExtension method call.

Parameters:

- `Sender` - the object that raised the event.
- `ExtName` - holds the extension name as string.
- `Extension` - the object to which the replied data is written.

See Also

TScSFTPExtension

RequestExtension

5.104.4.12 OnReplySpaceAvailable

```
type
  TScSFTPReplySpaceAvailableEvent = procedure(Sender: TObject; const Path:
```

```
  string; SpaceAvailableReplyExtension: TScSpaceAvailableReplyExtension)
  of object;

property OnReplySpaceAvailable: TScSFTPReplySpaceAvailableEvent;
```

Description

The **OnReplySpaceAvailable** event occurs when requesting available space for an arbitrary path during the QueryAvailableSpace method call.

Parameters:

- `Sender` - the object that raised the event.
- `Path` - the path for which the available space was requested.
- `SpaceAvailableReplyExtension` - the object that holds data about the available space.

See Also

TScSpaceAvailableReplyExtension

5.104.4.1 OnSetRemoteFileAttributes

```
type
  TScSFTPSetRemoteFileAttributesEvent = procedure(Sender: TObject; const
  LocalFileName, RemoteFileName: string; Attrs: TScSFTPFileAttributes) of
  object;

property OnSetRemoteFileAttributes: TScSFTPSetRemoteFileAttributesEvent;
```

Description

The **OnSetRemoteFileAttributes** event occurs when copying a file from local machine to remote during the UploadFile method call. On processing this event you can set the attributes for the remote file using the `Attrs` object.

Parameters:

- `Sender` - the object that raised the event.
- `LocalFileName` - the path to the file that is being copied on the local machine.
- `RemoteFileName` - the path on the server where the file will be copied.
- `Attrs` - the object where the attributes of the file on the server should be specified (the attributes of the local file may be used).

See Also

UploadFile

5.104.4.16 OnSuccess

```
type
  TScSFTPSuccessEvent = procedure(Sender: TObject; Operation:
  TScSFTPOperation; const FileName: string; const Handle:
  TScSFTPFileHandle; const Message: string) of object;

property OnSuccess: TScSFTPSuccessEvent;
```

Description

The **OnSuccess** event occurs after successful execution of an operation that does not return any data (for example, RemoveFile).

Parameters:

- `Sender` - the object that raised the event.
- `Operation` - determines which operation was executed.
- `FileName` - the name of the file or directory with which the operation was performed.
- `Handle` - the handle handle of the file with which the operation was performed. May be of the nil value.
- `Message` - holds the message about the result of the operation execution.

See Also

TScSFTPOperation

5.104.4.17 OnVersionSelect

```
type
  TScSFTPVersionSelectEvent = procedure(Sender: TObject; const Versions:
  TScSFTPVersions; var Version: TScSFTPVersion) of object;

property OnVersionSelect: TScSFTPVersionSelectEvent;
```

Description

The **OnVersionSelect** event occurs when establishing a connection to the SFTP server on calling the Initialize method. If the server supports higher versions, then it was specified during the initialization, it can notify the client about this and **OnVersionSelect** will be initiated.

Parameters:

- `Sender` - the object that raised the event.
- `Versions` - the set of SFTP protocols that are supported by the SFTP server.

- `Version` - the version of the SFTP protocol that is used. If client wants to change this version, the value of this parameter may be changed to one of the supported versions of the SFTP protocol.

See Also

Initialize

ServerVersion

Version

5.105 TScSFTPServerProperties

5.105.1 Description

Unit

ScSFTPClient

Description

The **TScSFTPServerProperties** class holds detailed information about the SFTP server that the server can send when establishing a connection.

This class consists of the pairs of properties. One property holds information about the specific extension, and the other is a boolean property that specifies if this property was received from the server. For example: FilenameCharset and FilenameCharsetAvailable. If the FilenameCharsetAvailable property is True, then the value of the FilenameCharset property may be used. Otherwise FilenameCharset is not initialized.

See also

TScSFTPClient.ServerProperties

5.105.2 Properties

5.105.2.1 FilenameCharset

```
property FilenameCharset: string;
```

Description

The **FilenameCharset** property contains the charset of file names used by server.

If server sends information about filename charset, then filenames can be received in the specified encoding. If the server does not send this information, then the names of files will be converted and will be received in the UTF-8 encoding. If the server sent the filename charset and you want to receive data in the specified encoding, you should send TScFilenameTranslationControlExtension with the DoTranslate property set to False. If you need to receive data in the UTF-8 encoding, you may send TScFilenameTranslationControlExtension with the DoTranslate property set to True.

To check if the information about filename charset was received from the server use the FilenameCharsetAvailable property.

See Also

FilenameCharsetAvailable

TScFilenameTranslationControlExtension

5.105.2.2 FilenameCharsetAvailable

```
property FilenameCharsetAvailable: boolean;
```

Description

Use the **FilenameCharsetAvailable** property to check if the information about filename charset was received from the server.

If **FilenameCharsetAvailable** is True, then the FilenameCharset property was set by the server. Otherwise FilenameCharset is not initialized.

See Also

FilenameCharset

5.105.2.3 Newline

```
property Newline: string;
```

Description

The **Newline** property contains newline sequences used on the server. Newline sequences are used in order to process text files in a cross platform compatible way correctly.

To check if data about newline sequences was received from the server, use the NewlineAvailable property.

See Also

NewlineAvailable

5.105.2.4 NewlineAvailable

```
property NewlineAvailable: boolean;
```

Description

Use the **NewlineAvailable** property to check if information about newline sequences was received from the server.

If the **NewlineAvailable** is True, then the Newline property is set by the server. Otherwise Newline is not initialized.

See Also

Newline

5.105.2.5 SupportedAcls

```
property SupportedAcls: TScSFTPSupportedAclExtension;
```

Description

The **SupportedAcls** property holds the supported by server capabilities of the ACL attribute.

To check if this extension was receied from the server use the SupportedAclsAvailable property.

See Also

SupportedAclsAvailable

5.105.2.6 SupportedAclsAvailable

```
property SupportedAclsAvailable: boolean;
```

Description

Use the **SupportedAclsAvailable** property to check if the information about supported capabilities of the ACL attribute was received from the server.

If the **SupportedAclsAvailable** property is True, then the SupportedAcls property is set by the server. Otherwise SupportedAcls is not initialized.

See Also

SupportedAcls

5.105.2.7 SupportedExtension

```
property SupportedExtension: TScSFTPSupportedExtension;
```

Description

The **SupportedExtension** contains features supported by server.

To check if this extension was received from the server, use the SupportedExtensionAvailable property.

See Also

SupportedExtensionAvailable

5.105.2.8 SupportedExtensionAvailable

```
property SupportedExtensionAvailable: boolean;
```

Description

Use the **SupportedExtensionAvailable** property to check if the information about supported features was received from the server.

If the **SupportedExtensionAvailable** property is True, then the SupportedExtension property was set by the server. Otherwise SupportedExtension is not initialized.

See Also

SupportedExtension

5.105.2.9 Vendor

```
property Vendor: TScSFTPVendorExtension;
```

Description

The **Vendor** property holds detailed information about the version and build of the SFTP server.

To check if this extension was received from the server use the VendorAvailable property.

See Also

VendorAvailable

5.105.2.10 VendorAvailable

```
property VendorAvailable: boolean;
```

Description

Use the **VendorAvailable** property to check if the vendor extension was received from the server.

If the **VendorAvailable** property is True, then the Vendor property is set by the server. Otherwise Vendor is not initialized.

See Also

Vendor

5.105.2.11 Versions

```
property Versions: TScSFTPVersionsExtension;
```

Description

The **Versions** property contains a list of the SFTP protocol versions supported by the server.

To check if this information was received from the server use the VersionsAvailable property.

See Also

VersionsAvailable

5.105.2.1 VersionsAvailable

```
property VersionsAvailable: boolean;
```

Description

Use the **VersionsAvailable** property to check if the list of the supported SFTP protocol versions was received from the server.

If the **VersionsAvailable** property is True, then the Versions property was set by the server. Otherwise Versions is not initialized.

See Also

Versions

5.106 TScSFTPACEItem

5.106.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPACEItem** class is a descendant of the TScCollectionItem class.

ACL (Access Control List) attribute (taken from NFS Version 4 Protocol [RFC3010]) is an array of access control entries (ACE). There are various access control entry types. The server is able to communicate which ACE types are supported by returning the appropriate value within the aclsupport attribute.

TScSFTPACEItem holds the parameters of the ACE attribute.

See Also

TScSFTPACEs

5.106.2 Properties

5.106.2.1 AceFlags

```
type
  TScSFTPAceFlag = (afFileInherit, afDirectoryInherit,
afNo_PropagateInherit, afInheritOnly, afSuccessfulAccess, afFailedAccess,
afIdentifierGroup);
  TScSFTPAceFlags = set of TScSFTPAceFlag;

property AceFlags: TScSFTPAceFlags;
```

Description

The **AceFlags** property holds a set of the ACE flags (taken from NFS Version 4 Protocol [RFC3010]).

5.106.2.2 AceMask

```
property AceMask: TScSFTPAceMask;
```

Description

The **AceMask** property holds a set of the ACE masks (taken from NFS Version 4 Protocol [RFC3010]).

5.106.2.3 AceType

```
type
  TScSFTPAceType = (atAccessAllowed, atAccessDenied, atSystemAudit,
atSystemAlarm);

property AceType: TScSFTPAceType;
```

Description

The **AceType** property holds the ACE type (taken from NFS Version 4 Protocol [RFC3010]).

5.106.2.4 Who

```
property Who: string;
```

Description

The **Who** property holds a string of the form described in the Owner and Group properties. Also, there are several identifiers that need to be understood universally. Some of these identifiers cannot be understood when a client accesses the server, but have meaning when a local process accesses the file. The ability to display and modify these permissions is permitted over SFTP.

Value	Meaning
OWNER	The owner of the file.
GROUP	The group associated with the file.
EVERYONE	The world.
INTERACTIVE	Accessed from an interactive terminal.
NETWORK	Accessed via the network.
DIALUP	Accessed as a dialup user to the server.
BATCH	Accessed from a batch job.
ANONYMOUS	Accessed without any authentication.
AUTHENTICATED	Any authenticated user (opposite of ANONYMOUS).
SERVICE	Access from a system service.

5.107 TScSFTPACEs

5.107.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPACEs** class is a descendant of the TScCollection class, and it is a container for TScSFTPACEItem objects.

ACL (Access Control List) attribute (taken from NFS Version 4 Protocol [RFC3010]) is an array of access control entries (ACE).

TScSFTPACEs keeps a list of the access control entries (ACE) which are used in the SFTP protocol, and represents them in the SFTP format.

See Also

TScSFTPACEItem

5.108 TScSFTPCustomExtension

5.108.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPCustomExtension** class is a base abstract class for other SFTP protocol extensions classes like TScSFTPExtension, TScCheckFileReplyExtension, TScSFTPSupportedExtension etc.

Extensions make it possible for clients to query the SFTP server for additional information which may not be widely supported, but may be implemented by some servers.

See Also

TScSFTPExtension

5.108.2 Properties

5.108.2.1 Name

```
property Name: string;
```

Description

The **Name** property holds the extension name.

5.109 TScSFTPExtension

5.109.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPExtension** class is used to implement users' extensions.

See Also

TScSFTPCustomExtension

5.109.2 Properties

5.109.2.1 Data

```
property Data: TBytes;
```

Description

The **Data** property holds the extension data defined by the specific extension.

5.110 TScSFTPFileAttributes

5.110.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPFileAttributes** class is defined for encoding file attributes. The same encoding is used both when returning file attributes from the server and when sending file attributes to the server. When sending it to the server, the properties specify which attributes are included, and the server will use the default values for the remaining attributes (or will not modify the values of remaining attributes). When receiving attributes from the server, the properties specify which attributes are included in the returned data. The server normally returns all attributes it knows about.

The ValidAttributes property specifies the attributes values with meaning.

5.110.2 Properties

5.110.2.1 AccessTime

```
property AccessTime: TDateTime;
```

Description

The **AccessTime** property contains the time of the last access to the file. Many operating systems either don't have this field, only optionally maintain it, or maintain it with less resolution than other fields.

This time is presented in the UTC time scale.

See Also

ValidAttributes

5.110.2.2 ACEs

```
property ACEs: TScSFTPACEs;
```

Description

ACL (Access Control List) attribute (taken from NFS Version 4 Protocol [RFC3010]) is an array of access control entries (ACE).

The **ACEs** property holds a list of TScSFTPACEItem objects, that represent ACE which are used in the SFTP protocol.

Note: This property is supported starting with version 4 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.3 AclFlags

```
type
  TScSFTPAclFlag = (aclControlIncluded, aclControlPresent,
aclControlInherited, aclAuditAlarmIncluded,aclAuditAlarmInherited);
  TScSFTPAclFlags = set of TScSFTPAclFlag;

property AclFlags: TScSFTPAclFlags;
```

Description

The **AclFlags** property holds the NFS Access Control attributes.

Note: This property is supported starting with version 6 of the SFTP protocol.

Value	Meaning
`aclControlIncluded`	if this flag is set when creating file attributes, then the client intends to modify the ALLOWED/DENIED entries of the ACEs property. Otherwise, the client intends for these entries to be preserved.
`aclControlPresent`	if this flag is not set, then the client wishes to remove control entries. If the flag is clear, then control of the file may be through the permissions mask. The server may also grant full access to the file. If both the aclControlIncluded and the aclControlPresent flags are set, but they are not ALLOW/DENY entries in the ACEs property, the client wishes to deny all access to the file or directory.
`aclControlInherited`	if this flag is set, then ALLOW/DENY ACEs may be inherited from the parent directory. If it is off, then they must not be INHERITED. If the server does not support controlling inheritance, then the client must clear this bit; in this case the inheritance properties of the server are undefined.
`aclAuditAlarmIncluded`	If flag is set when creating file attributes, then the client intends to modify the AUDIT/ALARM entries of ACEs. Otherwise, the client intends for these entries to be preserved.
`aclAuditAlarmInherited`	If flag is set, then AUDIT/ALARM ACEs may be inherited from the parent directory. If it is off, then they must not be INHERITED. If the server does not support controlling inheritance, then the client must clear this bit; in this case the inheritance properties of the server are undefined.

See Also

ValidAttributes

5.110.2.4 AllocationSize

```
property AllocationSize: Int64;
```

Description

Use the **AllocationSize** property to specify thi file size on disk (in bytes). If this property is set when the file is created, the file is created and the specified number of bytes is pre-allocated. If pre-allocation process fails, the file can be removed (if it was created), and an error will be arised. If the property is set when creating file attributes, the file can be extended or truncated to the specified size. The Size property may be affected by this operation.

Note: This property is supported starting with version 6 of the SFTP protocol.

See Also

Size

ValidAttributes

5.110.2.5 Attrs

```
type
  TScSFTPFileAttr = (faReadonly, faSystem, faHidden, faCaseInsensitive,
faArchive, faEncrypted, faCompressed, faSparse, faAppendOnly,
faImmutable, faSync, faTranslationError);
  TScSFTPFileAttrs = set of TScSFTPFileAttr;

property Attrs: TScSFTPFileAttrs;
```

Description

These flags reflect various attributes of the file or directory on the server.

Note: This property is supported starting with version 5 of the SFTP protocol.

Value	Meaning
faReadonly	advisory, read-only bit. This bit is not a part of the access control information on the file, but is rather an advisory field indicating that the file should not be written.
faSystem	the file is a part of the operating system.
faHidden	file should not be shown to user unless specifically requested. For

	example, most UNIX systems should set this bit if the filename begins with a 'period'. This bit may be read-only. Most UNIX systems will not allow this to be changed.
faCaseInsensitive	this attribute applies only to directories. This attribute is always read-only, and cannot be modified. This attribute means that files and directory names in this directory should be compared without regard to case. Unless otherwise specified, filenames are assumed to be case sensitive.
faArchive	the file should be included in backup/archive operations.
faEncrypted	the file is stored on disk using file-system level transparent encryption. This flag does not affect the file data on the wire (for either READ or WRITE requests.)
faCompressed	the file is stored on disk using file-system level transparent compression. This flag does not affect the file data on the wire.
faSparse	the file is a sparse file; this means that file blocks that have not been explicitly written are not stored on disk. For example, if a client writes a buffer at 10 M from the beginning of the file, the blocks between the previous EOF marker and the 10 M offset would not consume physical disk space. Some servers may store all files as sparse files, in which case this bit will be unconditionally set. Other servers may not have a mechanism for determining if the file is sparse, and so the file MAY be stored sparse even if this flag is not set.
faAppendOnly	opening the file without either the ofAppendData or the ofAppendDataAtomic flag (TScSFTPClient.OpenFile) must result in an SSH_FX_INVALID_PARAMETER error.
faImmutable	the file cannot be deleted or renamed, no hard link can be created to this file, and no data can be written to the file. This bit implies a stronger level of protection than aReadonly, the file permission mask, or ACLs. Typically even the superuser cannot write to immutable files, and only the superuser can set or remove the bit.
faSync	When the file is modified, the changes are written synchronously to the disk.
faTranslationError	The server may include this bit in a directory listing or realpath response. It indicates that there was a failure in the translation to UTF-8. If this flag is included, the server should also include the UntranslatedName property.

See Also

TScSFTPClient.OpenFile

ValidAttributes

5.110.2.6 ChangeAttrTime

```
property ChangeAttrTime: TDateTime;
```

Description

The **ChangeAttrTime** property contains the time of the last attribute modification. The exact meaning of this field depends on the server.

This time is presented in the UTC time scale.

Note: This property is supported starting with version 4 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.7 CreateTime

```
property CreateTime: TDateTime;
```

Description

The **CreateTime** property contains the time when the file was created.

This time is presented in the UTC time scale.

Note: This property is supported starting with version 4 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.8 ExtendedAttributes

```
property ExtendedAttributes: TObjectList;
```

Description

The **ExtendedAttributes** holds the list of the TScSFTPExtension objects that are extended attributes. Additional fields can be added to the file attributes by defining extended attributes for them.

See Also

ValidAttributes

5.110.2.9 FileType

```
property FileType: TScSFTPFileType;
```

Description

The **FileType** property represents the file type.

Note: this option is supported starting with version 4 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.10 GID

```
property GID: integer;
```

Description

The **GID** property contains numeric Unix-like group identifier for a file.

Note: This property is supported only with the version 3 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.11 Group

```
property Group: string;
```

Description

The **Group** property holds the name of the group to which the file belongs.

The string should be of the form "user@dns_domain". This will allow a client and server that do not use the same local representation to translate to common syntax that can be interpreted by both. In the case when no translation is possible for the client or server, the attribute value must be constructed without "@".

Note: This property is supported starting with version 4 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.12 LinkCount

```
property LinkCount: integer;
```

Description

The **LinkCount** property contains the hard link count of the file. This property should not be set when creating file attributes.

Note: This property is supported starting with version 6 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.13 MimeType

```
property MimeType: string;
```

Description

The **MimeType** property contains the mime-type [RFC1521] string. Most servers will not know this information and will not set the flag in their TScSFTPSupportedExtension.SupportedAttributes property.

Note: This property is supported starting with version 6 of the SFTP protocol.

See Also

SupportedAttributes

ValidAttributes

5.110.2.14 ModifyTime

```
property ModifyTime: TDateTime;
```

Description

The **ModifyTime** property contains the time of the last file modification.

This time is presented in the UTC time scale.

See Also

ValidAttributes

5.110.2.15 Owner

```
property Owner: string;
```

Description

The **Owner** property holds the name of the file owner.

The string should be of the form "user@dns_domain". This will allow a client and server that do not use the same local representation to translate to a common syntax that can be interpreted by both. In the case when no translation available to the client or server, the attribute value must be constructed without "@".

Note: This property is supported starting with version 4 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.10 Permissions

```
property Permissions: TScSFTPFilePermissions;
```

Description

The **Permissions** property contains the flags specifying file permissions. These permissions correspond to the st_mode field of the stat structure defined by POSIX [IEEE.1003-1.1996].

See Also

ValidAttributes

5.110.2.11 Size

```
property Size: Int64;
```

Description

Use the **Size** property to specify the number of bytes that can be read from the file, or in other words, the location of the end-of-file. This property should not be set while creating a file. If **Size** is set when creating file attributes, the file can be extended or truncated to the specified size.

See Also

AllocationSize

ValidAttributes

5.110.2.12 TextHint

```
type
  TScSFTPTextHint = (thKnownText, thGuessedText, thKnownBinary,
thGuessedBinary);
```

```
property TextHint: TScSFTPTextHint;
```

Description

The value of the **TextHint** property can be one of the following set, and it indicates what information does the server have about the file content.

This property should not be set when creating file attributes.

Note: This property is supported starting with version 6 of the SFTP protocol.

Value	Meaning
thKnownText	the server knows that the file is a text file, and it will be opened using the ofTextMode flag.
thGuessedText	the server will apply a hueristic or other mechanism and after that the file will be opened with the ofTextMode flag.
thKnownBinary	the server knows that the file has binary content.
thGuessedBinary	the server will apply a hueristic or other mechanism and believes has binary content, and after that file will not be opened with the ofTextMode flag.

See Also

ValidAttributes

5.110.2.19 UID

```
property UID: integer;
```

Description

The **UID** property contains numeric Unix-like user identifiers for a file.

Note: This property is supported only with the version 3 of the SFTP protocol.

See Also

ValidAttributes

5.110.2.20 UntranslatedName

```
property UntranslatedName: string;
```

Description

The **UntranslatedName** property contains the name of the file before its translation was attempted. It

should not be included unless the faTranslationError flag in the Attrs property is set on the server side.

Note: This property is supported starting with version 6 of the SFTP protocol.

See Also

Attrs

ValidAttributes

5.110.2.2 ValidAttributes

```
property ValidAttributes: TScSFTPAttributes;
```

Description

Use the **ValidAttributes** property to define the file attributes that have meaning. While receiving attributes from the server when a flag for the required property was set, this value was received from the server. If the flag is not set, the value of the property can be set to any value.

When sending attributes to the server, only properties with the corresponding flag are sent. Other properties are ignored.

Value	Meaning
aSize	the Size property is set;
aAllocationSize	the AllocationSize prperty is set;
aOwnerGroup	the GID, UID, Group and Owner properties are set;
aPermissions	the Permissions property is set;
aAccessTime	the AccessTime property is set;
aCreateTime	the CreateTime property is set;
aModifyTime	the ModifyTime property is set;
aChangeAttrTime	the ChangeAttrTime property is set;
aSubsecondTimes	the nseconds is to be added to the seconds AccessTime, CreateTime, ModifyTime, ChangeAttrTime fields for the final time representation.
aAcl	the AclFlags and ACEs are set;
aAttrs	the Attrs property is set;
aTextHint	the TextHint property is set;
aMimeType	the MimeType property is set;
aLinkCount	the LinkCount property is set;
aUntranslatedName	the UntranslatedName property is set;
aExtended	the ExtendedAttributes property is set;

5.110.3 Methods

5.110.3.1 GetAttributesAsLongname

```
function GetAttributesAsLongname: string;
```

Description

The **GetAttributesAsLongname** method returns an expanded description for the file, similar to the one returned by "ls -l" on Unix systems.

5.111 TScSFTPFileInfo

5.111.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPFileInfo** class holds the information about a file.

See also

TScSFTPClient.ReadDirectory

TScSFTPServer.DefaultReadDirectory

5.111.2 Properties

5.111.2.1 Attributes

```
property Attributes: TScSFTPFileAttributes;
```

Description

The **Attributes** property holds the attributes of the file or directory.

See also

TScSFTPFileAttributes

5.111.2.2 Filename

```
property Filename: string;
```

Description

The **Filename** property holds the name of the file.

5.111.2.3 Longname

```
property Longname: string;
```

Description

The **Longname** property holds an expanded format for the file name, similar to the one returned by "ls -l" on Unix systems.

Note: Is set only by version 3 of the SFTP protocol.

5.112 TScFilenameTranslationControlExtension

5.112.1 Description

Unit

ScSFTPUtils

Description

The **TScFilenameTranslationControlExtension** class represents the filename translation control extension.

If the server included the 'filename-charset' extension in initialization extensions, a client MAY send this extension to turn off server translation to UTF-8.

See Also

FilenameCharset

FilenameCharsetAvailable

RequestExtension

5.112.2 Properties

5.112.2.1 DoTranslate

```
property DoTranslate: Boolean;
```

Description

Set the **DoTranslate** property to True for server to enable filename translation to UTF-8. If this property is set to False, server disables filename translation.

5.113 TScCheckFileReplyExtension

5.113.1 Description

Unit

ScSFTPUtils

Description

The **TScCheckFileReplyExtension** class represents the 'check-file-reply' extension that holds the server answer for the check file extension query.

See Also

CheckFile

CheckFileByHandle

5.113.2 Properties

5.113.2.1 HashAlgorithm

```
property HashAlgorithm: string;
```

Description

The **HashAlgorithm** property defines the hash algorithm that was actually used.

5.113.2.2 Hashes

```
property Hashes[Index: Integer]: TBytes;
```

Description

The **Hashes** property holds the computed hashes.

5.113.2.3 HashesCount

```
property HashesCount: Integer;
```

Description

The **HashesCount** property holds the number of the computed hashes.

5.114 TScSFTPSupportedAclExtension

5.114.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPSupportedAclExtension** class represents the SFTP supported Acl extension that holds the capabilities of the ACL attribute supported by the server. The server sends this extension during the connection initialization.

Note: Is supported starting with the version 6 of the SFTP protocol.

See Also

TScSFTPServerProperties.SupportedAcls

5.114.2 Properties

5.114.2.1 SupportedAcls

type

```
  TScSFTPSupportedAcl = (saAllow, saDeny, saAudit, saAlarm,
saInheritAccess, saInheritAuditAlarm);
  TScSFTPSupportedAcls = set of TScSFTPSupportedAcl;

property SupportedAcls: TScSFTPSupportedAcls;
```

Description

The **SupportedAcls** property holds a set of the supported capabilities of the ACL attribute.

Value	Meaning
saAllow	The server supports the atAccessAllowed ACE type;
saDeny	The server supports the atAccessDenied ACE type;
saAudit	The server supports the atSystemAudit ACE type;
saAlarm	The server supports the atSystemAlarm ACE type;
saInheritAccess	The server can control whether an ACL will inherit DENY and ALLOW ACEs that are marked as inheritable from it's parent object;
saInheritAuditAlarm	The server can control whether an ACL will inherit AUDIT or ALARM ACEs that are marked inheritable from it's parent object.

5.115 TScSFTPSupportedExtension

5.115.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPSupportedExtension** class represents the SFTP supported extension.

SFTP protocol supports a number of features that may not be supported by all servers. When a server receives a request for a feature it does not support, it returns 'UNSUPPORTED' error status code, unless otherwise specified. The supported extension facilitates clients that are able to use the maximum available feature set, and yet not be overburdened by dealing with the error status codes.

Server sends the **TScSFTPSupportedExtension** extension during the connection initialization. When client executes some command, it checks if this operation and its attributes are supported. In case server does not support this operation or its attributes, it generates an exception or uses a mask for the attributes depending on the value of the RaiseError property.

Note: Is supported since the version 5 of the SFTP protocol.

See Also

TScSFTPServerProperties.SupportedExtension

5.115.2 Properties

5.115.2.1 MaxReadSize

```
property MaxReadSize: Integer;
```

Description

The **MaxReadSize** property holds the maximum read size that the server guarantees to complete. For example, certain server implementations complete only the first 4K of a read, even if there is additional data to be read from the file.

5.115.2.2 RaiseError

```
property RaiseError: Boolean;
```

Description

If the **RaiseError** property is set to True, an error is raised when attempting to execute an invalid command or attributes. In this case the command is not sent to the server.

If False, then a mask is applied to the attributes (if possible) and the command is sent to the server. In this case, if the server does not support the command, it will return an error message.

The default value is True.

5.115.2.3 SupportedAccessMask

```
property SupportedAccessMask: TScSFTPAceMask;
```

Description

Use the **SupportedAccessmask** property to specify the supported access flags on file opening using TScSFTPClient.OpenFile.

See Also

TScSFTPClient.OpenFile

5.115.2.4 SupportedAttribExtensionNames

```
property SupportedAttribExtensionNames: TStringList;
```

Description

The **SupportedAttribExtensionNames** property holds the list of extension names that can be used in TScSFTPFileAttributes.ExtendedAttributes.

5.115.2.5 SupportedAttributeBits

```
property SupportedAttributeBits: TScSFTPFileAttrs;
```

Description

Use the **SupportedAttributeBits** property to specify the file attributes (supported by the server) usage in TScSFTPFileAttributes.Attrs.

5.115.2.6 SupportedAttributes

```
property SupportedAttributes: TScSFTPAttributes;
```

Description

Use the **SupportedAttributes** property to specify the attributes (supported by the server) of TScSFTPFileAttributes.

See Also

ValidAttributes

5.115.2.7 SupportedBlockModes

```
property SupportedBlockModes: TScSFTPBlockModes;
```

Description

Use the **SupportedBlockModes** property to pass the supported block modes as one of the supported parameters to the TScSFTPClient.OpenFile method on file opening.

See Also

TScSFTPClient.OpenFile

5.115.2.8 SupportedExtensionNames

```
property SupportedExtensionNames: TStringList;
```

Description

The **SupportedExtensionNames** property holds the list of extension that can be used in the TScSFTPClient.RequestExtension method.

See Also

TScSFTPClient.RequestExtension

5.115.2.9 SupportedOpenFlags

```
property SupportedOpenFlags: TScSFTPFileOpenFlags;
```

Description

Use the **SupportedOpenFlags** property to specify the supported file open flags used in TScSFTPClient.OpenFile.

See Also

TScSFTPClient.OpenFile

5.115.3 Methods

5.115.3.1 IsBlockSetAvailable

```
function IsBlockSetAvailable: boolean;
```

Description

Call the **IsBlockSetAvailable** method to check if any block mode is supported by SFTP server on

requesting to block a file. If the mode is supported, it returns True. False otherwise.

See Also

Block

5.115.3.2 IsOpenBlockSetAvailable

```
function IsOpenBlockSetAvailable: boolean;
```

Description

Call the **IsOpenBlockSetAvailable** to check if any block mode is supported by SFTP server on file opening. If the mode is supported, it returns True. False otherwise.

See Also

OpenFile

5.115.3.3 IsSupportedBlockSet

```
function IsSupportedBlockSet(const BlockModes: TScSFTPBlockModes):
boolean;
```

Description

Call the **IsSupportedBlockSet** method to check if the specified block modes set is supported by SFTP server on requesting to block a file. If the mode is supported, it returns True. False otherwise.

See Also

Block

5.115.3.4 IsSupportedOpenBlockSet

```
function IsSupportedOpenBlockSet(const BlockModes: TScSFTPBlockModes):
boolean;
```

Description

Call the **IsSupportedOpenBlockSet** to check if the specified block modes set is supported by SFTP server on file opening. If the mode is supported, it returns True. False otherwise.

See Also

OpenFile

5.116 TScSFTPVendorExtension

5.116.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPVendorExtension** class represents the vendor extension.

It is often necessary to detect the version of the server to workaround bugs. The vendor extension allows the client to do so.

Server sends the **TScSFTPVendorExtension** extension during the connection initialization.

See Also

Vendor

5.116.2 Properties

5.116.2.1 ProductBuildNumber

```
property ProductBuildNumber: Int64;
```

Description

The **ProductBuildNumber** property holds the build-number for the product. So, if a bug is fixed in the build-number 'x', it can be assumed that (barring regression in the product) it is fixed in all build-numbers after 'x'.

5.116.2.2 ProductName

```
property ProductName: string;
```

Description

The **ProductName** property holds an arbitrary name identifying the product.

5.116.2.3 ProductVersion

```
property ProductVersion: string;
```

Description

The **ProductVersion** property holds an arbitrary string identifying the version of the product.

5.116.2.4 VendorName

```
property VendorName: string;
```

Description

The **VendorName** property holds an arbitrary name identifying the product vendor.

5.117 TScSFTPVersionsExtension

5.117.1 Description

Unit

ScSFTPUtils

Description

The **TScSFTPVersionsExtension** class represents the versions extension.

If the server supports any other versions besides the one that was sent by client during negotiation, it may send the versions extension to inform the client of this fact. In this case the client may choose which of the supported versions to use.

Server sends the **TScSFTPVersionsExtension** extension during the connection initialization.

See Also

Versions

5.117.2 Properties

5.117.2.1 AsString

```
property AsString: string;
```

Description

The **AsString** property holds a string of version numbers separated by commas. The defined versions are: "2", "3", "4", "5", "6". Any other version advertised by the server should follow the DNS extensibility naming convention outlined in [I-D.ietf-secsh-architecture]. For example: "2,3,6,private@example.com".

5.117.2.2 Versions

```
property Versions: TScSFTPVersions;
```

Description

The **Versions** property is an unparsed set of versions that are supported by the server. The AsString property holds this set as a string of version numbers separated by commas.

5.118 TScSpaceAvailableReplyExtension

5.118.1 Description

Unit

ScSFTPUtils

Description

The **TScSpaceAvailableReplyExtension** class represents the space available reply extension that holds the answer of the SFTP server on the request of the query available space extension.

See Also

QueryAvailableSpace

5.118.2 Properties

5.118.2.1 BytesAvailableToUser

```
property BytesAvailableToUser: Int64;
```

Description

The **BytesAvailableToUser** property holds the total number of bytes, both used and unused, available to the authenticated user on the device. Holds 0 if this number is unknown.

5.118.2.2 BytesOnDevice

```
property BytesOnDevice: Int64;
```

Description

The **BytesOnDevice** property holds the total number of bytes on the device, both used and unused. Is 0 if the total number of bytes is unknown.

5.118.2.3 BytesPerAllocationUnit

```
property BytesPerAllocationUnit: Int64;
```

Description

The **BytesPerAllocationUnit** property holds the number of bytes in each allocation unit on the device, or in other words, the minimum number of bytes that a file allocation size can grow or shrink by. If the server does not know this information, or the file-system in use does not use allocation blocks, this value must be 0.

5.118.2.4 UnusedBytesAvailableToUser

```
property UnusedBytesAvailableToUser: Int64;
```

Description

The **UnusedBytesAvailableToUser** property holds the total number of unused bytes available to the authenticated user on the device. Holds 0 if the number is unknown.

5.118.2.5 UnusedBytesOnDevice

```
property UnusedBytesOnDevice: Int64;
```

Description

The **UnusedBytesOnDevice** property holds the total number of unused bytes available on the device. Holds 0 if the number unknown.

5.119 TScSSLCipherSuiteItem

5.119.1 Description

Unit

ScSSLClient

Description

The **TScSSLCipherSuiteItem** class is a descendant of the TScCollectionItem class, and it represents encryption and data integrity algorithm in the TLS/SSL format.

See Also

TScSSLCipherSuites

5.119.2 Properties

5.119.2.1 CipherAlgorithm

```
property CipherAlgorithm: TScSSLCipherAlgorithm;
```

Description

CipherAlgorithm represents encryption and data integrity algorithm which can be using in TLS/SSL connection.

See Also

AsString

5.120 TScSSLCipherSuites

5.120.1 Description

Unit

ScSSLClient

Description

The **TScSSLCipherSuites** class is a descendant of the TScCollection class, and it is a container for TScSSLCipherSuiteItem objects.

TScSSLCipherSuites keeps a list of encryption and data integrity algorithms which can be using in TLS/SSL connection, and represents them in the TLS/SSL format.

See Also

TScSSLCipherSuiteItem

5.121 TScSSLSessionInfo

5.121.1 Description

Unit

ScSSLTypes

Description

The **TScSSLSessionInfo** class holds the information about the current TLS/SSL connection.

See also

TScSSLClient.SessionInfo

TScSSLServerConnection.SessionInfo

5.121.2 Properties

5.121.2.1 CipherAlgorithm

```
property CipherAlgorithm: TScSSLCipherAlgorithm;
```

Description

The **CipherAlgorithm** property holds the algorithm used in the current TLS/SSL session for data encryption and integrity verification.

This property is set during the TLS/SSL handshake.

5.121.2.2 Compression

```
property Compression: TScCompression;
```

Description

The **Compression** property specifies how data transferred between the client and server is compressed in the current TLS/SSL session.

This property is set during the TLS/SSL handshake.

5.121.2.3 Initialized

```
property Initialized: boolean;
```

Description

The **Initialized** property indicates whether a TLS/SSL session has been established. **Initialized** will be set to True after the TLS/SSL handshake with the remote party.

5.121.2.4 MasterSecret

```
property MasterSecret: TBytes;
```

Description

The **MasterSecret** property holds the session key for encrypting and supporting the integrity of the transferred data.

This property is set during the TLS/SSL handshake.

See Also

SessionID

5.121.2.5 NewSessionTicketCount

```
property NewSessionTicketCount: integer;
```

Description

The **NewSessionTicketCount** property holds the number of New Session Tickets received from the server during the TLS/SSL handshake. This property has effect only for the server session.

A New Session Ticket creates a unique association between the ticket value and the session key which is used to generate encryption keys and verify data integrity.

The TLS/SSL client may use these New Session Tickets for session resumption, which greatly improves the performance of the client and server when multiple parallel or serial connections are used between the same client and server.

This property is set during the TLS/SSL handshake.

See Also

NewSessionTickets

5.121.2.6 NewSessionTickets

```
property NewSessionTickets: TObjectList;
```

Description

The **NewSessionTickets** property holds the list of New Session Tickets received from the server during the TLS/SSL handshake. It has affect only when working with the client session.

A New Session Ticket creates a unique association between the ticket value and the session key used for generating encryption keys and verifying data integrity.

The TLS/SSL client may use these New Session Tickets for session resumption, which greatly improves the performance of the client and server when multiple parallel or serial connections are used between the same client and server.

This property is set during the TLS/SSL handshake.

See Also

TicketNonce

5.121.2.7 Protocol

```
property Protocol: TScSSLProtocol;
```

Description

The **Protocol** property holds the version of TLS/SSL protocol.

This property is set during the TLS/SSL handshake.

5.121.2.8 RemoteCertificate

```
property RemoteCertificate: TScCertificate;
```

Description

The **RemoteCertificate** property holds the certificate object obtained from the remote party during the TLS/SSL handshake.

This property is set during the TLS/SSL handshake.

5.121.2.9 SessionID

```
property SessionID: TBytes;
```

Description

The **SessionID** property holds the session identifier that is used during the TLS session resumption.

This property is set during the TLS/SSL handshake.

See Also

MasterSecret

5.121.2.1(TicketNonce

```
property TicketNonce: TBytes;
```

Description

The **TicketNonce** property holds the session ticket received from the TLS/SSL client during session resumption. The client use this New Session Ticket to request session resumption.

This property is set during the TLS/SSL handshake.

See Also

NewSessionTicketCount

NewSessionTickets

5.122 TTLSHelloExtension

5.122.1 Description

Unit

ScSSLExtensions

Description

The **TTLSHelloExtension** class is used for extensions to support the TLS protocol. The TLS extension represents a record that contains the extension type and data specific for a particular type.

Extensions can be sent from TLS client to TLS server or backward during handshake when starting a new TLS session and when requesting session resumption.

TLS extensions allow extending the information about client and server certificates, encryption abilities, signature algorithms, etc.

TTLSHelloExtension is an abstract base class, which is the ancestor for all other TLS extension classes. It declares an interface to parse extension data from raw bytes and to decode it backward to an array of bytes.

See Also

TTLSApplicationLayerProtocolNegotiationExtension

TTLSEllipticCurvePointFormatsExtension

TTLSExtendedMasterSecretExtension

TTLSHelloExtensions

TTLSRenegotiationIndicationExtension

TTLSServerNameExtension

TTLSSessionTicketExtension

TTLSSignatureAlgorithmsExtension

TTLSSupportedGroupsExtension

5.122.2 Methods

5.122.2.1 AsBytes

```
function AsBytes: TBytes;
```

Description

Returns the raw data for the TLS hello extension as an array of bytes. The format of the raw data is determined by the type of extension.

See Also

Parse

5.122.2.2 Parse

```
procedure Parse(const Buffer: TBytes; Offset, Count: integer); virtual;
abstract;
```

Description

Decodes the specified raw data to a specific extension class.

This method is abstract and it should be overridden in descendant classes to decode data for every specific extension.

See Also

AsBytes

5.123 TTLSServerNameExtension

5.123.1 Description

Unit

ScSSLExtensions

Description

The **TTLSServerNameExtension** class represents the server name extension that provides a mechanism for a TLS client to tell a server the name of the server it is contacting. It may be desirable for clients to provide this information to facilitate secure connections to servers that host multiple virtual servers at a single underlying network address.

This extension is described in RFC 6066, section 3.

See Also

TScSSLSecurityOptions.IdentityDNSName

TTLSHelloExtension

5.123.2 Properties

5.123.2.1 ServerNames

```
property ServerNames: TStringList;
```

Description

ServerNames contains list of the fully qualified DNS hostnames of the server, as understood by the client.

This extension provides a mechanism for a client to tell a server the name of the server it is contacting.

5.124 TTLSExtendedMasterSecretExtension

5.124.1 Description

Unit

ScSSLExtensions

Description

The **TTLSExtendedMasterSecretExtension** class represents TLS session hash and extended master secret extension. This extension is used to signal both client and server to use the extended master secret computation, thus preventing possible man-in-the-middle attacks.

To use this extension it's enough to create the **TTLSExtendedMasterSecretExtension** instance and add it to the TScSSLClient.ClientHelloExtensions list.

This extension is described in RFC 7627.

See Also

TTLSHelloExtension

5.125 TTLSSessionTicketExtension

5.125.1 Description

Unit

ScSSLExtensions

Description

The **TTLSSessionTicketExtension** class represents the session ticket TLS extension. This extension defines a way to resume a TLS session without requiring session-specific state at the TLS server. If the client possesses a ticket that it wants to use to resume a session, then it includes the ticket in the session ticket extension in the ClientHello packet.

This extension is described in RFC 4507, section 3.2.

See Also

TTLSHelloExtension

5.125.2 Properties

5.125.2.1 Ticket

```
property Ticket: TBytes;
```

Description

Determines the ticket that TLS client or server wants to use to resume a session.

5.126 TTLSSignatureAlgorithmsExtension

5.126.1 Description

Unit

ScSSLExtensions

Description

The **TTLSSignatureAlgorithmsExtension** class represents the signature algorithms extension to indicate to the TLS server which signature schemes may be used in digital signatures. **TTLSSignatureAlgorithmsExtension** contains a list of signature schemes that the client is willing to verify. The values are indicated in descending order of preference.

Note: this extension is not meaningful for TLS versions prior to 1.2.

This extension is described in RFC 5246, section 7.4.1.4.1.

See Also

TTLSHelloExtension

5.126.2 Properties

5.126.2.1 Count

```
property Count: integer;
```

Description

Read **Count** to determine the number of signature schemes that may be used in digital signatures. The **Count** property specifies the number of values in the SignatureSchemes list.

This property is read-only.

See Also

SignatureSchemes

5.126.2.2 SignatureSchemes

```
property SignatureSchemes[Index: integer]: TScSSLSignatureAlgorithm;
```

Description

Lists the signature algorithms which may be used.

Use **SignatureSchemes** to obtain the signature scheme. **SignatureSchemes** is a zero-based array: the first algorithm is indexed as 0, the second algorithm is indexed as 1, and so on. The `Index` parameter indicates the index of the algorithm. You can read the value at a specific index, or use **SignatureSchemes** with the Count property to iterate through the list.

This property is read-only.

See Also

Count

5.126.3 Methods

5.126.3.1 Add

```
procedure Add(Hash: TScHashAlgorithm; Signature:
TScSSLSignatureAlgorithm); overload;
procedure Add(SignatureScheme: TScSSLSignatureScheme); overload;
```

Description

Call **Add** to insert a hash/signature pair or a new signature scheme at the end of the list. **Add**

increments Count and, if necessary, allocates memory.

See Also

Count

SignatureSchemes

5.126.3.2 Clear

```
procedure Clear;
```

Description

Deletes all items from the SignatureSchemes lists.

Call **Clear** to empty the signature schemes list and set the Count to 0.

See Also

Count

SignatureSchemes

5.127 TTLSApplicationLayerProtocolNegotiationExtension

5.127.1 Description

Unit

ScSSLExtensions

Description

The **TTLSApplicationLayerProtocolNegotiationExtension** class represents the application-layer protocol negotiation extension.
For instances in which multiple application protocols are supported on the same TCP port, this extension allows the application layer to negotiate which protocol will be used within the TLS connection. The client sends the list of supported application protocols as part of the TLS ClientHello message. The server chooses a protocol and sends the selected protocol as part of the TLS ServerHello message.

This extension is described in RFC 7301.

See Also

TTLSHelloExtension

5.127.2 Properties

5.127.2.1 ProtocolNames

```
property ProtocolNames: TStringList;
```

Description

ProtocolNames contains the list of protocols advertised by the client, in descending order of preference. Protocols are named by IANA-registered, opaque, non-empty byte strings.

The client sends this list to the TLS server. The server chooses a protocol and sends the selected protocol to the client.

5.128 TTLSEllipticCurvePointFormatsExtension

5.128.1 Description

Unit

ScSSLExtensions

Description

The **TTLSEllipticCurvePointFormatsExtension** class represents the supported Elliptic Curve point formats extension that allows a TLS client to enumerate the point formats it can parse.
This extension allows negotiating the use of specific point formats (e.g., compressed vs. uncompressed, respectively) during a handshake starting a new TLS session.

This extension is described in RFC 4492, section 5.1.2.

See Also

TScECPointFormats

TTLSSupportedGroupsExtension

TTLSHelloExtension

5.128.2 Properties

5.128.2.1 ECPointFormats

```
property ECPointFormats: TScECPointFormats;
```

Description

The **ECPointFormats** property indicates the set of point formats (e.g., compressed vs. uncompressed, respectively) that the client can parse.

5.129 TTLSSupportedGroupsExtension

5.129.1 Description

Unit

ScSSLExtensions

Description

The **TTLSSupportedGroupsExtension** class represents the supported negotiated groups extension that allows a TLS client to enumerate the named groups it supports for key exchange.
This extension allows negotiating the use of specific named groups during a handshake when starting a new TLS session. The values are indicated in descending order of preference.

In versions of TLS prior to TLS 1.3, this extension was named "elliptic_curves" and only contained elliptic curve groups. TLS 1.3 allows using Elliptic Curve groups and Finite Field groups.

This extension is described in RFC 8446, section 4.2.7, and in RFC 4492, section 5.1.1

See Also

TScKExNamedGroupType

TTLSHelloExtension

5.129.2 Properties

5.129.2.1 Count

```
property Count: integer;
```

Description

Read **Count** to determine the number of the named groups which the TLS client supports for key exchange. The **Count** property specifies the number of values in the KExNamedGroups list.

This property is read-only.

See Also

Add

KExNamedGroups

5.129.2.2 ECCount

```
property ECCount: integer;
```

Description

Read **ECCount** to determine the number of the elliptic curves that TLS client supports. The **ECCount** property specifies the number of values in the EllipticCurves list.

This property is read-only.

See Also

Add

EllipticCurves

5.129.2.3 EllipticCurves

```
property EllipticCurves[Index: integer]: TScECName;
```

Description

Lists the elliptic curves which may be used. The values are indicated in descending order of preference.

Use **EllipticCurves** to obtain the elliptic curve type. **EllipticCurves** is a zero-based array: the first type is indexed as 0, the second type is indexed as 1, and so on. The `Index` parameter indicates the index of the elliptic curve. You can read the value at a specific index, or use **EllipticCurves** with the ECCount property to iterate through the list.

See Also

Add

ECCount

5.129.2.4 KExNamedGroups

```
property KExNamedGroups[Index: integer]: TScKExNamedGroupType;
```

Description

Lists the named groups, which the client supports for key exchange. The values are indicated in descending order of preference.

Use **KExNamedGroups** to obtain the named group for key exchange. **KExNamedGroups** is a zero-based array: the first type is indexed as 0, the second type is indexed as 1, and so on. The `Index`

parameter indicates the index of the named group. You can read the value at a specific index or use **KExNamedGroups** with the Count property to iterate through the list.

See Also

Add

Count

5.129.3 Methods

5.129.3.1 Add

```
procedure Add(const ECurve: TScECName); overload;
procedure Add(const FFGType: TScFiniteFieldGroupType); overload;
procedure Add(const KExNamedGroup: TScKExNamedGroupType); overload;
```

Description

Call **Add** to insert a named group, which the client supports for key exchange at the end of the list. **Add** increments Count and, if necessary, allocates the memory.

In versions of TLS prior to TLS 1.3, this extension was named "elliptic_curves" and contained only elliptic curve groups. TLS 1.3 allows using Elliptic Curve groups and Finite Field groups.

See Also

Count

5.129.3.2 Clear

```
procedure Clear;
```

Description

Deletes all items from the KExNamedGroups list.

Call **Clear** to empty the named groups list and set the Count and the ECCount to 0.

See Also

Count

KExNamedGroups

5.130 TTLSRenegotiationIndicationExtension

5.130.1 Description

Unit

ScSSLExtensions

Description

The **TTLSRenegotiationIndicationExtension** class represents the renegotiation indication extension to cryptographically tie renegotiations to the TLS connections they are being performed over, thus preventing a man-in-the-middle attack.
SSL and TLS renegotiation are vulnerable to an attack in which the attacker forms a TLS connection with the target server, injects content of his choice, and then splices in a new TLS connection from a client.

To use this extension it's enough to create the **TTLSRenegotiationIndicationExtension** instance and add it to the TScSSLClient.ClientHelloExtensions list.

This extension is described in RFC 5746.

See Also

TTLSHelloExtension

5.130.2 Properties

5.130.2.1 IsServerSupport

```
property IsServerSupport: boolean;
```

Description

Indicates whether the TLS server supports the renegotiation indication extension. The **IsServerSupport** property is set automatically on the TLS connection handshake. If the server supports this extension, the user can renegotiate the connection to avoid attacks.

This property is read-only.

5.130.3 Methods

5.130.3.1 Check

```
procedure Check(ReceivedExtension: TTLSRenegotiationIndicationExtension);
```

Description

Checks that renegotiation was successful. The **Check** method is called automatically on the TLS connection negotiation.

5.130.3.2 Clear

```
procedure Clear;
```

Description

Call **Clear** to empty all stored data about the previous renegotiations. This method is called automatically in the initial handshake on the TLS connection negotiation.

5.130.3.3 Renegotiate

```
procedure Renegotiate;
```

Description

Indicates that renegotiation was started by the TLS client or server. The **Renegotiate** method is called automatically on the ClientHello message negotiation.

5.131 TTLSHelloExtensions

5.131.1 Description

Unit

ScSSLExtensions

Description

TTLSHelloExtensions maintains a list of the TTLSHelloExtension objects.

Use **TTLSHelloExtensions** to store and maintain a list of objects. **TTLSHelloExtensions** provides properties and methods to add, delete, locate, and access objects. **TTLSHelloExtensions** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TTLSHelloExtensions** instance is itself destroyed.

See also

ClientHelloExtensions

ServerHelloExtensions

TTLSHelloExtension

5.131.2 Properties

5.131.2.1 Extensions

```
property Extensions[Index: integer]: TTLSHelloExtension;
```

Description

Lists the TTLSHelloExtension object references.

Use **Extensions** to access objects in the list. **Extensions** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Extensions** with the Count property to iterate through the list.

Reassigning an **Extensions** index frees the object that previously occupied that position in the list.

See also

TTLSHelloExtension

5.131.3 Methods

5.131.3.1 AsBytes

```
function AsBytes: TBytes;
```

Description

Returns the raw data for the list of hello extensions as an array of bytes that used in the ClientHello message on TLS negotiation.

5.131.3.2 Assign

```
procedure Assign(List: TTLSHelloExtensions);
```

Description

Call the **Assign** method to assign the elements of another extension list to this one.

5.132 TScSSLSecurityOptions

5.132.1 Description

Unit

ScSSLClient

Description

The **TScSSLSecurityOptions** class determines behaviour of the TLS/SSL client.

See also

TScSSLClient.SecurityOptions

5.132.2 Properties

5.132.2.1 AllowLoadCRLByHttp

```
property AllowLoadCRLByHttp: boolean; default True;
```

Description

Determines whether a certificate revocation list (CRL) will be loaded from an HTTP resource during the TLS/SSL handshake when starting a new session if the required CRL was not found in the current storage.

CRL is used to verify whether the server certificate is valid and has not been revoked by the user. When verifying the validity of an SSL certificate, CRL is searched for in the current storage based on the identifier contained in the certificate itself. If the CRL is not available in the current storage, it can be retrieved from an HTTP resource.

Set **AllowLoadCRLByHttp** to True to allow a download of a CRL list from a URI.

Set **AllowLoadCRLByHttp** to False to disable a download of a CRL list from a URL. If no CRL was found, the TLS server is considered invalid, and the session is closed.

If you do not want to check the revocation status of an SSL certificate, use the DisableCRLValidation property.

The default value is True.

See Also

DisableCRLValidation

5.132.2.2 DisableCRLValidation

```
property DisableCRLValidation: boolean; default False;
```

Description

Determines whether the client will check CRL when verifying the server certificate during the TLS/SSL handshake when starting a new session.

Set **DisableCRLValidation** to True to disable verification of the revocation status of a server certificate, which weakens TLS security.

Set **DisableCRLValidation** to False to enforce verification of the revocation status of a server certificate on the client. If a certificate is revoked or a CRL list is not found, the TLS server is considered invalid, and the session is closed.

The default value is False.

See Also

AllowLoadCRLByHttp

5.132.2.3 IdentityDNSName

```
property IdentityDNSName: string;
```

Description

Determines whether the TLS server name extension will be sent from the client to the TLS/SSL server during the TLS/SSL handshake when starting a new session.

Set **IdentityDNSName** to add the TTLSServerNameExtension object to the TScSSLClient.ClientHelloExtensions list.

See also

TTLSServerNameExtension

TScSSLClient.ClientHelloExtensions

5.132.2.4 IgnoreServerCertificateConstraints

```
property IgnoreServerCertificateConstraints: boolean; default False;
```

Description

Determines whether a server certificate will be verified for compliance with constraints during the TLS/SSL handshake when starting a new session.

Set **IgnoreServerCertificateConstraints** to True, in order for a client to ignore the permission to use the certificate for the required purposes. Set **IgnoreServerCertificateConstraints** to False, in order for a client to check the permission to use a certificate for the required purposes. In case if the certificate does not match the constraints, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateInsecurity

IgnoreServerCertificateValidity

TrustServerCertificate

TScSSLClient.ClientHelloExtensions

5.132.2.5 IgnoreServerCertificateInsecurity

```
property IgnoreServerCertificateInsecurity: boolean; default False;
```

Description

Determines whether a server certificate signature security will be checked during the TLS/SSL handshake when starting a new session.

Set **IgnoreServerCertificateInsecurity** to True, in order for the client to ignore the certificate signature security. Set **IgnoreServerCertificateInsecurity** to False, in order for the client to check the certificate signature security.

In case if a hash algorithm for signing the certificate is set to one of following algorithms: [haNone, haSHA1, haMD5, haMD4, haMD2], - signature is considered insecured, and therefore, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateConstraints

IgnoreServerCertificateValidity

TrustServerCertificate

TScSSLClient.ClientHelloExtensions

5.132.2.6 IgnoreServerCertificateValidity

```
property IgnoreServerCertificateValidity: boolean; default False;
```

Description

Determines whether a server certificate validity period will be checked during the TLS/SSL handshake when starting a new session.

Set **IgnoreServerCertificateValidity** to True, in order for the client to ignore the certificate validity period. Set **IgnoreServerCertificateValidity** to False, in order for the client to check the certificate validity period. In case if a certificate validity period expires, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateConstraints

IgnoreServerCertificateInsecurity

TrustServerCertificate

TScSSLClient.ClientHelloExtensions

5.132.2.7 TrustSelfSignedCertificate

```
property TrustSelfSignedCertificate: boolean; default False;
```

Description

Determines whether to trust a self-signed certificate to authenticate the server during the TLS/SSL handshake when starting a new session.

Set **TrustSelfSignedCertificate** to True to make the client trust a self-signed certificate received from the server.

Set **TrustSelfSignedCertificate** to False to forbid the use of self-signed certificate by the server—in this case, if the server certificate is self-signed, the TLS/SSL server is considered invalid, and the session is closed.

The default value is False.

See Also

IgnoreServerCertificateConstraints

IgnoreServerCertificateInsecurity

IgnoreServerCertificateValidity

TrustStorageCertificates

5.132.2.8 TrustServerCertificate

```
property TrustServerCertificate: boolean; default False;
```

Description

Determines whether a server certificate will be verified during the TLS/SSL handshake when starting a new session.

Set **TrustServerCertificate** to True, in order for the client to trust the server in any case regardless of compliance with the certificate.

Set **TrustServerCertificate** to False, in order for the client to verify a certificate. In this case if the certificate does not meet any requirements, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateConstraints

IgnoreServerCertificateInsecurity

IgnoreServerCertificateValidity

5.132.2.9 TrustStorageCertificates

```
property TrustStorageCertificates: boolean; default False;
```

Description

Determines whether certificates in the client storage will be used to verify the server certificate.

When verifying a certificate, the signature is checked in the entire certificate chain, up to the self-signed certificate. If a non-standard certificate, trusted by the client, was used to sign the certificate, that certificate can be added to the client storage.

Set **TrustStorageCertificates** to True to use the certificates kept in the client storage, in addition to the system certificates, when verifying the server certificate.

Set **TrustStorageCertificates** to False to use only the system ROOT certificates when verifying the server certificate.

The default value is False.

See also

TrustSelfSignedCertificate

5.132.2.10 UseExtendedMasterSecret

```
property UseExtendedMasterSecret: boolean; default True;
```

Description

Determines if the TLS extended master secret extension will be sent from the client to the TLS/SSL server during the TLS/SSL handshake when starting a new session.

Set **UseExtendedMasterSecret** to True, to add the TTLSExtendedMasterSecretExtension object to the TScSSLClient.ClientHelloExtensions list. Set **UseExtendedMasterSecret** to False, to remove the TTLSExtendedMasterSecretExtension instance from the TScSSLClient.ClientHelloExtensions list.

The default value is True.

See also

TTLSExtendedMasterSecretExtension

TScSSLClient.ClientHelloExtensions

5.132.2.11 UseSecureRenegotiation

```
property UseSecureRenegotiation: boolean; default True;
```

Description

Determines if the TLS renegotiation indication extension will be sent from the client to the TLS/SSL server during the TLS/SSL handshake when starting a new session and when requesting session resumption.

Set **UseSecureRenegotiation** to True, to add the TTLSRenegotiationIndicationExtension object to the TScSSLClient.ClientHelloExtensions list. Set **UseSecureRenegotiation** to False, to remove the TTLSRenegotiationIndicationExtension instance from the TScSSLClient.ClientHelloExtensions list.

The default value is True.

See also

TTLSRenegotiationIndicationExtension

TScSSLClient.ClientHelloExtensions

5.132.2.11 UseSecureSessionResumption

```
property UseSecureSessionResumption: boolean; default False;
```

Description

Determines whether the TLS session ticket extension will be sent from the client to the TLS/SSL server during the handshake when starting a new TLS/SSL session.

Set **UseSecureSessionResumption** to True, to add the TTLSSessionTicketExtension object to the TScSSLClient.ClientHelloExtensions list. Set **UseSecureSessionResumption** to False, to remove the TTLSSessionTicketExtension instance from the TScSSLClient.ClientHelloExtensions list.

The default value is False.

See also

TTLSSessionTicketExtension

TScSSLClient.ClientHelloExtensions

5.132.2.12 UseSignatureAlgorithmsExtension

```
property UseSignatureAlgorithmsExtension: boolean; default True;
```

Description

Determines if the TLS signature algorithms extension will be sent from the client to the TLS/SSL server during the handshake when starting a new TLS/SSL session.

Set **UseSignatureAlgorithmsExtension** to True, to add the TTLSSignatureAlgorithmsExtension object to the TScSSLClient.ClientHelloExtensions list. Set **UseSignatureAlgorithmsExtension** to False, to remove the TTLSSignatureAlgorithmsExtension instance from the TScSSLClient.ClientHelloExtensions list.

The default value is True.

See also

TTLSSignatureAlgorithmsExtension

TScSSLClient.ClientHelloExtensions

5.133 TScSSLClientOptions

5.133.1 Description

Unit

ScSSLClient

Description

The **TScSSLClientOptions** class determines behaviour of a TLS/SSL connection.

See Also

TScSSLClient.AssignOptions

TScHttpWebRequest.SSLOptions

TScHttpConnectionOptions.SSLOptions

5.133.2 Properties

5.133.2.1 AllowLoadCRLByHttp

```
property AllowLoadCRLByHttp: boolean; default True;
```

Description

Determines whether a certificate revocation list (CRL) will be loaded from an HTTP resource during the TLS/SSL handshake when starting a new session if the required CRL was not found in the current storage.

CRL is used to verify whether the server certificate is valid and has not been revoked by the user. When verifying the validity of an SSL certificate, CRL is searched for in the current storage based on the identifier contained in the certificate itself. If the CRL is not available in the current storage, it can be retrieved from an HTTP resource.

Set **AllowLoadCRLByHttp** to True to allow a download of a CRL list from a URI.

Set **AllowLoadCRLByHttp** to False to disable a download of a CRL list from a URL. If no CRL was found, the TLS server is considered invalid, and the session is closed.

If you do not want to check the revocation status of an SSL certificate, use the DisableCRLValidation property.

The default value is True.

See Also

DisableCRLValidation

5.133.2.2 CACertificateName

```
property CACertificateName: string;
```

Description

Specifies the name of the server CA certificate stored in Storage. The CA certificate is used to authenticate the server through TLS/SSL.

The server sends a certificate to authenticate itself. The certificate must be signed by the specified CA certificate. If the received certificate is not signed by the CA certificate, the Accept parameter will be set to False in the OnServerCertificateValidation event handler. If the server certificate is signed by the CA certificate, Accept will be set to True.

See Also

OnServerCertificateValidation

5.133.2.3 CipherSuites

```
property CipherSuites: TScSSLCipherSuites;
```

Description

Specifies a list of acceptable algorithms for encryption and integrity of data transferred between the client and server through a secure connection.

The algorithms are stored in order of preference.

5.133.2.4 ClientCertificateName

```
property ClientCertificateName: string;
```

Description

Specifies the client certificate name that is stored in Storage. The client certificate is used to authenticate the client by the server. It must be signed by CACertificateName, and must have a private key (TScCertificate.Key.IsPrivate is True).

If the specified certificate was not found in the storage and the TLS/SSL server requires the client

certificate for authentication, secure connection will not be established.

See Also

CACertificateName

5.133.2.5 ClientHelloExtensions

```
property ClientHelloExtensions: TTLSHelloExtensions;
```

Description

Gets a collection of TTLSHelloExtension objects. Use **ClientHelloExtensions[Index]** to obtain a pointer to a specific extension. The `Index` parameter indicates the index of the extension. 0 is the index of the first extension.

This extension list will be sent from client to TLS server during handshake when starting a new TLS session and when requesting session resumption. TLS extensions allow extending the information about client and server certificates, encryption abilities, signature algorithms, etc.

This property is read-only.

See Also

TTLSHelloExtension

5.133.2.6 DisableCRLValidation

```
property DisableCRLValidation: boolean; default False;
```

Description

Determines whether the client will check CRL when verifying the server certificate during the TLS/SSL handshake when starting a new session.

Set **DisableCRLValidation** to True to disable verification of the revocation status of a server certificate, which weakens TLS security.

Set **DisableCRLValidation** to False to enforce verification of the revocation status of a server certificate on the client. If a certificate is revoked or a CRL list is not found, the TLS server is considered invalid, and the session is closed.

The default value is False.

See Also

AllowLoadCRLByHttp

5.133.2.7 IdentityDNSName

```
property IdentityDNSName: string;
```

Description

Determines whether the TLS server name extension will be sent from the client to the TLS/SSL server during the TLS/SSL handshake when starting a new session.

Set **IdentityDNSName** to add the TTLSServerNameExtension object to the TScSSLClient.ClientHelloExtensions list.

See also

TTLSServerNameExtension

TScSSLClient.ClientHelloExtensions

5.133.2.8 IgnoreServerCertificateConstraints

```
property IgnoreServerCertificateConstraints: boolean; default False;
```

Description

Determines whether the server certificate will be verified for compliance with constraints during the TLS/SSL handshake when starting a new session.

Set **IgnoreServerCertificateConstraints** to True, in order for the client to ignore the permission to use a certificate for the required purposes. Set **IgnoreServerCertificateConstraints** to False, in order for the client to check the permission to use a certificate for the required purposes. In this case if the certificate does not match the constraints, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateInsecurity

IgnoreServerCertificateValidity

TrustServerCertificate

TScSSLClient.ClientHelloExtensions

5.133.2.9 IgnoreServerCertificateInsecurity

```
property IgnoreServerCertificateInsecurity: boolean; default False;
```

Description

Determines whether a server certificate signature security will be checked during the TLS/SSL handshake when starting a new session.

Set **IgnoreServerCertificateInsecurity** to True, in order for the client to ignore the certificate signature security. Set **IgnoreServerCertificateInsecurity** to False, in order for the client to check the certificate signature security.

In case if a hash algorithm for signing the certificate is set to one of following algorithms: [haNone, haSHA1, haMD5, haMD4, haMD2], - signature is considered insecured, and therefore, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateConstraints

IgnoreServerCertificateValidity

TrustServerCertificate

TScSSLClient.ClientHelloExtensions

5.133.2.11 IgnoreServerCertificateValidity

```
property IgnoreServerCertificateValidity: boolean; default False;
```

Description

Determines whether a server certificate validity period will be checked during the TLS/SSL handshake when starting a new session.

Set **IgnoreServerCertificateValidity** to True, in order for the client to ignore the certificate validity period. Set **IgnoreServerCertificateValidity** to False, in order for the client to check the certificate validity period. In case if a certificate validity period expires, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateConstraints

IgnoreServerCertificateInsecurity

TrustServerCertificate

TScSSLClient.ClientHelloExtensions

5.133.2.12 Protocols

```
property Protocols: TScSSLProtocols; default [spTls11, spTls12];
```

Description

Specifies the supported security protocols.

The default value is [spTls11, spTls12].

5.133.2.11 Storage

```
property Storage: TScStorage;
```

Description

The **Storage** property is used to access certificate list in the linked storage.

See Also

CACertificateName

ClientCertificateName

5.133.2.12 TrustSelfSignedCertificate

```
property TrustSelfSignedCertificate: boolean; default False;
```

Description

Determines whether to trust a self-signed certificate to authenticate the server during the TLS/SSL handshake when starting a new session.

Set **TrustSelfSignedCertificate** to True to make the client trust a self-signed certificate received from the server.

Set **TrustSelfSignedCertificate** to False to forbid the use of self-signed certificate by the server—in this case, if the server certificate is self-signed, the TLS/SSL server is considered invalid, and the session is closed.

The default value is False.

See Also

IgnoreServerCertificateConstraints

IgnoreServerCertificateInsecurity

IgnoreServerCertificateValidity

TrustStorageCertificates

5.133.2.14 TrustServerCertificate

```
property TrustServerCertificate: boolean; default False;
```

Description

Determines whether the server certificate will be verified during the TLS/SSL handshake when starting a new session.

Set **TrustServerCertificate** to True, in order for the client to trust the server in any case regardless of compliance with the certificate. Set **TrustServerCertificate** to False, in order for the client to verify a certificate. In this case, if the certificate does not meet any requirements, the TLS server is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreServerCertificateConstraints

IgnoreServerCertificateInsecurity

IgnoreServerCertificateValidity

TScSSLClient.ClientHelloExtensions

5.133.2.15 TrustStorageCertificates

```
property TrustStorageCertificates: boolean; default False;
```

Description

Determines whether certificates in the client storage will be trusted to verify the server certificate.

When verifying a certificate, the signature is checked in the entire certificate chain, up to the self-signed certificate. If a non-standard certificate, trusted by the client, was used to sign the certificate, that certificate will be added to the client storage.

Set **TrustStorageCertificates** to True to use the certificates kept in the client storage, in addition to the system certificates, to verify the server certificate.

Set **TrustStorageCertificates** to False to use only the system ROOT certificates to verify the server certificate.

The default value is False.

See also

TrustSelfSignedCertificate

5.133.2.1 UseSecureSessionResumption

```
property UseSecureSessionResumption: boolean; default False;
```

Description

Determines whether the TLS session ticket extension will be sent from the client to the TLS/SSL server during a handshake when starting a new TLS/SSL session.

Set **UseSecureSessionResumption** to True, to add the TTLSSessionTicketExtension object to the TScSSLClient.ClientHelloExtensions list. Set **UseSecureSessionResumption** to False, to remove the TTLSSessionTicketExtension instance from the TScSSLClient.ClientHelloExtensions list.

The default value is False.

See also

TTLSSessionTicketExtension

TScSSLClient.ClientHelloExtensions

5.133.3 Events

5.133.3.1 OnObtainCRL

type

```
  TScObtainCRLEvent = procedure (Sender: TObject; DistributionPointName:
TScGeneralNames; Update: boolean; out CRL: TScCRL) of object;
```

```
property OnObtainCRL: TScObtainCRLEvent;
```

Description

The **OnObtainCRL** event occurs when the server certificate is received from the server and checking if certificate was revoked is performed.

Parameters:

- `Sender` - the object that raised the event;
- `DistributionPointName` - identifies how CRL information for a certificate is obtained;
- `Update` - the boolean parameter indicates, if CRL was expired and need to be reloaded from CRL server. If `Update` is True, CRL need to been reloaded.
- `CRL` - set this out parameter to the found CRL object.

See also

OnServerCertificateValidation

5.133.3.2 OnServerCertificateValidation

```
type
  TScRemoteCertificateValidationEvent = procedure (Sender: TObject;
RemoteCertificate: TScCertificate; CertificateList: TList; var Errors:
TScCertificateStatusSet) of object;

property OnServerCertificateValidation:
TScRemoteCertificateValidationEvent;
```

Description

The **OnServerCertificateValidation** event occurs when the server certificate is received from the server.

When authenticating a TLS/SSL server, from the server comes set of certificates that must be signed by a CA certificate. If received certificate is not signed by the CA certificate, the `Errors` parameter of the **OnServerCertificateValidation** event handler will contain the information about errors. If the server certificate is signed by the CA certificate, the `Errors` set will be empty. A handler of this event can perform additional verifications to authenticate the server. If you trust the server, clear the `Errors` set and the connection will be established.

Parameters:

- `Sender` - the object that raised the event;
- `RemoteCertificate` - the certificate received from the server that identifies this one;
- `CertificateList` - the list of server certificates received from the server;
- `Errors` - TScSSLClient determines the value of the `Errors` parameter and passes it into this event. You can change the `Errors` value within this event handler. If `Errors` is empty, the server is considered valid, and the server authentication is considered successful. Otherwise, the server is considered invalid, and the connection is closed.

See also

OnObtainCRL

5.134 TScSSLClient

5.134.1 Description

Unit

ScSSLClient

Description

TScSSLClient is a component that implements functionality of the TLS/SSL client. **TScSSLClient**

connects to a server supporting the TLS/SSL protocol to which point the HostName and Port properties.

When you connect to a server that supports TLS/SSL, data will be transferred as plaintext until you switch IsSecure to True.

To establish secure connection through TLS/SSL, you can use the following parameters:

- security protocol kind;
- encryption and data integrity algorithms to encrypt the data to be transferred;
- the client certificate.

To exchange data, you should use the ReadBuffer and WriteBuffer methods.

See Also

Connected

IsSecure

5.134.2 Properties

5.134.2.1 BindAddress

```
property BindAddress: string;
```

Description

Determines the TCP/IP address on the local machine as the source address of the connection. Only useful on systems with more than one TCP/IP address.

See Also

HostName

5.134.2.2 CACertName

```
property CACertName: string;
```

Description

Specifies the name of the server CA certificate stored in Storage. The CA certificate is used to authenticate the server through TLS/SSL

The server sends a certificate to authenticate itself. The certificate must be signed by the specified CA certificate. If the received certificate is not signed by the CA certificate, the Accept parameter will be set to False in the OnServerCertificateValidation event handler. If the server certificate is signed by the CA certificate, Accept will be set to True.

If the specified certificate was not found, an exception is raised.

See Also

OnServerCertificateValidation

5.134.2.3 CertName

```
property CertName: string;
```

Description

Specifies the client certificate name that is stored in Storage. The client certificate is used to authenticate the client by the server. It should be signed by CACertName, and must have a private key (TScCertificate.Key.IsPrivate is True).

If the specified certificate was not found in the storage and the TLS/SSL server requires the client certificate for authentication, secure connection will not be established.

Note: If **CertName** is not specified, the certificate is searched by HostName.

See Also

CACertName

5.134.2.4 CipherSuites

```
property CipherSuites: TScSSLCipherSuites;
```

Description

The **CipherSuites** property holds a list of acceptable algorithms that can be used for encrypting and support integrity of the data transferred between the client and the server through a secure connection.

The algorithms are stored in order of preference.

See Also

IsSecure

5.134.2.5 ClientHelloExtensions

```
property ClientHelloExtensions: TTLSHelloExtensions;
```

Description

Holds a collection of TTLSHelloExtension objects. Use **ClientHelloExtensions[Index]** to obtain a pointer to a specific extension. The `Index` parameter indicates the index of an extension. The index of the first extension is 0.

Basic security extensions can be added automatically in this list depending on the SecurityOptions setting.

This extension list will be sent from client to TLS server during handshake when starting a new TLS session and when requesting session resumption. TLS extensions allow extending the information about client and server certificates, encryption abilities, signature algorithms, etc.

This property is read-only.

See Also

TTLSHelloExtension

5.134.2.6 Compression

```
property Compression: TScCompression; default csNone;
```

Description

Determines how data transferred between the client and server will be compressed in the current TLS/SSL session.

Compression is not used by default.

Note: Data compression weakens TLS security, and thus is not recommended. For this very reason TLS 1.3 does not allow data compression.

5.134.2.7 Connected

```
property Connected: Boolean;
```

Description

Determines whether the connection to a TLS/SSL server is established. Switch **Connected** to True, to establish connection to a TLS/SSL server. Switch **Connected** to False, to close the connection.

See Also

Connect

Disconnect

5.134.2.8 EventsCallMode

```
property EventsCallMode: TScEventCallMode; default ecAsynchronous;
```

Description

The **EventsCallMode** property determines how the OnAsyncError and OnAsyncReceive event handlers will be called. The thing is that data coming from the server is processed in a separate thread of the SSL connection. And the event handlers call can occur in a different way to synchronize with the main application thread.

The default value is the `ecAsynchronous` mode when events are added to a queue and then asynchronously synchronized from this queue with the main thread. This allows not slowing down the thread in which events occur and at the same time calling the event handlers in the main thread.

When setting the property to the `ecSynchronous` value, the event call be immediately synchronized with the main thread.

When setting the property to the `ecDirectly` value, there is no synchronization with the main thread.

See Also

OnAsyncReceive

5.134.2.9 HostName

```
property HostName: string;
```

Description

Specifies the host name or the IP addres to connect to the server through TLS/SSL.

The Connect method uses values in the **HostName** and Port properties to establish a connection through TLS/SSL.

See Also

Connected

Port

5.134.2.1(HttpOptions

```
property HttpOptions: THttpOptions;
```

Description

The **HttpOptions** property holds a THttpOptions object that contains settings for HTTP connection.

For more information on HTTP tunneling refer to the Network Tunneling article.

See Also

Connected

5.134.2.11 InCount

```
property InCount: integer;
```

Description

Determines data size received from the server. This data can be obtained using the ReadNoWait or ReadBuffer methods.

See Also

ReadNoWait

5.134.2.12 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is `ivIPv4`.

See Also

TIPVersion

5.134.2.13 IsSecure

```
property IsSecure: Boolean;
```

Description

Determines whether the connection to the TLS/SSL server is protected.

When you connect to a server that supports TLS/SSL, data will be transferred as plaintext until you switch **IsSecure** to True.

See Also

Protocols

5.134.2.14 Port

```
property Port: integer;
```

Description

Specifies the port number for TCP/IP connection to the TLS/SSL server.

The Connect method uses values in the HostName and **Port** properties to establish a connection through TLS/SSL.

See Also

Connected

HostName

5.134.2.15 Protocols

```
property Protocols: TScSSLProtocols; default [spTls11, spTls12];
```

Description

Specifies supported security protocols.

The default value is [spTls11, spTls12].

See Also

IsSecure

5.134.2.16 SecurityOptions

```
property SecurityOptions: TScSSLSecurityOptions;
```

Description

SecurityOptions determines behaviour of an SSL client.

5.134.2.17 ServerHelloExtensions

```
property ServerHelloExtensions: TTLSHelloExtensions;
```

Description

Gets a collection of TTLSHelloExtension objects. Use **ServerHelloExtensions[Index]** to obtain a pointer to a specific extension. The `Index` parameter indicates the index of the extension. 0 is the index of the first extension.

ServerHelloExtensions is set automatically after the session is negotiated. This extension list is sent from TLS server to the client during handshake when starting a new TLS session and when requesting session resumption.

This property is read-only.

See Also

TTLSHelloExtension

5.134.2.15 SessionInfo

```
property SessionInfo: TScSSLSessionInfo;
```

Description

Holds information about the current TLS/SSL session. **SessionInfo** is initialized after the client has been authenticated by the server.

See Also

IsSecure

5.134.2.16 Storage

```
property Storage: TScStorage;
```

Description

The **Storage** property is used to access certificate list in the linked storage. If **Storage** is not assigned, an exception will be raised on connect.

See Also

CACertName

CertName

5.134.2.22 Timeout

```
property Timeout: integer; default 15;
```

Description

Determines the time interval in seconds during which the client will wait for a response from the server when connecting, or wait for data from the server when reading by the ReadBuffer method, or passing data to the server by the WriteBuffer method. After the time has expired, methods return the result and control to the program.

The default value is 15 seconds.

See Also

Connected

ReadBuffer

WriteBuffer

5.134.3 Methods

5.134.3.1 AssignOptions

```
procedure AssignOptions(Options: TScSSLClientOptions);
```

Description

AssignOptions copies the options of the specified `Options` parameter that determine the behaviour of a TLS/SSL client object to the current object.

See Also

Connect

5.134.3.2 AssignSession

```
procedure AssignSession(Source: TScSSLClient);
```

Description

AssignSession copies properties, which describe the current session state to the current object.

The connection to the `Source` object must be established and secure. This method is used to make implementation of renegotiation of session possible.

See Also

Renegotiate

5.134.3.3 Connect

```
procedure Connect;
```

Description

Establishes connection to the specified TLS/SSL server. **Connect** sets the Connected property to True.

See Also

AfterConnect

BeforeConnect

Disconnect

5.134.3.4 Disconnect

```
procedure Disconnect;
```

Description

Closes an existent connection to the TLS/SSL server. **Disconnect** sets the Connected property to False.

See Also

AfterDisconnect

BeforeDisconnect

Connect

5.134.3.5 GetLastException

```
function GetLastException: Exception;
```

Description

Returns the last exception, which occurred during the current SSL/TLS session work.

5.134.3.6 GetLocalIP

```
function GetLocalIP: string;
```

Description

Returns the host IP address for TCP/IP connection with the TLS/SSL server.

See Also

GetLocalPort
GetRemoteIP
GetRemotePort

5.134.3.7 GetLocalPort

```
function GetLocalPort: integer;
```

Description

Returns the host port number for TCP/IP connection with the TLS/SSL server.

See Also

GetLocalIP
GetRemoteIP
GetRemotePort

5.134.3.8 GetRemoteIP

```
function GetRemoteIP: string;
```

Description

Returns the peer IP address for TCP/IP connection with the TLS/SSL server.

See Also

GetLocalIP
GetLocalPort
GetRemotePort

5.134.3.9 GetRemotePort

```
function GetRemotePort: integer;
```

Description

Returns the peer port number for TCP/IP connection with the TLS/SSL server.

See Also

GetLocalIP

GetLocalPort

GetRemoteIP

5.134.3.10 ReadBuffer

```
function ReadBuffer(var Buffer; const Count: integer): integer; overload;
function ReadBuffer(var Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Reads `Count` bytes from the stream of server data into `Buffer`. **ReadBuffer** returns the count of bytes that were actually read.

If the size of the received data is less than `Count` bytes, **ReadBuffer** waits the amount of time specified in Timeout, and then returns control.

See Also

ReadNoWait

Timeout

WriteBuffer

5.134.3.11 ReadNoWait

```
function ReadNoWait(var Buffer; const Count: integer): integer; overload;
function ReadNoWait(var Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Reads the number of bytes equal to or less than specified in `Count` from the stream of server data into `Buffer`, and then immediately returns control to the application. **ReadNoWait** returns the count of bytes that were actually read.

See Also

ReadBuffer

Timeout

WriteBuffer

5.134.3.12 Renegotiate

```
procedure Renegotiate;
```

Description

Call the **Renegotiate** method to request the TLS/SSL client to renegotiate session. This is necessary to improve safety. To avoid rejecting the server, the TTLSRenegotiationIndicationExtension instance should be added to the TScSSLClient.ClientHelloExtensions list before the connection establishing. This extension can be added automatically on switching the SecurityOptions.UseSecureRenegotiation property to True.

See Also

TTLSRenegotiationIndicationExtension

TScSSLSecurityOptions.UseSecureRenegotiation

5.134.3.13 WaitForData

```
function WaitForData(Timeout: integer = -1): boolean;
```

Description

Waits for data from the TLS/SSL server during the period of time specified in `Timeout` (in milliseconds), and then returns control. The calling thread will wait indefinitely when `Timeout` is set to -1.

When data from the server has been received and is available for reading from the buffer, the method returns True. Otherwise, False.

See Also

ReadBuffer

5.134.3.14 WriteBuffer

```
function WriteBuffer(const Buffer; const Count: integer): integer;
overload;
function WriteBuffer(const Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Passes `Count` bytes from `Buffer` through an existing connection. **WriteBuffer** returns the count of bytes that were actually passed.

See Also

ReadBuffer

Timeout

5.134.4 Events

5.134.4.1 AfterConnect

```
property AfterConnect: TNotifyEvent;
```

Description

Occurs after a connection to a TLS/SSL server is established.

See Also

AfterDisconnect

BeforeConnect

BeforeDisconnect

Connected

5.134.4.2 AfterDisconnect

```
property AfterDisconnect: TNotifyEvent;
```

Description

Occurs after the connection to a TLS/SSL server becomes closed.

See Also

AfterConnect

BeforeConnect

BeforeDisconnect

Connected

5.134.4.3 BeforeConnect

```
property BeforeConnect: TNotifyEvent;
```

Description

Occurs immediately before establishing a connection to a TLS/SSL server.

See Also

AfterConnect
AfterDisconnect
BeforeDisconnect
Connected

5.134.4.4 BeforeDisconnect

```
property BeforeDisconnect: TNotifyEvent;
```

Description

Occurs immediately before the connection to a TLS/SSL server becomes closed.

See Also

AfterConnect
AfterDisconnect
BeforeConnect
Connected

5.134.4.5 OnAsyncError

```
type
  TScErrorEvent = procedure (Sender: TObject; E: Exception) of object;

property OnAsyncError: TScErrorEvent;
```

Description

The **OnAsyncError** event occurs when an exception is raised during asynchronous data receiving.

`Sender` is an object that raised the exception.

`E` is the exception object that describes the exception.

See Also

OnAsyncReceive

5.134.4.6 OnAsyncReceive

```
type
  TScAsyncReceiveEvent = procedure(Sender: TObject) of object;

property OnAsyncReceive: TScAsyncReceiveEvent;
```

Description

The **OnAsyncReceive** event occurs if data was received from the server. The data can be read with the ReadBuffer method. The InCount property indicates size of the received data.

See Also

ReadBuffer

5.134.4.7 OnObtainCRL

```
type
  TScObtainCRLEvent = procedure (Sender: TObject; DistributionPointName:
TScGeneralNames; Update: boolean; out CRL: TScCRL) of object;

property OnObtainCRL: TScObtainCRLEvent;
```

Description

The **OnObtainCRL** event occurs when the server certificate is received from the TLS/SSL server and checking if certificate was revoked is performed.

Parameters:

- `Sender` - the object that raised the event;
- `DistributionPointName` - identifies how CRL information for a certificate is obtained;
- `Update` - the boolean parameter indicates, if CRL was expired and need to be reloaded from CRL server. If `Update` is True, CRL need to been reloaded.
- `CRL` - set this out parameter to the found CRL object.

See Also

OnServerCertificateValidation

5.134.4.8 OnServerCertificateValidation

```
type
  TScRemoteCertificateValidationEvent = procedure (Sender: TObject;
RemoteCertificate: TScCertificate; CertificateList: TList; var Errors:
TScCertificateStatusSet) of object;

property OnServerCertificateValidation:
TScRemoteCertificateValidationEvent;
```

Description

The **OnServerCertificateValidation** event occurs when the server certificate is received from the TLS/SSL server.

When authenticating a TLS/SSL server, from the server comes set of certificates that must be signed by a CA certificate. If received certificate is not signed by the CA certificate, the `Errors` parameter of the **OnServerCertificateValidation** event handler will contain the information about errors. If the server certificate is signed by the CA certificate, the `Errors` set will be empty. A handler of this event can perform additional verifications to authenticate the server. If you trust the server, clear the `Errors` set and the connection will be established.

Parameters:

- `Sender` - the object that raised the event;
- `RemoteCertificate` - the certificate received from the server that identifies this one;
- `CertificateList` - the list of server certificates received from the server;
- `Errors` - TScSSLClient determines the value of the `Errors` parameter and passes it into this event. You can change the `Errors` value within this event handler. If `Errors` is empty, the server is considered valid, and the server authentication is considered successful. Otherwise, the server is considered invalid, and the connection is closed.

See Also

IsSecure

5.135 TScSSLServerOptions

5.135.1 Description

Unit

ScSSLServer

Description

The **TScSSLServerOptions** class determines behaviour of a TLS/SSL server connection.

See Also

TScSSLServerConnection.Options

5.135.2 Properties

5.135.2.1 AllowLoadCRLByHttp

```
property AllowLoadCRLByHttp: boolean; default True;
```

Description

Determines whether a certificate revocation list (CRL) will be loaded from an HTTP resource during the TLS/SSL handshake when starting a new session if the required CRL was not found in the current storage.

CRL is used to verify whether the client certificate is valid and has not been revoked by the user. When verifying the validity of an SSL certificate, CRL is searched for in the current storage based on the identifier contained in the certificate itself. If the CRL is not available in the current storage, it can be retrieved from an HTTP resource.

Set **AllowLoadCRLByHttp** to True to allow a download of a CRL list from a URI.

Set **AllowLoadCRLByHttp** to False to disable a download of a CRL list from a URL. If no CRL was found, the TLS client is considered invalid, and the session is closed.

If you do not want to check the revocation status of an SSL certificate, use the DisableCRLValidation property.

The default value is True.

See Also

DisableCRLValidation

5.135.2.2 AsyncStartTLS

```
property AsyncStartTLS: boolean; default False;
```

Description

Determines whether the server will wait till the TLS/SSL handshake for a new session is complete, or whether the control should be returned to the user immediately after creating a socket connection.

Set **AsyncStartTLS** to True to return control to the server immediately after creating a socket connection, without waiting to a TSL connection. A TLS connection will be established

asynchronously in a separate thread.

Set **AsyncStartTLS** to False to to return control to the server only after the TLS/SSL handshake is complete.

The default value is False.

5.135.2.3 ClientInitiatedRenegotiationIsAllowed

```
property ClientInitiatedRenegotiationIsAllowed: boolean; default False;
```

Description

Determines whether the server will allow client-initiated renegotiation.

Set **ClientInitiatedRenegotiationIsAllowed** to True to allow the client to renegotiate new encryption parameters. Note that the TLS/SSL renegotiation makes it easy to carry out a DoS attack by repeatedly initiating renegotiation, causing the server to waste resources.

Set **ClientInitiatedRenegotiationIsAllowed** to False to forbid the client to renegotiate new encryption parameters. If the client sends a renegotiation request, the server will deny it and close the connection. This behavior may protect against DoS attacks.

The default value is False.

5.135.2.4 DisableCloseSocketOnShutdownAlert

```
property DisableCloseSocketOnShutdownAlert: boolean; default False;
```

Description

Determines whether a connection with the client will be closed when the client requests to shutdown the TLS/SSL session.

Set **DisableCloseSocketOnShutdownAlert** to True to keep the connection alive when the client requests to shutdown the TLS/SSL session. The TLS connection will be closed, but the socket connection will remain open, and data will be transferred unencrypted between the client and server.

Set **DisableCloseSocketOnShutdownAlert** to False to close the connection when the client requests to shutdown the TLS/SSL session.

The default value is False.

5.135.2.5 DisableCRLValidation

```
property DisableCRLValidation: boolean; default False;
```

Description

Determines whether the server will check CRL when verifying the client certificate during the TLS/SSL handshake when starting a new session.

Set **DisableCRLValidation** to True to disable verification of the revocation status of a client certificate, which weakens TLS security.

Set **DisableCRLValidation** to False to enforce verification of the revocation status of a client certificate on the server. If a certificate is revoked or a CRL list is not found, the TLS client is considered invalid, and the session is closed.

The default value is False.

See Also

AllowLoadCRLByHttp

5.135.2.6 ExtendedMasterSecretMode

```
property ExtendedMasterSecretMode: TScExtendedMasterSecretMode; default
emsmAllow;
```

Description

Determines whether Extended Master Secret TLS Extension will be used to compute the master secret. The extended mode is available in TLS1.0, TLS1.1, and TLS1.2.

The client initiates the use of this mode. The server may allow or forbid the extended mode for the secret key.

The default value is emsmAllow.

See Also

TScExtendedMasterSecretMode

5.135.2.7 IgnoreClientCertificateConstraints

```
property IgnoreClientCertificateConstraints: boolean; default False;
```

Description

Determines whether the client certificate will be verified for compliance with constraints during the TLS/SSL handshake when starting a new session.

Set **IgnoreClientCertificateConstraints** to True to make the server ignore the permission to use the client certificate for the required purposes.

Set **IgnoreClientCertificateConstraints** to False to make the server ignore for the server to check the permission to use a certificate for the required purposes. If the certificate does not match the constraints, the TLS client is considered invalid, and the session is closed.

The default value is False.

See also

IgnoreClientCertificateInsecurity

IgnoreClientCertificateValidity

IsClientCertificateRequired

5.135.2.8 IgnoreClientCertificateInsecurity

```
property IgnoreClientCertificateInsecurity: boolean; default False;
```

Description

Determines whether the signature security of the client certificate will be verified during the TLS/SSL handshake when starting a new session.

Set **IgnoreClientCertificateInsecurity** to True to make the server ignore the client certificate signature security.

Set **IgnoreClientCertificateInsecurity** to False to make the server verify the signature security of the client certificate.

If the hash algorithm for signing the certificate is set to one of following: [haNone, haSHA1, haMD5, haMD4, haMD2],— the signature is considered insecure. Thus, the TLS client is considered invalid, and the session is closed.

The default value is False.

See Also

IgnoreClientCertificateConstraints

IgnoreClientCertificateValidity

IsClientCertificateRequired

5.135.2.9 IgnoreClientCertificateValidity

```
property IgnoreClientCertificateValidity: boolean; default False;
```

Description

Determines whether the validity period of the client certificate will be verified during the TLS/SSL handshake when starting a new session.

Set **IgnoreClientCertificateValidity** to True to make the server ignore the validity period of the client certificate.

Set **IgnoreClientCertificateValidity** to False to make the server verify the certificate validity period. If the the validity period of the certificate has expired, the TLS is considered invalid, and the session is closed.

The default value is False.

See Also

IgnoreClientCertificateConstraints

IgnoreClientCertificateInsecurity

IsClientCertificateRequired

5.135.2.1 IsClientCertificateRequired

```
property IsClientCertificateRequired: boolean; default False;
```

Description

Determines whether the client certificate is required to establish a TLS connection.

Set **IsClientCertificateRequired** to True to require a certificate to authenticate the client. If the client does not send a certificate to the sever or sends an invalid certificate, the TLS client is considered invalid, and the session is closed.

Set **IsClientCertificateRequired** to False to not request the client certificate during the TLS/SSL handshake when starting a new session. The client should be authenticated on a higher level in the application in this case.

The default value is False.

5.135.2.1 NewSessionTicketDistributedCount

```
property NewSessionTicketDistributedCount: integer; default 2;
```

Description

Determines the number of New Session Tickets that the server will issue during the TLS/SSL handshake when starting a new session.

A New Session Ticket creates a unique association between the ticket value and the session key which is used to generate encryption keys, decrypt the data transmitted, and verify data integrity.

The TLS client may use these a New Session Ticket for session resumption, which greatly improves the performance of the client and server when multiple parallel or serial connections are used between the same client and server.

When **NewSessionTicketDistributedCount** is less than or equal to 0, the server will not issue any New Session Ticket, which makes session resumption impossible.

The default value is 2 tickets.

See Also

NewSessionTicketLifetime

5.135.2.11 NewSessionTicketLifetime

```
property NewSessionTicketLifetime: integer; default 600;
```

Description

Determines the lifetime of New Session Tickets that are issued by the server during the TLS/SSL handshake when starting a new session.

NewSessionTicketDistributedCount holds the number of tickets to be issued.

The TLS client uses a New Session Ticket for session resumption. During session resumption, the server checks if the ticket has expired, in which case a new TLS/SSL session will be initiated.

The default value is 600 seconds.

See Also

NewSessionTicketDistributedCount

5.135.2.12 RecordSizeLimit

```
property RecordSizeLimit: integer;
```

Description

Determines the maximum size (in bytes) of the record layer packet transferred between the client and server. The server will split the data into packets that do not exceed that maximum allowed size. The server also verifies that packets received from the client do not exceed the maximum size. If a packet exceeds the maximum size, it is considered invalid, and the server closes the session.

When **RecordSizeLimit** is set to 0, standard TLS packet size (2^14 bytes) will be applied.

5.135.2.13 TrustSelfSignedCertificate

```
property TrustSelfSignedCertificate: boolean; default False;
```

Description

Determines whether a self-signed certificate will be trusted to authenticate the client during the TLS/SSL handshake when starting a new session.

Set **TrustSelfSignedCertificate** to True to make the server trust the self-signed certificate received from the client.

Set **TrustSelfSignedCertificate** to False to forbid the use of a self-signed certificate. If the TLS client sends a self-signed certificate, the client is considered invalid, and the session is closed.

The default value is False.

See Also

IgnoreClientCertificateConstraints

IgnoreClientCertificateInsecurity

IgnoreClientCertificateValidity

IsClientCertificateRequired

TrustStorageCertificates

5.135.2.1 TrustStorageCertificates

```
property TrustStorageCertificates: boolean; default False;
```

Description

Determines whether the certificates in the server storage will be trusted to verify the client certificate.

When verifying a certificate, the signature is checked in the entire certificate chain, up to the self-signed certificate. If a non-standard certificate, trusted by the server, was used to sign the certificate, that certificate will be added to the server storage.

Set **TrustStorageCertificates** to True to use the certificates in the storage storage, in addition to the system certificates, to verify the client certificate.

Set **TrustStorageCertificates** to False to use only the system ROOT certificates to verify the client certificate.

The default value is False.

See Also

IsClientCertificateRequired

TrustSelfSignedCertificate

5.136 TScSSLServerConnection

5.136.1 Description

Unit

ScSSLServer

Description

TScSSLServerConnection is a component that implements functionality of the TLS/SSL server. **TScSSLServerConnection** connects to a client supporting the TLS/SSL protocol.

Before establishing a connection, you should initialize the server by calling the Init method and passing the input and output sources for the connection.

A connection is opened by calling the Connect method.

When you connect to a client that supports TLS/SSL, data will be transferred as plaintext form unless you set the IsSecure property to True.

To establish a secure connection through TLS/SSL, you can use the following parameters:

- the type of the security protocol;
- the encryption and data integrity algorithms to encrypt the data to be transferred;
- the chain of the server certificate names (CertNameChain).

You use the ReadBuffer and WriteBuffer methods to exchange data.

See Also

Connected

Init

IsSecure

5.136.2 Properties

5.136.2.1 CACertName

```
property CACertName: string;
```

Description

Specifies the name of the client CA certificate from the Storage. The CA certificate is used to authenticate the client through TLS/SSL.

The client sends a certificate to authenticate itself. The certificate must be signed by the specified CA certificate. If the received certificate is not signed by the CA certificate, the Accept parameter will be set to False in the OnClientCertificateValidation event handler. If the client certificate is signed by the CA certificate, Accept will be set to True.

If **CACertName** is not specified, a list of system CA certificates will be used.

See Also

OnClientCertificateValidation

5.136.2.2 CertNameChain

```
property CertNameChain: string;
```

Description

Specifies the chain of the server certificate names from the Storage. The server certificate is used by

the client to authenticate the server.

The list of certificates should be delimited with a semicolon and contain the entire chain of certificates, starting with the certificate that authenticates the identity of the server, except for the root self-signed certificate which should be either installed in the OS or specified in the CACertName property.

If the specified certificates were not found in the storage, and the TLS/SSL client requires the server certificate for authentication, a secure connection will not be established.

You can verify a chain of the server certificate names using the CheckCertificateChain method.

See Also

CACertName

CheckCertificateChain

5.136.2.3 CipherSuites

```
property CipherSuites: TScSSLCipherSuites;
```

Description

Specifies a list of acceptable algorithms for encryption and integrity of data transferred between the client and server through a secure connection.

The algorithms are stored in order of preference.

See Also

IsSecure

5.136.2.4 ClientHelloExtensions

```
property ClientHelloExtensions: TTLSHelloExtensions;
```

Description

Holds a collection of TTLSHelloExtension objects. Use **ClientHelloExtensions[Index]** to obtain a pointer to a specific extension. The `Index` parameter indicates the index of an extension. The index of the first extension is 0.

ClientHelloExtensions is set automatically after the session has been negotiated. The client sends the extension list to the server during the TLS/SSL handshake when starting a new session or requesting session resumption.

TLS extensions allow extending the information about the client and server certificates, encryption capabilities, signature algorithms, etc.

The list of extensions received from the client during a handshake can be verified using the AfterClientHello event.

This property is read-only.

See Also

AfterClientHello

ServerHelloExtensions

TTLSHelloExtension

5.136.2.5 Compression

```
property Compression: TScCompression; default csNone;
```

Description

Determines how data transferred between the client and server will be compressed in the current TLS/SSL session. Compression is not used by default.

Note: Data compression weakens TLS security, and thus is not recommended. Therefore TLS 1.3 does not allow data compression.

5.136.2.6 Connected

```
property Connected: boolean;
```

Description

Determines whether a connection to the TLS/SSL client has been established.

Set **Connected** to True to open a connection to the TLS/SSL client.

Set **Connected** to False to close the connection.

See Also

Connect

Disconnect

5.136.2.7 EventsCallMode

```
property EventsCallMode: TScEventCallMode; default ecAsynchronous;
```

Description

The **EventsCallMode** property determines how the OnAsyncReceiveData and OnAsyncError event handlers will be called. The thing is that data coming from the client is processed in a separate

thread of the TLS connection. And the event handlers call can occur in a different way to synchronize with the main application thread.

The default value is the `ecAsynchronous` mode when events are added to a queue and then asynchronously synchronized from this queue with the main thread. This allows not slowing down the thread in which events occur and at the same time calling the event handlers in the main thread.

When setting the property to the `ecSynchronous` value, the event call be immediately synchronized with the main thread.

When setting the property to the `ecDirectly` value, there is no synchronization with the main thread.

See Also

OnAsyncReceiveData

5.136.2.8 IsSecure

```
property IsSecure: boolean;
```

Description

Determines whether the connection to the TLS/SSL client is protected.

When you connect to a client that supports TLS/SSL, data will be transferred as plaintext until you set **IsSecure** to True.

See Also

Protocols

SessionInfo

5.136.2.9 Options

```
property Options: TScSSLServerOptions;
```

Description

Determines behaviour of the TLS/SSL server connection.

5.136.2.10 Protocols

```
property Protocols: TScSSLProtocols; default [spTls12, spTls13];
```

Description

Specifies supported security protocols. The default value is [spTls12, spTls13].

See Also

IsSecure

5.136.2.11 RequestDNList

```
property RequestDNList: TScDistinguishedNameList;
```

Description

Specifies a list of Distinguished Names (DN) that the server expects in the issuer field of the client certificate. The server sends **RequestDNList** to the client in the certificate request.

You can use the IsClientCertificateRequired property to specify whether the client certificate is required to establish a TLS session. When IsClientCertificateRequired is set to False, **RequestDNList** has no effect.

5.136.2.12 ServerHelloExtensions

```
property ServerHelloExtensions: TTLSHelloExtensions;
```

Description

Holds a collection of TTLSHelloExtension objects. Use **ServerHelloExtensions[Index]** to obtain a pointer to a specific extension. The `Index` parameter indicates the index of an extension. The index of the first extension is 0.

The server sends the extension list to the client during the TLS/SSL handshake when starting a new session or requesting session resumption. TLS extensions allow extending the information about the client and server certificates, encryption capabilities, signature algorithms, etc.

See Also

ClientHelloExtensions

TTLSHelloExtension

5.136.2.13 SessionInfo

```
property SessionInfo: TScSSLSessionInfo;
```

Description

Holds information about the current TLS/SSL session. **SessionInfo** is initialized after the client has

been authenticated by the server.

See Also

IsSecure

5.136.2.14 Storage

```
property Storage: TScStorage;
```

Description

The **Storage** property is used to access certificate list in the linked storage. If **Storage** is not assigned, an exception is raised on connection attempt.

See Also

CACertName

CertNameChain

5.136.2.15 SupportedKExNamedGroups

```
property SupportedKExNamedGroups: TScKExNamedGroupTypes;
```

Description

Specifies a set of named groups which can be used for key exchange during the TLS/SSL handshake when starting a new session.

The default value = [

kex_x25519,

kex_secp256r1, kex_secp256k1, kex_secp384r1, kex_secp521r1,

kex_sect283r1, kex_sect283k1, kex_sect409r1, kex_sect409k1,

kex_sect571r1, kex_sect571k1,

kex_ffDHE2048, kex_ffDHE3072, kex_ffDHE4096,

kex_ffDHE6144, kex_ffDHE8192

];

See Also

IsSecure

5.136.2.1 Timeout

```
property Timeout: integer; default 15;
```

Description

Determines the time to wait for a response from the client when connecting, the time to wait for data from the client when using the ReadBuffer method, or the time to wait when passing data to the client using the WriteBuffer method. After the timeout expires, the methods return the result and control to the program.

The default value is 15 seconds.

See Also

Connected

ReadBuffer

WriteBuffer

5.136.3 Methods

5.136.3.1 CheckCertificateChain

```
procedure CheckCertificateChain(out StatusSet: TScCertificateStatusSet);
overload;
procedure CheckCertificateChain; overload;
```

Description

Verifies the chain of the server certificate names.

The method parses the list of certificates in CertNameChain, then checks the availability of these certificates in Storage.

After that, the chain of certificates is validated against the key requirements of the TLS protocol.

If any errors are found, they are returned in the `StatusSet` parameter. When the method is called without parameters and errors are found during validation, an appropriate exception is generated.

See Also

CertNameChain

Storage

5.136.3.2 Connect

```
procedure Connect;
```

Description

Opens a connection to the specified TLS/SSL client and sets the Connected property to True.

Before creating a connection, you should initialize the server by calling the Init method and passing the input and output sources for the connection. If the server is not initialized, an appropriate error will occur.

See Also

AfterConnect

BeforeConnect

Disconnect

Init

5.136.3.3 Disconnect

```
procedure Disconnect;
```

Description

Closes an existing connection with the TLS/SSL client and sets the Connected property to False.

See Also

AfterDisconnect

BeforeDisconnect

Connect

5.136.3.4 GetLastException

```
function GetLastException: Exception;
```

Description

Returns the last exception, which occurred during the current SSL/TLS session work.

5.136.3.5 Init

```
type
  TScReceiveDataFromSourceEvent = function (Sender: TObject; var Data;
Offset, Count: integer): integer of object;
```

```
  TScSendDataToTargetEvent = procedure (Sender: TObject; const Data;
Offset, Count: integer) of object;
```

```
procedure Init(Connection: TScTCPConnection); overload;
procedure Init(OnConnect: TNotifyEvent; OnReceiveDataFromSource:
TScReceiveDataFromSourceEvent; OnSendDataToTarget:
TScSendDataToTargetEvent); overload;
```

Description

Initializes a connection by passing the input and output sources for the connection.

You can get data either through a socket-based connection (`TScTCPConnection`) or using events for reading and sending data. **Init** is an overloaded method and allows getting data both ways.

Init should be called before connecting to the client.

Parameters:

- `Connection` - an initialized TScTCPConnection that enables the server to connect to the client directly through a socket connection. The server calls the methods of the connection object to send and receive data.
- `OnConnect` - the event hanlder that is called when a connection to the client has been established.
- `OnReceiveDataFromSource` - the event hander that is called when the server needs to get data from the client. The event accepts the pointer to the data and its size.
- `OnSendDataToTarget` - the event handler that is called when the server needs to send data to the client. The event accepts the pointer to the data and its size.

See Also

Connect

OnReceiveDataFromSource

OnSendDataToTarget

5.136.3.6 ReadBuffer

```
function ReadBuffer(var Buffer; const Count: integer): integer; overload;
function ReadBuffer(var Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Reads `Count` bytes from the stream of client data into `Buffer`. **ReadBuffer** returns the count of bytes that were actually read.

If the size of the received data is less than `Count` bytes, **ReadBuffer** waits the amount of time specified in the Timeout property, and then returns control.

See Also

ReadNoWait

Timeout

WriteBuffer

5.136.3.7 ReadNoWait

```
function ReadNoWait(var Buffer; const Count: integer): integer; overload;
function ReadNoWait(var Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Reads the number of bytes equal to or less than specified in `Count` from the stream of client data into `Buffer`, and then immediately returns control to the application. **ReadNoWait** returns the count of bytes that were actually read.

See Also

ReadBuffer

Timeout

WriteBuffer

5.136.3.8 WaitForData

```
function WaitForData(Timeout: integer = -1): boolean;
```

Description

Waits for data from the TLS/SSL client during the period of time specified in `Timeout` (in milliseconds), and then returns control. The calling thread will wait indefinitely when `Timeout` is set to -1.

When data from the client has been received and is available for reading from the buffer, the method returns True. Otherwise, False.

See Also

ReadBuffer

5.136.3.9 WriteBuffer

```
function WriteBuffer(const Buffer; const Count: integer): integer;
overload;
function WriteBuffer(const Buffer: TBytes; const Offset, Count: integer):
integer; overload;
```

Description

Passes `Count` bytes from `Buffer` through an existing connection. **WriteBuffer** returns the count of bytes that were actually passed.

See Also

ReadBuffer

Timeout

5.136.4 Events

5.136.4.1 AfterClientHello

```
type
  TScAfterClientHelloMessageEvent = procedure (Sender: TObject; var
Cancel: boolean) of object;

property AfterClientHello: TScAfterClientHelloMessageEvent;
```

Description

Occurs after receiving a "Client Hello" message from the client during the TLS/SSL handshake when starting a new session.

A handler of this event can verify the list of ClientHelloExtensions.

If the received extensions or any other properties do not comply with the server requirements, the session may be closed by setting the `Cancel` parameter to False.

See Also

ClientHelloExtensions

Connected

5.136.4.2 AfterConnect

```
property AfterConnect: TNotifyEvent;
```

Description

Occurs after a connection to the TLS/SSL client has been established.

See Also

AfterDisconnect
BeforeConnect
BeforeDisconnect
Connected

5.136.4.3 AfterDisconnect

```
property AfterDisconnect: TNotifyEvent;
```

Description

Occurs after the connection to the TLS/SSL client has been closed.

See Also

AfterConnect
BeforeConnect
BeforeDisconnect
Connected

5.136.4.4 BeforeConnect

```
property BeforeConnect: TNotifyEvent;
```

Description

Occurs immediately before establishing a connection to the TLS/SSL client.

See Also

AfterConnect
AfterDisconnect
BeforeDisconnect
Connected

5.136.4.5 BeforeDisconnect

```
property BeforeDisconnect: TNotifyEvent;
```

Description

Occurs immediately before the connection to the TLS/SSL client is closed.

See Also

AfterConnect
AfterDisconnect
BeforeConnect
Connected

5.136.4.6 OnAsyncError

```
type
  TScErrorEvent = procedure (Sender: TObject; E: Exception) of object;

property OnAsyncError: TScErrorEvent;
```

Description

Occurs when an exception is raised during asynchronous data receiving.

`Sender` is the object that raised the exception.

`E` is the exception object that describes the exception.

See Also

OnAsyncReceiveData

5.136.4.7 OnAsyncReceiveData

```
type
  TScAsyncReceiveEvent = procedure(Sender: TObject) of object;

property OnAsyncReceiveData: TScAsyncReceiveEvent;
```

Description

Occurs when data has been received from the client. The data can be read using the ReadBuffer

method.

See Also

ReadBuffer

5.136.4.8 OnClientCertificateValidation

type

```
  TScRemoteCertificateValidationEvent = procedure (Sender: TObject;
RemoteCertificate: TScCertificate; CertificateList: TList; var Errors:
TScCertificateStatusSet) of object;

property OnClientCertificateValidation:
TScRemoteCertificateValidationEvent;
```

Description

Occurs when the client certificate is received from the TLS/SSL client.

During authentication the TLS/SSL client should send a set of certificates signed by a CA certificate. If the received certificate is not signed by the CA certificate, the `Errors` parameter of the **OnClientCertificateValidation** event handler will contain information about the errors. If the client certificate is signed by the CA certificate, the `Errors` set will be empty. A handler of this event can perform additional verifications to authenticate the client. If you trust the client, clear the `Errors` set, and a connection will be established.

You can specify whether the client certificate is required to create a TLS connection in the IsClientCertificateRequired property.

Parameters:

- `Sender` - the object that raised the event;
- `RemoteCertificate` - the certificate received from the client;
- `CertificateList` - the list of client certificates received from the client;
- `Errors` - TScSSLServerConnection determines the value of the `Errors` parameter and passes it into this event. You can change the `Errors` value within this event handler. If `Errors` is empty, the client is considered valid, and the client authentication is considered successful. Otherwise, the client is considered invalid, and the connection is closed.

See Also

IsSecure

5.136.4.9 OnCreateNewSessionTicket

```
type
  TScCreateNewSessionTicketEvent = procedure (Sender: TObject;
SessionInfo: TScSSLSessionInfo; NewSessionTicket: TScNewSessionTicket) of
object;

property OnCreateNewSessionTicket: TScCreateNewSessionTicketEvent;
```

Description

Occurs when a "New Session Ticket" message is sent during the TLS/SSL handshake when starting a new session. You should initialize a `NewSessionTicket` object in this event handler.

A New Session Ticket creates a unique association between the ticket value and the session key which is used to generate encryption keys, decrypt data, and verify data integrity. The TLS/SSL client may use this object for session resumption.

A New Session Ticket can be initialized automatically or manually in the **OnCreateNewSessionTicket** event. When **OnCreateNewSessionTicket** is not set, a ticket will be initialized automatically. To initialize a ticket manually, you should handle **OnCreateNewSessionTicket**—in this case, the OnDecodeTicket event must be handled to decode the ticket.

See Also

OnDecodeTicket

OnGetPasswordByTicketName

OnGetTicketNameAndPassword

Options.NewSessionTicketDistributedCount

5.136.4.10 OnDecodeTicket

```
type
  TScDecodeTicketEvent = procedure (Sender: TObject; NewSessionTicket:
TScNewSessionTicket; NewSessionInfo: TScSSLSessionInfo; var IsValid:
boolean) of object;

property OnDecodeTicket: TScDecodeTicketEvent;
```

Description

Occurs when the client resumes a session using New Session Ticket to decode a ticket. The data of `NewSessionTicket` should be decoded in the event handler, and appropriate properties should be set in the `NewSessionInfo` object. These properties will be used for session resumption.

A New Session Ticket creates a unique association between the ticket value and the session key which is used to generate encryption keys, decrypt data, and verify data integrity. It can be initialized automatically or manually in the OnCreateNewSessionTicket event. When OnCreateNewSessionTicket is not set, a ticket will be initialized automatically. To initialize a ticket manually, you should handle OnCreateNewSessionTicket—in this case, the **OnDecodeTicket** event must be handled to decode the ticket.

See Also

OnCreateNewSessionTicket

OnGetPasswordByTicketName

OnGetTicketNameAndPassword

Options.NewSessionTicketDistributedCount

5.136.4.11 OnGetPasswordByTicketName

type

```
  TScGetPasswordByTicketNameEvent = procedure (Sender: TObject; const
Ticket: TBytes; out Password: TBytes) of object;

property OnGetPasswordByTicketName: TScGetPasswordByTicketNameEvent;
```

Description

Occurs when the client resumes a session using New Session Ticket to get the password that was used to encrypt the contents of the ticket.

A New Session Ticket can be initialized automatically or manually in the OnCreateNewSessionTicket event. When OnCreateNewSessionTicket is not set, a ticket will be initialized automatically.

When a ticket is created automatically, the contents of the ticket should be encrypted. If a user wants to set its own password, it can do it the event handler OnGetTicketNameAndPassword. An **OnGetPasswordByTicketName** event, which will be triggered to decode the session state, should return `Password` for `Ticket`.

See Also

OnCreateNewSessionTicket

OnDecodeTicket

OnGetTicketNameAndPassword

Options.NewSessionTicketDistributedCount

5.136.4.12 OnGetTicketNameAndPassword

type

```
  TScGetTicketNameAndPasswordEvent = procedure (Sender: TObject; out
TicketName, Password: TBytes) of object;

property OnGetTicketNameAndPassword: TScGetTicketNameAndPasswordEvent;
```

Description

Occurs when a "New Session Ticket" message is sent to the client during the TLS/SSL handshake to get `TicketName` and `Password` that will be used to encrypt the contents of the ticket.

A New Session Ticket can be initialized automatically or manually in the OnCreateNewSessionTicket event. When OnCreateNewSessionTicket is not set, a ticket will be initialized automatically. When a ticket is created automatically, the contents of the ticket should be encrypted. If a user wants to set their own password, they can do it the event handler **OnGetTicketNameAndPassword**. An OnGetPasswordByTicketName event, which will be triggered to decode the session state, should return `Password` for `Ticket`.

See Also

OnCreateNewSessionTicket

OnDecodeTicket

OnGetPasswordByTicketName

Options.NewSessionTicketDistributedCount

5.136.4.13 OnObtainCRL

```
type
  TScObtainCRLEvent = procedure (Sender: TObject; DistributionPointName:
TScGeneralNames; Update: boolean; out CRL: TScCRL) of object;

property OnObtainCRL: TScObtainCRLEvent;
```

Description

Occurs when the server verifies the revocation status of the received client certificate. The event is intended to get a CRL by its `DistributionPointName`.

Parameters:

- `Sender` - the object that raised the event;
- `DistributionPointName` - identifies how CRL information for the certificate is obtained;
- `Update` - the boolean parameter indicates whether CRL has expired and needs to be reloaded from the CRL server. When `Update` is True, CRL needs to been reloaded.
- `CRL` - set this out parameter to the found CRL object.

See Also

OnClientCertificateValidation

5.136.4.14 OnReceiveDataFromSource

type

```
  TScReceiveDataFromSourceEvent = function (Sender: TObject; var Data;
Offset, Count: integer): integer of object;

property OnReceiveDataFromSource: TScReceiveDataFromSourceEvent;
```

Description

Occurs when the server gets data from the client. The event parameters hold the pointer to the data and its size.

The event is only triggered if a TScTCPConnection object, which enables the server to directly access the client through a socket connection, has not been specified during initialization of TScSSLServerConnection. In this case, the user should get data from the client and write it to the buffer specified in **OnReceiveDataFromSource**.

See Also

Init

OnSendDataToTarget

5.136.4.15 OnSendDataToTarget

type

```
  TScSendDataToTargetEvent = procedure (Sender: TObject; const Data;
Offset, Count: integer) of object;

property OnSendDataToTarget: TScSendDataToTargetEvent;
```

Description

Occurs when the server sends data to the client. The event parameters hold the pointer to the data and its size.

The event is only triggered if a TScTCPConnection object, which enables the server to directly access the client through a socket connection, has not been specified during initialization of TScSSLServerConnection. In this case, the user should send the data specified in the parameters of **OnSendDataToTarget**, to the client.

See Also

Init

OnReceiveDataFromSource

5.137 TScTCPConnection

5.137.1 Description

Unit

ScUtils

Description

The **TScTCPConnection** class implements functionality of a TCP/IP connection. **TScTCPConnection** can connect to the TCP/IP server identified by Host and Port.

You use the Read and Write methods to exchange data.

See Also

TScSSLServerConnection.Init

5.137.2 Properties

5.137.2.1 BindAddress

```
property BindAddress: string;
```

Description

Determines the TCP/IP address on the local machine as the source address of the connection. Only has effect on systems with more than one TCP/IP address.

See Also

Host

5.137.2.2 Connected

```
property Connected: boolean;
```

Description

Determines whether a connection to the TCP/IP server has been established.

Set **Connected** to True to establish a connection to the TCP/IP server.

Set **Connected** to False to close the connection.

See Also

Connect

Close

5.137.2.3 ConnectionTimeout

```
property ConnectionTimeout: integer;
```

Description

Determines the time in seconds that the client will wait for a response from the remote side during connection attempt. After the timeout expires, the result and control will be returned to the program.

The default value is 30 seconds.

See Also

Connected

5.137.2.4 Host

```
property Host: string;
```

Description

Specifies the hostname or IP address of the TCP/IP server.

The Connect method uses values in the **Host** and Port properties to establish a connection for the TCP/IP session.

See Also

Connected

Port

5.137.2.5 IPVersion

```
property IPVersion: TIPVersion;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is `ivIPv4`.

See Also

TIPVersion

5.137.2.6 LocalSockAddr

```
property LocalSockAddr: PSockAddr;
```

Description

The **LocalSockAddr** property represents the information in the WinSocket format about the local socket used for data exchange.

This property is read-only.

5.137.2.7 Port

```
property Port: integer;
```

Description

Specifies the port number for a TCP/IP connection.

The Connect method uses values in the Host and **Port** properties to establish a connection for the TCP/IP session.

See Also

Connected

Host

5.137.2.8 ReceiveTimeout

```
property ReceiveTimeout: integer;
```

Description

Determines the time in seconds that the client will wait for a response from the remote side when reading with the Read method. After the timeout expires, the result and control are returned to the program.

The default value is 30 seconds.

See Also

Read

5.137.2.9 RemoteSockAddr

```
property RemoteSockAddr: PSockAddr;
```

Description

Holds the information in the WinSocket format about the peer socket used for data exchange.

This property is read-only.

5.137.2.10 SendTimeout

```
property SendTimeout: integer;
```

Description

Determines the time in seconds that the client will wait for a response when passing data to the remote side with the Write method. After the timeout expires, the result and control are returned to the program.

The default value is 30 seconds.

See Also

Write

5.137.3 Methods

5.137.3.1 Bind

```
procedure Bind;
```

Description

Binds a socket to the BindAddress on the local machine.

See Also

BindAddress

5.137.3.2 Close

```
procedure Close;
```

Description

Closes an existing connection to the remote side and sets the Connected property to False.

See Also

Connect

5.137.3.3 Connect

```
procedure Connect;
```

Description

Opens a connection to the specified TCP/IP server and sets the Connected property to True.

See Also

Close

5.137.3.4 GetLocalIP

```
function GetLocalIP: string;
```

Description

Returns the host IP address for a TCP/IP connection with the remote side.

See Also

GetLocalPort
GetRemoteIP
GetRemotePort

5.137.3.5 GetLocalPort

```
function GetLocalPort: integer;
```

Description

Returns the host port number for a TCP/IP connection with the remote side.

See Also

GetLocalIP
GetRemoteIP
GetRemotePort

5.137.3.6 GetRemoteIP

```
function GetRemoteIP: string;
```

Description

Returns the peer IP address for a TCP/IP connection with the remote side.

See Also

GetLocalIP
GetLocalPort
GetRemotePort

5.137.3.7 GetRemotePort

```
function GetRemotePort: integer;
```

Description

Returns the peer port number for a TCP/IP connection with the remote side.

See Also

GetLocalIP
GetLocalPort
GetRemoteIP

5.137.3.8 Read

```
function Read(var Buffer; Count: integer): integer; overload;
function Read(var Buffer: TBytes; Offset, Count: integer): integer;
overload;
```

Description

Reads `Count` bytes from the stream of peer data into `Buffer`. **Read** returns the count of bytes that

were actually read.

If the size of the received data is less than `Count` bytes, **Read** waits the amount of time specified in the ReceiveTimeout property, and then returns control.

See Also

ReceiveTimeout

Write

5.137.3.9 WaitForData

```
function WaitForData(Timeout: integer = -1): boolean;
```

Description

Waits for data from the remote side during the period of time specified in `Timeout` (in milliseconds), and then returns control. The calling thread will wait indefinitely when `Timeout` is set to -1.

When data from the remote side has been received and is available for reading from the buffer, the method returns True. Otherwise, False.

See Also

Read

5.137.3.10 Write

```
function Write(const Buffer; Count: integer): integer; overload;
function Write(const Buffer: TBytes; Offset, Count: integer): integer;
overload;
```

Description

Passes `Count` bytes from `Buffer` through an existing connection. **Write** returns the count of bytes that were actually passed.

See Also

Read

SendTimeout

5.137.4 Events

5.137.4.1 OnClose

```
property OnClose: TNotifyEvent;
```

Description

Occurs after the connection to the remote side has been closed.

See Also

Connected

5.138 TScTCPServer

5.138.1 Description

Unit

ScTCPServer

Description

The **TScTCPServer** component implements functionality of the TCP/IP server.

TScTCPServer listens to the TCP/IP port specified in the Port property. When a TCP client tries to connect to this port, and **TScTCPServer** accepts the connection, an OnAcceptConnection event is triggered. The event returns a TScTCPConnection object which can be used for direct data exchange with the client or for TLS/SSL encrypted communucations using the TScSSLServerConnection class.

The information about the currently connected clients in stored in the Connections property.

See Also

Active

OnAcceptConnection

TScTCPConnection

TScSSLServerConnection

5.138.2 Properties

5.138.2.1 Active

```
property Active: boolean;
```

Description

Indicates whether the TCP server is running.

Set the **Active** property to True to run the TCP server and listen for incoming TCP/IP connections on Port.

Set the **Active** property to False to stop the TCP server.

See Also

BindAddress

Port

Start

5.138.2.2 BindAddress

```
property BindAddress: string;
```

Description

Specifies the local address the TCP server should listen to.

If there are several network cards installed on the computer, and **BindAddress** is not assigned, the server will listen to "0.0.0.0", which means that it is possible to connect to any of the available networking cards.

See Also

Start

5.138.2.3 ConnectionCount

```
property ConnectionCount: integer;
```

Description

Holds the number of currently connected clients.

The information about connections is stored in the Connections property.

See also

Connections

5.138.2.4 Connections

```
property Connections[Index: integer]: TScTCPConnection;
```

Description

Holds information about the currently connected clients. Information about a client is added to the list after establishing a connection, and is deleted after closing the connection.

The count of connections is stored in ConnectionCount.

See also

ConnectionCount

TScTCPConnection

5.138.2.5 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Specifies the Internet Protocol version.

The default value is `ivIPv4`.

See also

TIPVersion

5.138.2.6 ListenBacklog

```
property ListenBacklog: integer; default 5;
```

Description

Specifies the maximum number of queued connection requests that can be pending.

ListenBacklog is a socket-level property that describes the number of "pending accept" requests to be queued. If the listen backlog queue fills up, new socket requests will be rejected.

The default value is 5.

5.138.2.7 MaxAcceptedConnections

```
property MaxAcceptedConnections: integer; default 0;
```

Description

Specifies the maximum number of connections that the server can accept. When this number is exceeded, the thread for listening will be closed.

For example, if the server allows only one connection from the client (**MaxAcceptedConnections** is set to 1), it will stop listening on port once a connection has been established.

The default value is 0 — the server will accept an unlimited number of connections.

See Also

MaxOpenedConnections

5.138.2.8 MaxOpenedConnections

```
property MaxOpenedConnections: integer; default 0;
```

Description

Specifies the maximum number of concurrent connections to the server. Additional connections will be dropped until any of the opened connections is closed.

The default value is 0 — the server will allow an unlimited number of opened connections.

See Also

MaxAcceptedConnections

5.138.2.9 Port

```
property Port: integer;
```

Description

Specifies the TCP/IP port number that the sever will listen on.

See Also

Start

5.138.2.1(Timeout

```
property Timeout: integer; default 60;
```

Description

Determines time interval in seconds during which the server will be trying to obtain data from the

client during connection attempt. If the data has not been received during this period, the server closes the connection.

The default value is 60 seconds.

5.138.3 Methods

5.138.3.1 Start

```
procedure Start;
```

Description

Runs the TCP server.

Start binds a socket to BindAddress and starts listening for incoming TCP/IP connections on Port.

Start sets the Active property to True.

See also

BindAddress

Port

Stop

5.138.3.2 Stop

```
procedure Stop;
```

Description

Stops the TCP server and sets the Active property to False.

See also

Start

5.138.4 Events

5.138.4.1 AfterListenerEnd

```
property AfterListenerEnd: TNotifyEvent;
```

Description

Occurs after the server has been stopped and does not listen on the TCP/IP port anymore.

See Also

Active

5.138.4.2 BeforeBind

```
property BeforeBind: TNotifyEvent;
```

Description

Occurs immediately before binding a socket to BindAddress.

See Also

BindAddress

5.138.4.3 BeforeListenerRun

```
property BeforeListenerRun: TNotifyEvent;
```

Description

Occurs immediately before the server starts listening on the TCP/IP port.

See Also

Active

5.138.4.4 OnAcceptConnection

```
type
  TScAcceptConnectionEvent = procedure (Sender: TObject; Connection:
    TScTCPConnection; var Cancel: boolean) of object;

property OnAcceptConnection: TScAcceptConnectionEvent;
```

Description

Occurs on establishing a new TCP/IP connection, when there is a socket connection to the port listened that the TCP server.

To cancel a connection, set the `Cancel` property to True. This can be done for a specific IP address by using the Connection.GetRemoteIP property or for any other reason.

To further work with the connection, use a `Connection` object. To exchange data, you use the Connection.Read and Connection.Write methods.

To establish an SSL/TLS connection, initialize the TScSSLServerConnection class by passing a `Connection` object to the Init method.

Parameters:

- `Sender` - the object whose event handler is called;
- `Connection` - wraps the information about the socket that is trying to connect to the TCP server;
- `Cancel` - determines whether a TCP/IP connection to the specified socket will be established. Set `Cancel` to True if you want to cancel establishing a connection.

See Also

Active

5.138.4.5 OnCheckIfStopListen

```
type
  TScCheckIfStopListenEvent = procedure (Sender: TObject; var NeedStop:
    boolean) of object;

property OnCheckIfStopListen: TScCheckIfStopListenEvent;
```

Description

Occurs before the server starts listening on the TCP/IP port for a new connections, and then occurs every time a new connection has been accepted to decide whether to continue listening or stop the server.

To stop listening, set the `NeedStop` parameter to True.

Parameters:

- `Sender` - the object whose event handler is called;
- `NeedStop` - determines whether the server will continue listening on the TCP/IP port. Set `NeedStop` to True if you want to stop listening.

See Also

Active

OnAcceptConnection

5.138.4.6 OnException

```
type
  TScErrorEvent = procedure(Sender: TObject; E: Exception) of object;

property OnException: TScErrorEvent;
```

Description

Occurs when an error arises in the main thread of the TCP server.

The event handler is called in the thread where the exception has been generated.

Parameters:

- `Sender` - the object whose event handler is called;
- `E` - the object that describes the exception.

5.139 TScVersion

5.139.1 Description

Unit

ScSecureConnection

Description

The **TScVersion** represents the version number of an assembly, operating system, or the common language runtime.

Version numbers consist of two to four components: major, minor, build, and revision. The major and minor components are required; the build and revision components are optional, but the build component is required if the revision component is defined. All defined components must be integers greater than or equal to 0. The format of the version number is as follows (optional components are shown in square brackets ([and]):

`major.minor[.build[.revision]]`

See also

TScHttpWebRequest.ProtocolVersion

TScHttpWebResponse.ProtocolVersion

5.139.2 Properties

5.139.2.1 Build

```
property Build: integer; default -1;
```

Description

Gets the value of the build component of the version number for the current instance, or -1 if the build number is undefined.

For example, if the version number is 1.2.3.4, the build number is 3. If the version number is 1.2, the build number is undefined.

5.139.2.2 Major

```
property Major: integer;
```

Description

Gets the value of the major component of the version number for the current instance.

For example, if the version number is 1.2, the major version is 1.

5.139.2.3 Minor

```
property Minor: integer;
```

Description

Gets the value of the minor component of the version number for the current instance.

For example, if the version number is 1.2, the minor version is 2.

5.139.2.4 Revision

```
property Revision: integer; default -1;
```

Description

Gets the value of the revision component of the version number for the current instance, or -1 if the revision number is undefined.

For example, if the version number is 1.2.3.4, the revision number is 4. If the version number is 1.2, the revision number is undefined.

5.139.3 Methods

5.139.3.1 Create

```
constructor Create; overload;
constructor Create(Major, Minor: integer); overload;
constructor Create(Major, Minor, Build: integer); overload;
```

```
constructor Create(Major, Minor, Build, Revision: integer); overload;
```

Description

Create **TScVersion** instance and initialize it with the specified major, minor, build, and revision numbers.

The `Major` parameter is a value of the major component of the version number for this object. The Major property is set from the value of this parameter. The default value is 0.

The `Minor` parameter is a value of the minor component of the version number for this object. The Minor property is set from the value of this parameter. The default value is 0.

The `Build` parameter is a value of the build component of the version number for this object. The Build property is set from the value of this parameter. The default value is -1.

The `Revision` parameter is a value of the revision component of the version number for this object. The Revision property is set from the value of this parameter. The default value is -1.

5.139.3.2 IsEqual

```
function IsEqual(Obj: TScVersion): boolean;
```

Description

Use the **IsEqual** method to compare whether the current object and a specified `Obj` object represent the same value. If every component of the current object matches the corresponding component of the `Obj` parameter, the method returns True; otherwise, False.

5.139.3.3 Parse

```
procedure Parse(const Value: string);
```

Description

Converts the string `Value` that contains a version number to the current object.

The input parameter must have the following format:

`major.minor[.build[.revision]]`
where major, minor, build, and revision are the string representations of the version number's four components: major version number, minor version number, build number, and revision number. The components must appear in the specified order, and must be separated by periods.

See also

ToString

5.139.3.4 ToString

```
function ToString: string;
```

Description

Converts the value of the current object to its equivalent string representation.

See also

Parse

5.140 TScNetworkCredential

5.140.1 Description

Unit

ScSecureConnection

Description

The **TScNetworkCredential** class provides credentials for password-based authentication schemes such as basic, digest, NTLM, and Kerberos authentication.

See also

TScWebProxy.Credentials

TScHttpWebRequest.Credentials

TScHttpConnectionOptions.Credentials

5.140.2 Properties

5.140.2.1 Domain

```
property Domain: string;
```

Description

Gets or sets the domain or computer name that verifies the credentials.

The **Domain** property specifies the domain or realm to which the user name belongs. Typically, this is the host computer name where the application runs or the user domain for the currently logged in user.

See also

Password

UserName

5.140.2.2 Password

```
property Password: string;
```

Description

Gets or sets the password for the user name associated with the credentials.

See also

Domain

UserName

5.140.2.3 UserName

```
property UserName: string;
```

Description

Gets or sets the user name associated with the credentials.

See also

Domain

Password

5.141 TScCancellationToken

5.141.1 Description

Unit

ScCLRClasses

Description

The **TScCancellationToken** class is intended to send a notification that the current operation must be canceled.

An instance of this class is commonly passed to an asynchronous method. **TScCancellationToken** allows the calling or any other thread to cancel the operation performed by the method.

To cancel an operation you use the Cancel method. To check if an operation has been canceled you use the IsCancellationRequested method.

5.141.2 Methods

5.141.2.1 Cancel

```
procedure Cancel;
```

Description

Issues a cancellation request.

The **Cancel** method is commonly called in a thread different from the thread where a cancelable operation is running and listening for a cancellation request. When a cancellation request is received, the operation is stopped, and an exception is generated in the thread where the operation was running. The method does not generate any exceptions, and returns control to the program once the cancellation flag is set, without waiting for the operation cancellation.

See Also

IsCancellationRequested

ThrowIfCancellationRequested

5.141.2.2 Delay

```
procedure Delay(Timeout: cardinal);
```

Description

Checks whether a cancellation request has been issued. If no cancellation request is received during the `Timeout` period (in milliseconds), control is returned to the program. If a cancellation request is received during that period, an appropriate exception is generated.

Note: To issue a cancellation request you use the Cancel method.

See Also

Cancel

5.141.2.3 IsCancellationRequested

```
function IsCancellationRequested: boolean;
```

Description

Returns True if cancellation has been requested; otherwise, returns False.

Note: To issue a cancellation request you use the Cancel method.

See Also

Cancel

ThrowIfCancellationRequested

5.141.2.4 ThrowIfCancellationRequested

```
procedure ThrowIfCancellationRequested;
```

Description

Checks if a cancellation request has been issued. If there was a cancellation request, an appropriate exception is generated.

Note: To issue a cancellation request you use the Cancel method.

See Also

Cancel

IsCancellationRequested

5.142 TScWebProxy

5.142.1 Description

Unit

ScSecureConnection

Description

The **TScWebProxy** class contains the HTTP proxy settings that TScHttpWebRequest and TScWebSocketClient instances use to determine whether a Web proxy is used to send requests.

See also

TScHttpWebRequest.Proxy

TScWebSocketClient.Proxy

TScHttpConnectionOptions.Proxy

5.142.2 Properties

5.142.2.1 Address

```
property Address: string;
```

Description

Gets or sets the address of the proxy server.

5.142.2.2 Credentials

```
property Credentials: TScNetworkCredential;
```

Description

Gets or sets the credentials to submit to the proxy server for authentication.

5.142.2.3 Port

```
property Port: integer;
```

Description

Gets or sets the port number of the proxy server.

5.143 TScRequestCachePolicy

5.143.1 Description

Unit

ScHttp

Description

The **TScRequestCachePolicy** class defines an application's caching requirements for resources obtained by using TScHttpWebRequest objects.

You can specify the cache policy for an individual request by using the TScHttpWebRequest.CachePolicy property.

See also

TScHttpWebRequest.CachePolicy

5.143.2 Properties

5.143.2.1 Level

```
property Level: TScRequestCacheLevel;
```

Description

Gets or sets the TScRequestCacheLevel value that specifies the cache behavior for resources obtained using TScHttpWebRequest objects.

Applications typically use clDefault as their cache policy level.

5.143.3 Methods

5.143.3.1 Create

```
constructor Create(RequestCacheLevel: TScRequestCacheLevel = clDefault);
```

Description

Initializes the **TScRequestCachePolicy** instance using the specified cache policy.

The `RequestCacheLevel` parameter represents the cache behavior for resources obtained using TScHttpWebRequest objects. The Level property is set from the value of this parameter. If this parameter is not specified, the Level property will be set to the `clDefault` value.

5.144 TStrValueStringList

5.144.1 Description

Unit

ScTypes

Description

TStrValueStringList maintains a list of key-value pairs.

Use **TStrValueStringList** to store and maintain a list of key-value pairs. **TStrValueStringList** provides properties and methods to add, delete, locate, and access items.

See also

TScWebHeaderCollection

5.144.2 Properties

5.144.2.1 Count

```
property Count: Integer;
```

Description

Read **Count** to determine the number of entries in the Keys and Values arrays.

This property is read-only.

See also

Keys

Values

5.144.2.2 Keys

```
property Keys[Index: Integer]: string;
```

Description

Indicates the key part of items that are key-value pairs.

Use **Keys** to obtain a key part of an item from the list. The `Index` parameter indicates the index of the key, where 0 is the index of the first item, 1 is the index of the second item, and so on.

You can read the key at a specific index, or use **Keys** with the Count property to iterate through the list.

This property is read-only.

See also

Count

Values

5.144.2.3 Values

```
property Values[Index: Integer]: string;
```

Description

Indicates the value part of items that are key-value pairs.

Use **Values** to obtain a value part of an item from the list. The `Index` parameter indicates the index of the value, where 0 is the index of the first item, 1 is the index of the second item, and so on.

You can read or change the value at a specific index, or use **Values** with the Count property to iterate through the list.

See also

Count

Keys

5.144.3 Methods

5.144.3.1 Add

```
function Add(const Key, Value: string): Integer;
```

Description

Call **Add** to insert a key-value pair at the end of the list. **Add** returns the position of the item in the list, where the first item in the list has a value of 0.

Add increments Count and, if necessary, allocates memory.

See also

Count

Insert

5.144.3.2 Assign

```
procedure Assign(Source: TStrValueStringList);
```

Description

Call the **Assign** method to assign the elements of another list to this one.

5.144.3.3 Clear

```
procedure Clear;
```

Description

Deletes all items from the list. Call **Clear** to empty the Keys and Values arrays and set the Count to 0.

See also

Count

Keys
Values

5.144.3.4 Delete

```
procedure Delete(Index: Integer);
```

Description

Call **Delete** to remove the key-value pair at the position given by the `Index` parameter from the list. The index is zero-based, so the first item has an index value of 0, the second item has an index value of 1, and so on. Calling **Delete** moves up all items in the Keys and Values arrays that follow the deleted item, and reduces the Count.

See also

Clear
Count

5.144.3.5 IndexOf

```
function IndexOf(const Key: string): Integer;
```

Description

Returns the index of the first string `Key` in the list of key-value pairs with a specified value.

Call **IndexOf** to get the index for a specified key in the list, where the first object has index 0, the second object has index 1, and so on. If a key is not in the list, **IndexOf** returns -1. If a key appears more than once, **IndexOf** returns the index of the first appearance.

See also

Count
Keys

5.144.3.6 Insert

```
procedure Insert(Index: Integer; const Key, Value: string);
```

Description

Call **Insert** to add a key-value pair at a specified position in the list, shifting the item that previously occupied that position (and all subsequent items) up. **Insert** increments Count and, if necessary,

allocates memory.

The `Index` parameter is zero-based, so the first position in the list has an index of 0.

See also

Add

Count

5.144.3.7 TryGetValue

```
function TryGetValue(const Key: string; out Value: string): Boolean;
```

Description

Returns the value of the first string `Key` in the list of key-value pairs with a specified value.

Call **TryGetValue** to get the value for a specified key in the list. If a key is not in the list, **TryGetValue** returns False, else it returns True. If a key appears more than once, **TryGetValue** returns the value of the first appearance.

See also

Count

Keys

Values

5.145 TScWebHeaderCollection

5.145.1 Description

Unit

ScHttp

Description

TScWebHeaderCollection maintains a list of key-value pairs, that are protocol headers associated with an HTTP request or response.

The **TScWebHeaderCollection** class is generally accessed through TScHttpWebRequest.Headers or TScHttpWebResponse.Headers.

See also

TScHttpWebRequest.Headers

TScHttpWebResponse.Headers

TScWebRequestHeaderCollection

TScWebResponseHeaderCollection

TScHttpConnectionOptions.Headers

5.145.2 Methods

5.145.2.1 ToString

```
function ToString: string; virtual;
```

Description

Represents the list of key-value pairs as a string in the HTTP header format:

```
'Header1: Value1#13#10Header2: Value2#13#10Header3: Value3'
```

5.146 TScWebRequestHeaderCollection

5.146.1 Description

Unit

ScHttp

Description

TScWebRequestHeaderCollection maintains a list of key-value pairs, that are protocol headers associated with an HTTP request.

The **TScWebRequestHeaderCollection** class is accessed through TScHttpWebRequest.Headers.

See also

TScHttpWebRequest.Headers

5.146.2 Methods

5.146.2.1 Create

```
constructor Create(Owner: TScHttpWebRequest);
```

Description

Create **TScWebRequestHeaderCollection** instance.

The `Owner` parameter is an object that represents the HTTP request that contains protocol headers.

5.147 TScWebResponseHeaderCollection

5.147.1 Description

Unit

ScHttp

Description

TScWebResponseHeaderCollection maintains a list of key-value pairs, that are protocol headers associated with an HTTP response.

The **TScWebResponseHeaderCollection** class is accessed through TScHttpWebResponse.Headers.

See also

TScHttpWebResponse.Headers

5.147.2 Methods

5.147.2.1 Create

```
constructor Create(Owner: TScHttpWebResponse);
```

Description

Create **TScWebResponseHeaderCollection** instance.

The `Owner` parameter is an object that represents the HTTP response that contains protocol headers.

5.148 TScHttpWebRequest

5.148.1 Description

Unit

ScHttp

Description

TScHttpWebRequest is a component for the request/response model for accessing data by the HTTP protocol. Requests are sent from an application to a particular URI, such as a Web page on a server.

The **TScHttpWebRequest** class provides support for the properties and methods that enable the user to interact directly with servers using HTTP.

The GetResponse method makes a request to the resource specified in the RequestUri property and

returns an TScHttpWebResponse that contains the response object. The response data can be received by using the TScHttpWebResponse.ReadBuffer method.

When you want to send data to the resource, use the WriteBuffer method or the RequestStream property in the chunked mode (see SendChunked).

The **TScHttpWebRequest** class throws a HttpException when errors occur while accessing a resource. The HttpException.StatusCode property contains a TScHttpStatusCode value that indicates the source of the error.

TScHttpWebRequest exposes common HTTP header values sent to the Internet resource as properties. You can set other headers in the Headers property as name/value pairs. Note that servers and caches may change or add headers during the request.

See Also

GetResponse

Headers

RequestUri

SendChunked

TScHttpWebResponse

WriteBuffer

5.148.2 Properties

5.148.2.1 Accept

```
property Accept: string;
```

Description

Gets or sets the value of the Accept HTTP header.

See also

Headers

5.148.2.2 Address

```
property Address: string;
```

Description

Gets the Uniform Resource Identifier (URI) of the Internet resource that actually responds to the request. The default is the URI used by the Create method to initialize the request.

The **Address** property is set to the URI after any redirections that happen during the request are complete. The URI of the original request is kept in the RequestUri property.

This property is read-only.

See also

RequestUri

5.148.2.3 CachePolicy

```
property CachePolicy: TScRequestCachePolicy;
```

Description

Gets or sets the cache policy for this request.

The current cache policy and the presence of the requested resource in the cache determine whether a response can be retrieved from the cache. Using cached responses usually improves application performance, but there is a risk that the response in the cache does not match the response on the server.

A copy of a resource is only added to the cache if the response stream for the resource is retrieved and read to the end of the stream. So another request for the same resource could use a cached copy, depending on the cache policy level for this request.

See also

Headers

5.148.2.4 Connection

```
property Connection: string;
```

Description

Gets or sets the value of the Connection HTTP header.

The request sends the **Connection** property to the Internet resource as the Connection HTTP header. If the value of the KeepAlive property is True, the value "Keep-alive" is appended to the end of the Connection header.

See also

Headers

KeepAlive

5.148.2.5 ConnectionGroupName

```
property ConnectionGroupName: string;
```

Description

Gets or sets the name of the connection group for the request.

The **ConnectionGroupName** property enables you to associate a request with a connection group. This is useful when your application makes requests to one server for different users, such as a Web site that retrieves customer information from a database server. In this case, you can add this connection to the pool by setting the **ConnectionGroupName** property. And then after receiving a response from the server, the connection will not break. The next time you create an HTTP request to the same server, a new TCP connection will not be created, but an existing connection from the pool will be taken.

See Also

GetResponse

5.148.2.6 ContentLength

```
property ContentLength: Int64; default -1;
```

Description

Gets or sets the Content-length HTTP header. The **ContentLength** value is the number of bytes of data to send to the Internet resource. The default is -1, which indicates the property has not been set and that there is no request data to send.

The **ContentLength** property contains the value to send as the Content-length HTTP header with the request. Any value other than -1 in the **ContentLength** property indicates that the request uploads data and that only methods that upload data are allowed to be set in the Method property.

See also

Headers

Method

5.148.2.7 ContentType

```
property ContentType: string;
```

Description

Gets or sets the value of the Content-type HTTP header.

The **ContentType** property contains the media type of the request. Values assigned to the **ContentType** property replace any existing contents when the request sends the Content-type HTTP header.

See also

Headers

5.148.2.8 Cookies

```
property Cookies: TStringList;
```

Description

Gets or sets the cookies associated with the request.

You must assign a cookies string to the property to have cookies returned in the Cookies property of the TScHttpWebResponse returned by the GetResponse method.

See also

TScHttpWebResponse.Cookies

5.148.2.9 Credentials

```
property Credentials: TScNetworkCredential;
```

Description

Gets or sets authentication information for the request.

The **Credentials** property contains authentication information to identify the maker of the request. The user, password, and domain information contained in the TScNetworkCredential object is used to authenticate the request.

5.148.2.10 Date

```
property Date: TDateTime;
```

Description

Get or set the Date HTTP header value to use in an HTTP request.

See also

Headers

5.148.2.1.1 Expect

```
property Expect: string;
```

Description

Gets or sets the value of the Expect HTTP header.

See also

Headers

5.148.2.1.2 From

```
property From: string;
```

Description

Gets or sets the value of the From HTTP header.

See also

Headers

5.148.2.1.3 Headers

```
property Headers: TScWebHeaderCollection;
```

Description

Specifies a collection of the name/value pairs that make up the HTTP headers.

The **Headers** collection contains the protocol headers associated with the request. The following table lists the HTTP headers that are not stored in the **Headers** collection but are set by properties or methods.

Header	Set by
Accept	Set by the Accept property.
Connection	Set by the Connection property and KeepAlive property.
Content-Length	Set by the ContentLength property.

Header	Set by
Content-Type	Set by the ContentType property.
Date	Set by the Date property.
Expect	Set by the Expect property.
From	Set by the From property.
Host	Set by the Host property.
If-Modified-Since	Set by the IfModifiedSince property.
Range	Set by the Range property.
Referer	Set by the Referer property.
Transfer-Encoding	Set by the TransferEncoding property.
Upgrade	Set by the Upgrade property.
User-Agent	Set by the UserAgent property.

The TScWebHeaderCollection.Add method throws an Exception if you try to set one of these protected headers.

You should not assume that the header values will remain unchanged, because Web servers and caches may change or add headers to a Web request.

5.148.2.14 Host

```
property Host: string;
```

Description

Get or set the Host header value to use in an HTTP request independent from the request URI.

The **Host** property can be used to set the Host header value to use in an HTTP request independent from the request URI. The **Host** property can consist of a hostname and an optional port number. A Host header without port information implies the default port for the service requested (port 80 for an HTTP URL, for example).

The format for specifying a host and port must follow the rules in section 14.23 of RFC2616 published by the IETF. An example complying with these requirements that specifies a port of 8080 would be the following value for the **Host** property: 'www.host.com:8080'.

If the **Host** property is not set, then the Host header value to use in an HTTP request is based on the request URI.

See also

Headers

5.148.2.15 IfModifiedSince

```
property IfModifiedSince: TDateTime;
```

Description

Gets or sets the value of the If-Modified-Since HTTP header.

See also

Headers

5.148.2.16 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is `ivIPv4`.

See also

TIPVersion

5.148.2.17 IsSecure

```
property IsSecure: boolean;
```

Description

Determines whether the connection to a Web server is protected.

When **IsSecure** is set to False, data is transferred as plaintext.

When **IsSecure** is set to True, the TLS/SSL protocol is used, and data is transferred in an encrypted form.

This property is read-only.

See also

SSLOptions

5.148.2.15 KeepAlive

```
property KeepAlive: boolean; default True;
```

Description

Gets or sets a value that indicates whether to make a persistent connection to the Internet resource.

Set this property to True to send a Connection HTTP header with the value Keep-alive. An application uses **KeepAlive** to indicate a preference for persistent connections. When the **KeepAlive** property is True, the application makes persistent connections to the servers that support them.

See also

Headers

5.148.2.16 MaximumAutomaticRedirections

```
property MaximumAutomaticRedirections: integer; default 50;
```

Description

Gets or sets the maximum number of redirects that the request follows. The default value is 50.

5.148.2.20 Method

```
property Method: TScRequestMethod; default rmGET;
```

Description

Gets or sets the method for the request. The request method to use to contact the Internet resource. The default value is GET.

The **Method** property can be set to any of the HTTP 1.1 protocol verbs: GET, HEAD, OPTIONS, POST, PUT, DELETE, TRACE, or PATCH.

If the ContentLength property is set to any value other than -1, the **Method** property must be set to a protocol property that uploads data.

See also

ContentLength

GetResponse

5.148.2.22 ProtocolVersion

```
property ProtocolVersion: TScVersion;
```

Description

Gets or sets the version of HTTP to use for the request. The default is 1.1 version.

The TScHttpWebRequest class supports only versions 1.0 and 1.1 of HTTP.

5.148.2.23 Proxy

```
property Proxy: TScWebProxy;
```

Description

Gets or sets proxy information for the connection.

If it is necessary to connect to server in another network, sometimes the client can reach it only through proxy. In this case you have to setup the **Proxy** property that identifies the TScWebProxy object to use to process requests to Internet resources.

5.148.2.24 Range

```
property Range: string;
```

Description

Gets or sets the value of the Range HTTP header.

See also

Headers

5.148.2.25 ReadWriteTimeout

```
property ReadWriteTimeout: integer; default 15;
```

Description

Gets or sets a time-out in seconds when writing to or reading from a stream.

ReadWriteTimeout is the number of seconds before the writing or reading times out.

The default value is 15 seconds.

5.148.2.25 Referer

```
property Referer: string;
```

Description

Gets or sets the value of the Referer HTTP header.

If the MaximumAutomaticRedirections property is greater than zero, the **Referer** property is set automatically when the request is redirected to another site.

See also

Headers

MaximumAutomaticRedirections

5.148.2.26 RequestStream

```
property RequestStream: TStream;
```

Description

The **RequestSteam** property gets or sets a stream that is used to write the request data. This property can be only set if SendChunked is set to True and Method is not rmGET or rmHEAD.

If the value of **RequestStream** is not empty and SendChunked is set to False, HttpException will be raised.

When request data is sent to the HTTP server (that is performed by the GetResponse method), TScHttpWebRequest reads data from the **RequestStream** object in blocks of size specified in the SendBlockSize property and sends this chunk to the server. The number of bytes of request data to be sent is specified in the ContentLength property.

When **RequestStream** is not set and SendChunked is set to True, TScHttpWebRequest checks if the OnGetNextChunkData event handler is set - if OnGetNextChunkData is not set, HttpException is raised.

See also

OnGetNextChunkData

SendBlockSize

SendChunked

5.148.2.27 RequestUri

```
property RequestUri: string;
```

Description

Gets or sets the original Uniform Resource Identifier (URI) of the request.

Following a redirection header does not change the **RequestUri** property. To get the actual URI that responded to the request, examine the Address property.

See also

Address

Create

GetResponse

5.148.2.28 SendBlockSize

```
property SendBlockSize: integer; default 32768;
```

Description

The **SendBlockSize** property determines the maximum size of the data block that will be sent to the HTTP server as a single data packet. Before sending each data block, the BeforeSendData event handler will be called, where the sending can be canceled.

The default value is 32768.

See also

RequestStream

SendChunked

WriteBuffer

5.148.2.29 SendChunked

```
property SendChunked: boolean; default False;
```

Description

The **SendChunked** property gets or sets the value that indicates whether to send data in segments to the Internet resource.

When **SendChunked** is True, the request sends data to the Internet resource in segments. The Internet resource must support receiving chunked data.

If you change the **SendChunked** property after the request has been started by calling the GetResponse method, the HttpException is raised.

If **SendChunked** is set to True, TScHttpWebRequest checks whether the RequestStream property or the OnGetNextChunkData event handler is set, and if both values are nil, the HttpException is raised.

The default value is False.

See also

OnGetNextChunkData

RequestStream

SendBlockSize

5.148.2.3 SSLOptions

```
property SSLOptions: TScSSLClientOptions;
```

Description

SSLOptions determines behaviour of a TLS/SSL connection.

See also

IsSecure

GetResponse

5.148.2.3 StatusCode

```
property StatusCode: TScHttpStatusCode;
```

Description

The **StatusCode** property holds a value that indicates the status of the HTTP response. The expected values for status are defined in the TScHttpStatusCode enumeration.

This property is read-only.

See also

StatusDescription

TScHttpStatusCode

5.148.2.3 StatusDescription

```
property StatusDescription: string;
```

Description

The **StatusDescription** property holds a string that describes the status of the HTTP response.

This property is read-only.

See also

StatusCode

TScHttpStatusCode

5.148.2.33 TransferEncoding

```
property TransferEncoding: string;
```

Description

Gets or sets the value of the Transfer-encoding HTTP header.

See also

Headers

5.148.2.34 TrustServerCertificate

```
property TrustServerCertificate: boolean;
```

Description

The **TrustServerCertificate** property specifies if the SSL certificate of Web server will be validated by client.

When **TrustServerCertificate** is set to True, the TScHttpWebRequest will not validate the SSL certificate of Web server.

5.148.2.35 Upgrade

```
property Upgrade: string;
```

Description

Gets or sets the value of the Upgrade HTTP header.

See also

Headers

5.148.2.3 UserAgent

```
property UserAgent: string;
```

Description

Gets or sets the value of the User-agent HTTP header.

See also

Headers

5.148.3 Methods

5.148.3.1 Abort

```
procedure Abort;
```

Description

The **Abort** method cancels a request to a resource. After a request is canceled, calling the GetResponse method causes an Exception with the StatusCode property set to RequestCanceled.

Use this method only when a connection hangs. Use the Disconnect method to close a connection normally.

See Also

Disconnect

5.148.3.2 Create

```
constructor Create(const URI: string);
```

Description

Create **TScHttpWebRequest** instance and initialize it with the specified URI scheme.

The `URI` parameter is the URI that identifies the Internet resource. The RequestUri property is set from the value of this parameter.

See Also

GetResponse

RequestUri

5.148.3.3 Disconnect

```
procedure Disconnect;
```

Description

Call **Disconnect** to close a connection to an HTTP resource.

See Also

Abort

5.148.3.4 GetResponse

```
function GetResponse: TScHttpWebResponse;
```

Description

The **GetResponse** method creates and returns a TScHttpWebResponse object that contains the response from the Internet resource.

You must free the returned TScHttpWebResponse instance to close the response and release the connection.

5.148.3.5 WriteBuffer

```
function WriteBuffer(const Buffer: TValueArr; Offset, Count: integer):
integer; overload;
procedure WriteBuffer(const Buffer: TBytes); overload;
```

Description

Call **WriteBuffer** to send `Count` bytes from `Buffer` to an HTTP resource. If request data exists (**WriteBuffer** or WriteData has been already called), the method appends data to the end of an existing buffer.

If an application needs to set the value of the ContentLength property, the value must be set before sending data.

Use this method only in non-chunked mode. If SendChunked is set to True, the method call will raise the HttpException.

See also

ContentLength

WriteData

5.148.3.6 WriteData

```
procedure WriteData(Stream: TStream);
```

Description

Call **WriteData** to send data from `Stream` to an HTTP resource. If request data exists (**WriteData** or WriteBuffer has already been called), this method overrides existing data, unlike the WriteBuffer method.

If an application needs to set the value of the ContentLength property, the value must be set before sending data.

Use this method only in non-chunked mode. If SendChunked is set to True, the method call will raise the HttpException.

See also

ContentLength

RequestStream

SendChunked

WriteBuffer

5.148.4 Events

5.148.4.1 AfterSendRequest

```
property AfterSendRequest: TNotifyEvent;
```

Description

The **AfterSendRequest** event occurs after all request data has been sent to the web server, but before retrieving a response from the server.

AfterSendRequest is similar to the OnConnected event and occurs right before it.

See Also

BeforeSendRequest

OnConnected

5.148.4.2 BeforeSendData

```
type
  TScBeforeSendDataEvent = procedure (Sender: TObject; Offset, Count:
Int64; var Cancel: boolean) of object;

property BeforeSendData: TScBeforeSendDataEvent;
```

Description

The **BeforeSendData** event occurs before sending each block of request data to the web server. The data blocks are divided into pieces specified in the SendBlockSize property.

BeforeSendData occurs for content data and not for the headers.

`Offset` is the number of bytes of data that has been already sent to the Internet resource.

`Count` is the common number of bytes of data to be sent to the Internet resource. Count is the same as the ContentLength property.

You can set the `Cancel` parameter to True to cancel request sending, in which case the OperationCanceledException will be raised.

See Also

AfterSendRequest

BeforeSendRequest

SendBlockSize

5.148.4.3 BeforeSendRequest

```
property BeforeSendRequest: TNotifyEvent;
```

Description

The **BeforeSendRequest** event occurs before establishing a connection to the web server and sending request data.

See Also

AfterSendRequest

AfterSendRequest

5.148.4.4 OnAuthenticationNeeded

```
property OnAuthenticationNeeded: TNotifyEvent;
```

Description

The **OnAuthenticationNeeded** event occurs when a Web server returns the HTTP status 401, that indicates that the requested resource requires authentication.

See Also

OnConnected

TScHttpStatusCode

5.148.4.5 OnConnected

```
property OnConnected: TNotifyEvent;
```

Description

The **OnConnected** event occurs after a connection to the web server is established, but before retrieving a response from the server.

OnConnected is similar to the AfterSendRequest event and occurs right after it.

See Also

AfterSendRequest

OnAuthenticationNeeded

5.148.4.6 OnGetNextChunkData

```
type
  TScOnGetNextChunkDataEvent = procedure (Sender: TObject; out Buffer:
TValueArr; out Count: Integer) of object;

property OnGetNextChunkData: TScOnGetNextChunkDataEvent;
```

Description

The **OnGetNextChunkData** event occurs when writing request data to HTTP server if SendChunked is set to True and RequestStream is set to nil.

This property can only be set if SendChunked is set to True and Method is not rmGET or rmHEAD, otherwise the HttpException will be raised.

When writing request data to the HTTP server (that is performed by the GetResponse method) TScHttpWebRequest gets data from the **OnGetNextChunkData** event handler in blocks of size specified in the SendBlockSize property and sends this chunk to the server. Set the `Count` parameter to 0 to indicate that all data has been transferred. The size of request data in bytes can be also specified in the ContentLength property.

When SendChunked is set to True, TScHttpWebRequest checks whether the RequestStream property or the **OnGetNextChunkData** event handler is set, and if both are nil, the HttpException will be raised.

See Also

RequestStream

SendBlockSize

SendChunked

5.149 TScHttpWebResponse

5.149.1 Description

Unit

ScHttp

Description

TScHttpWebResponse is a component that used to build HTTP stand-alone client applications that send HTTP requests and receive HTTP responses.

You should never directly create an instance of the **TScHttpWebResponse** class. Instead, use the instance returned by a call to TScHttpWebRequest.GetResponse. You must free this **TScHttpWebResponse** instance to close the response and release the connection.

Common header information returned from the Internet resource is exposed as properties of the class. Other headers can be read from the Headers property as name/value pairs.

See Also

TScHttpWebRequest

5.149.2 Properties

5.149.2.1 ContentEncoding

```
property ContentEncoding: string;
```

Description

Gets the method that is used to encode the body of the response.

The **ContentEncoding** property contains the value of the Content-Encoding header returned with the response.

This property is read-only.

See also

Headers

5.149.2.2 ContentLength

```
property ContentLength: Int64;
```

Description

Gets the length of the content returned by the request. Content length does not include header information.

The **ContentLength** property contains the value of the Content-Length header returned with the response. If the Content-Length header is not set in the response, **ContentLength** is set to the value -1.

This property is read-only.

See also

Headers

5.149.2.3 ContentType

```
property ContentType: string;
```

Description

Gets the content type of the response. The **ContentType** property contains the value of the Content-Type header returned with the response.

This property is read-only.

See also

Headers

5.149.2.4 Cookies

```
property Cookies: TStringList;
```

Description

Gets the cookies that are associated with this response.

Any cookie information sent by the server will be available in the Headers property.

This property is read-only.

See also

Headers

5.149.2.5 Headers

```
property Headers: TScWebHeaderCollection;
```

Description

Gets the headers that are associated with this response from the server.

The **Headers** property is a collection of name/value pairs that contain the HTTP header values returned with the response. Common header information returned from the Internet resource is exposed as properties of the TScHttpWebResponse class.

This property is read-only.

See also

TScHttpWebRequest.Headers

5.149.2.6 IsSecure

```
property IsSecure: boolean;
```

Description

Determines whether the connection to a Web server is protected.

When **IsSecure** is set to False, data is transferred as plaintext.

When **IsSecure** is set to True, the TLS/SSL protocol is used, and data is transferred in an encrypted form.

This property is read-only.

5.149.2.7 LastModified

```
property LastModified: TDateTime;
```

Description

Gets the last date and time that the contents of the response were modified.

The **LastModified** property contains the value of the Last-Modified header received with the response. The date and time are assumed to be local time.

This property is read-only.

See also

Headers

5.149.2.8 Method

```
property Method: TScRequestMethod;
```

Description

Gets the method that is used to return the response.

Common HTTP 1.1 protocol methods are GET, HEAD, POST, PUT, and DELETE.

This property is read-only.

See also

TScRequestMethod

TScHttpWebRequest.Method

5.149.2.9 ProtocolVersion

```
property ProtocolVersion: TScVersion;
```

Description

The **ProtocolVersion** property contains the HTTP protocol version number of the response sent by the Internet resource.

This property is read-only.

5.149.2.1(ResponseUri

```
property ResponseUri: string;
```

Description

Gets the URI of the Internet resource that responded to the request.

The **ResponseUri** property contains the URI of the Internet resource that actually responded to the request. This URI might not be the same as the originally requested URI, if the original server redirected the request.

This property is read-only.

See also

TScHttpWebRequest.Address

TScHttpWebRequest.RequestUri

5.149.2.1.1 Server

```
property Server: string;
```

Description

Gets the name of the server that sent the response.

The **Server** property contains the value of the Server header returned with the response.

This property is read-only.

See also

Headers

5.149.2.1.2 StatusCode

```
property StatusCode: TScHttpStatusCode;
```

Description

Gets the status of the response.

The **StatusCode** property holds a value that indicates the status of the HTTP response. The expected values for status are defined in the TScHttpStatusCode enumeration.

This property is read-only.

See also

StatusDescription

5.149.2.1.3 StatusDescription

```
property StatusDescription: string;
```

Description

The **StatusDescription** property holds a string that describes the status of the HTTP response.

This property is read-only.

See also

StatusCode

TScHttpStatusCode

5.149.3 Methods

5.149.3.1 Abort

```
procedure Abort;
```

Description

The **Abort** method cancels getting of the response from a Web resource.

5.149.3.2 GetResponseHeader

```
function GetResponseHeader(const HeaderName: string): string;
```

Description

Use **GetResponseHeader** to retrieve the contents of particular headers. You must specify which header you want to return.

5.149.3.3 ReadAsBytes

```
function ReadAsBytes: TBytes;
```

Description

Call **ReadAsBytes** to read the body of the response from the Web server. The function returns a byte array containing the body of the response.

See also

ReadAsString

ReadBuffer

ReadToStream

5.149.3.4 ReadAsString

```
function ReadAsString: string;
```

Description

Call **ReadAsString** to read the body of the response from the Web server. The function returns a string containing the body of the response.

See also

ReadAsBytes

ReadBuffer

ReadToStream

5.149.3.5 ReadBuffer

```
function ReadBuffer(const Buffer: TValueArr; Offset, Count: integer):
integer;
```

Description

Call **ReadBuffer** to read `Count` bytes from the body of the response from the Web server into `Buffer`. **ReadBuffer** returns bytes count that was actually read.

Using this method, the resulting content can be read in a non-bloking mode. This is especially important when the body of a message is returned by the server as a series of chunk. In this case, the method will return a result as it receives pieces of data without causing the calling stream to hang.

See also

ReadAsBytes

ReadAsString

ReadToStream

5.149.3.6 ReadToStream

```
function ReadToStream(Stream: TStream): integer;
```

Description

Call **ReadToStream** to read data from the body of the response from the web server into `Stream`. **ReadToStream** returns the count of read bytes.

See also

ReadAsBytes

ReadAsString

ReadBuffer

5.149.3.7 WaitForData

```
function WaitForData(Timeout: integer): boolean;
```

Description

Waits for the received data from the Web server during amount of time specified in the `Timeout` parameter in milliseconds, and then returns control. The calling thread will wait indefinitely when the `Timeout` parameter is set to -1.

If data from server was received and is in the buffer for reading, the method returns True. False otherwise.

See also

ReadBuffer

5.150 TScWebSocketClientOptions

5.150.1 Description

Unit

ScWebSocketClient

Description

The **TScWebSocketClientOptions** class determines behaviour of a WebSocket client.

See also

TScWebSocketClient.Options

5.150.2 Properties

5.150.2.1 Cookies

```
property Cookies: string;
```

Description

Determines the cookies associated with the request to a Web server. This value is included in a handshake when establishing a connection.

5.150.2.2 Credentials

```
property Credentials: TScNetworkCredential;
```

Description

Gets or sets authentication information for the WebSocket client.

The **Credentials** property contains authentication information to identify the maker of the request. The user, password, and domain information contained in the TScNetworkCredential object is used to authenticate the request.

5.150.2.3 Extensions

```
property Extensions: string;
```

Description

Determines the extensions supported by the client. This value is included in a handshake when establishing a connection.

5.150.2.4 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is `ivIPv4`.

See also

TIPVersion

5.150.2.5 MaxFragmentSize

```
property MaxFragmentSize: cardinal; default 0;
```

Description

MaxFragmentSize is the maximum size of fragment sent by the client to the server. If the data size, which the client wants to send to the server (TScWebSocketClient.Send), exceeds the specified value, the client will break data into pieces by himself, which won't exceed the specified size and will send them to the server in different frames.

0 value means that the client won't break data into pieces.

The default value is 0.

5.150.2.6 Origin

```
property Origin: string;
```

Description

Determines the origin generating the WebSocket connection request. This value is included in a handshake when establishing a connection.

The **Origin** field is used to protect against unauthorized cross-origin use of a WebSocket server. The server is informed of the script origin generating the WebSocket connection request. If the server does not wish to accept connections from this origin, it can choose to reject the connection.

5.150.2.7 ReadWriteTimeout

```
property ReadWriteTimeout: integer; default 15;
```

Description

Gets or sets a time-out in seconds when writing to or reading from a stream.

ReadWriteTimeout is the number of seconds before the writing or reading times out.

The default value is 15 seconds.

5.150.2.8 RequestHeaders

```
property RequestHeaders: TScWebHeaderCollection;
```

Description

Specifies a collection of the name/value pairs that make up the HTTP headers.

The **RequestHeaders** collection contains the protocol headers associated with the request. These headers are included in a handshake when establishing a connection.

5.150.2.9 SubProtocols

```
property SubProtocols: string;
```

Description

Determines the list of specific sub-protocols supported by a WebSocket client. This value is included in a handshake when establishing a connection.

5.150.2.1 UserAgent

```
property UserAgent: string;
```

Description

Determines the User-agent HTTP header associated with the request to a Web server. This value is included in a handshake when establishing a connection.

5.151 TScHeartBeatOptions

5.151.1 Description

Unit

ScWebSocketClient

Description

The **TScHeartBeatOptions** class determines the behaviour of the HeartBeat mode.

If this mode is Enabled, a WebSocket client tries to keep a websocket connection alive automatically sending a ping every Interval seconds.

If the timeout defined in theTimeout property expires, the client does not receive a response from the server, the connection will be closed and the corresponding error will be raised.

See also

TScWebSocketClient.HeartBeatOptions

5.151.2 Properties

5.151.2.1 Enabled

```
property Enabled: boolean; default False;
```

Description

The **Enabled** property indicates whether the HeratBeat mode is enabled.

If True, a WebSocket client tries to keep a websocket connection alive automatically sending a ping every Interval seconds.

See also

Interval

5.151.2.2 Interval

```
property Interval: integer; default 15;
```

Description

The **Interval** property holds a number of seconds between each ping.

The default value is 15 seconds.

See also

Timeout

5.151.2.3 Timeout

```
property Timeout: integer; default 120;
```

Description

The **Timeout** property holds a maximum number of seconds between a ping and pong.

If the time defined in the **Timeout** property expires, the client does not receive a response from the server, the connection will be closed and the corresponding error will be raised.

The default value is 120 seconds.

See also

Interval

5.152 TScWatchDogOptions

5.152.1 Description

Unit

ScWebSocketClient

Description

The **TScWatchDogOptions** class determines the behaviour of the WatchDog mode.

If this mode is Enabled, a WebSocket client tries to reconnect to the server automatically every Interval seconds, when an unexpected disconnection is detected.

The maximum number of reconnects is set in the Attempts property.

See also

TScWebSocketClient.WatchDogOptions

5.152.2 Properties

5.152.2.1 Attempts

```
property Attempts: integer; default 0;
```

Description

The **Attempts** property holds a maximum number of reconnects.

If the property value is set to -1, then a number of attempts is unlimited.

The default value is 0.

See also

Interval

5.152.2.2 Enabled

```
property Enabled: boolean; default False;
```

Description

The **Enabled** property indicates whether the WatchDog mode is enabled.

If True, a WebSocket client will try reconnecting to server automatically, when an unexpected disconnection is detected.

See also

Interval

5.152.2.3 Interval

```
property Interval: integer; default -1;
```

Description

The **Interval** property holds number of seconds before reconnects.

If the property value is set to -1, the interval will be generated randomly ranging from 1 to 5 seconds. This is important to prevent server overload if there is a short server or network failure when a lot of clients try to join the server simultaneously.

The default value is -1.

See also

Attempts

5.153 TScWebSocketClient

5.153.1 Description

Unit

ScWebSocketClient

Description

TScWebSocketClient is a component that implements functionality of the WebSocket client and allows connecting to a WebSocket server without using third-party libraries or components.

TScWebSocketClient supports both non-secure communication mode and secure mode using TLS/SSL protocol. To set up a TLS connection, use the SSLOptions property.

The Connect method makes a request using HTTP protocol to the resource specified in the RequestUri property and if the server supports a WebSocket protocol, establishes a connection.

To send data to the server, use the Send method.

You can retrieve data both in a synchronous mode, using the Receive method for reading, and in an asynchronous mode, handling the OnMessage event.

The **TScWebSocketClient** class throws a WebSocketException exception when an error occurs during the session.

Also, the component supports the HeartBeat mode to keep a websocket connection alive and the WatchDog mode to reconnect to the server automatically when there is an unexpected disconnection.

See Also

Connect

RequestUri

SSLOptions

5.153.2 Properties

5.153.2.1 CloseStatus

```
property CloseStatus: TScWebSocketCloseStatus;
```

Description

The **CloseStatus** property holds a value that indicates the reason for closing the connection. The expected values for close status are defined in the TScWebSocketCloseStatus enumeration.

This property is read-only.

See also

CloseStatusDescription

5.153.2.2 CloseStatusDescription

```
property CloseStatusDescription: string;
```

Description

The **CloseStatusDescription** property holds a string that describes the reason for closing a connection.

This property is read-only.

See also

CloseStatus

5.153.2.3 EventsCallMode

```
property EventsCallMode: TScEventCallMode; default ecAsynchronous;
```

Description

The **EventsCallMode** property determines how the event handlers will be called. The thing is that data coming from the server is processed in a separate thread of the WebSocket connection. And the call of the event handlers can occur in a different way for synchronization with the main thread of the application.

The default value is the `ecAsynchronous` mode when the events are added to a queue and then asynchronously synchronized from this queue with the main thread. This allows not slowing down the thread in which events occur and at the same calling the event handlers in the main thread.

When setting the property to the `ecSynchronous` value, the event call will be immediately synchronized with the main thread.

When setting the property to the `ecDirectly` value, there is no synchronization with the main thread.

Default value is the `ecAsynchronous` mode.

See also

OnControlMessage

OnMessage

5.153.2.4 ExtensionsInUse

```
property ExtensionsInUse: string;
```

Description

Gets the extensions that are supported by the server, which a connection is established to. The **ExtensionsInUse** property is set when establishing a connection to a WebSocket server.

This property is read-only.

See also

Connect

5.153.2.5 HeartBeatOptions

```
property HeartBeatOptions: TScHeartBeatOptions;
```

Description

HeartBeatOptions determines the behaviour of the HeartBeat mode. This mode is used for attempting to keep a websocket connection alive automatically sending a ping every x seconds.

If after the specified time, the client does not receive a response from the server, the connection will be closed and a corresponding error will be raised.

See also

Ping

5.153.2.6 IsSecure

```
property IsSecure: boolean;
```

Description

Determines whether the connection to a Web server is protected.

When **IsSecure** is set to False, data is transferred as plaintext.

When **IsSecure** is set to True, the TLS/SSL protocol is used, and data is transferred in an encrypted form.

This property is read-only.

See also

SSLOptions

5.153.2.7 Options

```
property Options: TScWebSocketClientOptions;
```

Description

Options determines the behaviour of a WebSocket client.

5.153.2.8 Proxy

```
property Proxy: TScWebProxy;
```

Description

Gets or sets proxy information for the connection.

If it is necessary to connect to a server in another network, sometimes the client can reach it only through proxy. In this case you have to setup the **Proxy** property that identifies the TScWebProxy object to use to process requests to Internet resources.

5.153.2.9 RequestUri

```
property RequestUri: string;
```

Description

Gets or sets the original Uniform Resource Identifier (URI) of the request. Following a redirection header does not change the **RequestUri** property.

See also

Create

Connect

5.153.2.10 ResponseHeaders

```
property ResponseHeaders: TScWebHeaderCollection;
```

Description

Gets the headers that are associated with this response from the server.

The **ResponseHeaders** property is a collection of name/value pairs that contain the HTTP header values returned with the response. Common header information returned from the Internet resource is exposed as properties of the TScHttpWebResponse class.

This property is read-only.

5.153.2.11 SecWebSocketKey

```
property SecWebSocketKey: string;
```

Description

Gets the Sec-WebSocket-Key that is used for identifying the client in the current connection. The **SecWebSocketKey** key is automatically generated when calling the Connect method.

The **SecWebSocketKey** field is used in the WebSocket opening handshake. It is sent from the client to the server to provide part of the information used by the server to prove that it received a valid WebSocket opening handshake.

This property is read-only.

See also

Connect

5.153.2.12 SSLOptions

```
property SSLOptions: TScSSLClientOptions;
```

Description

SSLOptions determines the behaviour of a TLS/SSL connection.

See also

IsSecure

5.153.2.13 State

```
property State: TScWebSocketState;
```

Description

The **State** property holds a value that indicates a WebSocket connection state. The expected values for states are defined in the TScWebSocketState enumeration.

If a connection was closed, the reason for this can be found in the CloseStatus property.

This property is read-only.

See also

CloseStatus

5.153.2.14 SubProtocolInUse

```
property SubProtocolInUse: string;
```

Description

Gets the sub-protocol used in the established connection. The **SubProtocolInUse** property is set when tehre is an established connection to a WebSocket server and returns a server response.

This property is read-only.

See also

Connect

5.153.2.15 WatchDogOptions

```
property WatchDogOptions: TScWatchDogOptions;
```

Description

WatchDogOptions determines the behaviour of the WatchDog mode. This mode is used for attempting to reconnect to the server automatically, when an unexpected disconnection is detected.

If a number of reconnect attempts exceeds the specified value, the OnConnectFail event occurs.

See also

OnConnectFail

5.153.3 Methods

5.153.3.1 Abort

```
procedure Abort;
```

Description

Call **Abort** to cancel a connection to a Web server. **Abort** sets the State property to the `sAborted` value.

Use this method only if a connection hangs. To close a connection normally, use the Close method.

See Also

Close

5.153.3.2 Close

```
procedure Close; overload;
procedure Close(Status: TScWebSocketCloseStatus; const Description:
string = ''); overload;
```

Description

Call **Close** to close a connection to a Web server. When calling this method, a Close control message is sent to the server and a response from the server is expected.

When executed successfully, the **Close** method sets the State property to the sClose value.

The `Status` parameter holds a value that indicates the reason for closing a connection. The CloseStatus property is set from the value of this parameter.

The `Description` parameter holds a string that describes a reason for closing a connection. The CloseStatusDescription property is set from the value of this parameter.

These values are sent to the server in the Close control message.

When calling the method without parameters, CloseStatus is set to csNormalClosure.

See Also

Connect

5.153.3.3 Connect

```
procedure Connect(const Uri: string; CancellationToken:
TScCancellationToken = nil); overload;
procedure Connect(CancellationToken: TScCancellationToken = nil);
overload;
```

Description

Establishes a connection to the specified WebSocket server. **Connect** sets the State property to the sOpen value.

The `URI` parameter is the URI that identifies the Internet resource. The RequestUri property is set from the value of this parameter.

If the method is called without parameters, URI of the request is taken from the RequestUri property.

See Also

Close
AfterConnect
BeforeConnect

5.153.3.4 Create

```
constructor Create(const URI: string);
```

Description

Create **TScWebSocketClient** instance and initialize it with the specified URI scheme.

The `URI` parameter is the URI that identifies the Internet resource. The RequestUri property is set from the value of this parameter.

See Also

Connect
RequestUri

5.153.3.5 Ping

```
procedure Ping; overload;
procedure Ping(const Data: TBytes); overload;
```

Description

Call the **Ping** message to send a ping control message to a WebSocket server and wait for a response pong message from the server. If after the time specified in the Options.ReadWriteTimeout property, the client does not get a response from the server, the method will generate an error.

A Ping message may serve either as a keepalive or as a means to verify that the remote endpoint is still responsive.

`Data` is an array of bytes, which contains the sent message body.

See Also

OnControlMessage
PingAsync
Pong

TScWebSocketControlMessageType

5.153.3.6 PingAsync

```
procedure PingAsync;
```

Description

Call the **PingAsync** message to send a ping control message to a WebSocket server. After sending a message, the method returns control immediately without waiting for an answer from the server.

You can find out about receiving a Pong response by handling the OnControlMessage event.

A Ping message may serve either as a keepalive or as a means to check that the remote endpoint is still responsive.

See Also

OnControlMessage

Ping

Pong

TScWebSocketControlMessageType

5.153.3.7 Pong

```
procedure Pong;
```

Description

Call the **Pong** message to send a pong control message to a WebSocket server. This kind of message can be used for a keepalive connection. A server does not send any response to this message.

See Also

Ping

PingAsync

5.153.3.8 Receive

```
function Receive(const Buffer: TBytes; Offset, Count: integer; out
MessageType: TScWebSocketMessageType; out EndOfMessage: boolean):
integer; overload;
function Receive(const Buffer: PByteArray; Count: integer; out
```

```
MessageType: TScWebSocketMessageType; out EndOfMessage: boolean):
integer; overload;
```

Description

Call **Receive** to read `Count` bytes from the received from the Web server data message into `Buffer`. **Receive** returns bytes count that was actually read.

If the client did not receive any messages from the server when calling the method, it will wait for the time specified in the Options.ReadWriteTimeout property. If no messages were received during this time, the method will return 0.

If at least one frame of the message was received, the method will return the data immediately, without waiting for the end of the message. If more than one message was received from the server, the method will return the data from the first one received.

To receive the entire message, use the ReceiveMessage method.

Parameters:

- `Buffer` - an array of bytes, where the body of the received message will be recorded into;
- `Offset` - zero-based byte offset in `Buffer` that points to the beginning of the data filling;
- `Count` - a maximum number of data that can be recorded to `Buffer`;
- `MessageType` - types of data message received from the server and returned by this parameter;
- `EndOfMessage` - returns a status whether the message was completely retrieved. If a message was not completely retrieved, the parameter is set to False.

Note: This method cannot be called in case if the OnMessage event handler is set. In this case, an exception will be generated.

See also

ReceiveMessage

5.153.3.9 ReceiveMessage

```
function ReceiveMessage(out MessageType: TScWebSocketMessageType):
TBytes; overload;
function ReceiveMessage(MSecTimeout: cardinal; out MessageType:
TScWebSocketMessageType): TBytes; overload;
procedure ReceiveMessage(Stream: TStream; out MessageType:
TScWebSocketMessageType); overload;
procedure ReceiveMessage(Stream: TStream; MSecTimeout: cardinal; out
MessageType: TScWebSocketMessageType); overload;
```

Description

Call **ReceiveMessage** to read the first message received from the Web server. **ReceiveMessage**

returns an array of bytes or fill the `Stream` parameter, where the received message body will be recorded.

Types of data message received from the server and returned by the `MessageType` parameter.

If the client did not receive any messages from the server when calling the method, it will wait for the time specified in the Options.ReadWriteTimeout property or in the `MSecTimeout` parameter. If no messages were received during this time, the method will generate an error. If at least part of the message was received, the method will wait for the end of the message.

To retrieve data as it is received from the server, use the Receive method.

Note: This method cannot be called if the OnMessage event handler is set. In this case, an exception will be generated.

See Also

Receive

5.153.3.1 Send

```
procedure Send(const Buffer; Count: integer; MessageType:
TScWebSocketMessageType = mtBinary; EndOfMessage: boolean = True);
overload;
procedure Send(const Buffer: TBytes; Offset, Count: integer;
MessageType: TScWebSocketMessageType = mtBinary; EndOfMessage: boolean =
True); overload;
procedure Send(const Str: string); overload;
```

Description

Call the **Send** message to send a data message to a WebSocket server.

The method allows splitting a large message into fragments and sequentially sending them to the server using the `EndOfMessage` parameter.

Parameters:

- `Buffer` - an array of bytes, which contains the sent message body;
- `Offset` - zero-based byte offset in `Buffer` that points to the beginning of the data location;
- `Count` - the amount of data, which will be sent;
- `MessageType` - types of the sent data message.
- `EndOfMessage` - determines a status whether the message is complete or whether data related to this message will be further added. This allows you to break one large message into fragments and send them to the server one by one. You can set automatic splitting of a large message into fragments by setting the TScWebSocketClientOptions.MaxFragmentSize property.

If the method is called with the only parameter `Str`, `MessageType` is considered equal to `mtText`, and the message is considered complete.

See Also

Receive

5.153.4 Events

5.153.4.1 AfterConnect

```
property AfterConnect: TNotifyEvent;
```

Description

Occurs after a connection to a WebSocket server is established.

See Also

AfterDisconnect
BeforeConnect
Connect

5.153.4.2 AfterDisconnect

```
property AfterDisconnect: TNotifyEvent;
```

Description

Occurs after the connection to a WebServer becomes closed.

To find out the reason for closing the connection by checking the CloseStatus property.

See Also

AfterConnect
Abort
Close
CloseStatus

5.153.4.3 BeforeConnect

```
property BeforeConnect: TNotifyEvent;
```

Description

Occurs immediately before establishing a connection to a WebSocket server.

See Also

AfterConnect

Connect

5.153.4.4 OnAsyncError

```
type
  TScErrorEvent = procedure (Sender: TObject; E: Exception) of object;

property OnAsyncError: TScErrorEvent;
```

Description

The **OnAsyncError** event occurs when an exception is raised during asynchronous data receiving.

`Sender` is an object that raised the exception.

`E` is the exception object that describes the exception.

See Also

OnMessage

5.153.4.5 OnConnectFail

```
property OnConnectFail: TNotifyEvent;
```

Description

Occurs in a WatchDog mode if the number of attempts to restore a broken connection has exceeded the value set in WatchDogOptions.Attempts.

The occurrence of this event means that the connection to the server is broken, and the client will no longer attempt to reconnect itself.

See Also

AfterConnect

WatchDogOptions

5.153.4.6 OnControlMessage

```
type
  TScWebSocketOnControlMessageEvent = procedure (Sender: TObject;
ControlMessageType: TScWebSocketControlMessageType) of object;

property OnControlMessage: TScWebSocketOnControlMessageEvent;
```

Description

The **OnControlMessage** event occurs if a control message was received from the WebSocket server.

`Sender` is the object that raised the event.

`ControlMessageType` defines the types of the control message received from the server.

See Also

OnMessage

Ping

PingAsync

TScWebSocketControlMessageType

5.153.4.7 OnMessage

```
type
  TScWebSocketOnMessageEvent = procedure (Sender: TObject; const Data:
TBytes; MessageType: TScWebSocketMessageType; EndOfMessage: boolean) of
object;

property OnMessage: TScWebSocketOnMessageEvent;
```

Description

The **OnMessage** event occurs if a data message was received from the WebSocket server.

If at least one message frame was received, the event occurs immediately, without waiting for the end of the message.

Parameters:

- `Sender` - the object that raised the event;
- `Data` - an array of bytes, which contains the body of the received message;
- `MessageType` - types of a data message received from the server.
- `EndOfMessage` - returns the status whether the message was read till the end. If the message was not read till the end, the parameter is set to False.

Note: if the event handler is set, the Receive and ReceiveMessage methods cannot be called. In this case, an exception will be generated when calling these methods.

See Also

OnControlMessage

5.154 TScFTPDirectoryListing

5.154.1 Description

Unit

ScFTPListParser

Description

The **TScFTPDirectoryListing** class is a descendant of the TCollenction class, and it is a container for TScFTPListItem objects.

TScFTPDirectoryListing is a container for items used in the structured directory listing in the FTP protocol.

See Also

TScFTPListItem

TScFTPClient.DirectoryListing

5.154.2 Properties

5.154.2.1 Items

```
property Items[Index: integer]: TScFTPListItem; default;
```

Description

Lists the TScFTPListItem object references.

Use **Items** to access objects in the list. **Items** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read the value at a specific index, or use **Items** with the Count property to iterate through the list.

Note: Items is the default property of TScFTPDirectoryListing. This means you can omit the property name.

5.154.3 Methods

5.154.3.1 Add

```
function Add: TScFTPListItem;
```

Description

Call **Add** to insert an object at the end of the list. **Add** places the object after the last item, even if the array contains nil references, and returns the index of the inserted object. The first object in the list has an index of 0.

See also

Items

5.154.3.2 IndexOf

```
function IndexOf(Item: TScFTPListItem): integer;
```

Description

Returns the index of the first object in the list with a specified value.

Call **IndexOf** to get the index for a specified object in the list, where the first object has index 0, the second object has index 1, and so on. If an object is not in the list, **IndexOf** returns -1. If an object appears more than once, **IndexOf** returns the index of the first appearance.

See also

Items

5.155 TScFTPListItem

5.155.1 Description

Unit

ScFTPListParser

Description

TScFTPListItem is a TCollectionItem descendant that represents an item in structured directory listing in the FTP protocol.

TScFTPListItem contains properties that represent information parsed from an FTP response.

See Also

TScFTPDirectoryListing.Items

5.155.2 Properties

5.155.2.1 BlockSize

```
property BlockSize: integer;
```

Description

The **BlockSize** property holds the size of blocks where the file is stored in.

5.155.2.2 Data

```
property Data: string;
```

Description

The **Data** property holds an initial value received from the FTP directory listing response.

5.155.2.3 FileName

```
property FileName: string;
```

Description

The **FileName** property represents the file name for the resource.

5.155.2.4 FileType

```
property FileType: TScFTPFileType;
```

Description

The **FileType** property represents the file type.

5.155.2.5 GroupName

```
property GroupName: string;
```

Description

The **GroupName** property holds the name of the group to which the file belongs.

5.155.2.6 LocalFileName

```
property LocalFileName: string;
```

Description

The **LocalFileName** property represents a suggested file name for the resource when stored on the local file system.

5.155.2.7 ModifiedAvailable

```
property ModifiedAvailable: boolean;
```

Description

The **ModifiedAvailable** property indicates if last modification time is available for the resource on the remote FTP server.

5.155.2.8 ModifiedDate

```
property ModifiedDate: TDateTime;
```

Description

The **ModifiedDate** property contains the time of the last file modification.

5.155.2.9 ModifiedDateGMT

```
property ModifiedDateGMT: TDateTime;
```

Description

The **ModifiedDateGMT** property contains the time of the last file modification.

This time is presented in the UTC time scale.

5.155.2.10 NumberBlocks

```
property NumberBlocks: integer;
```

Description

The **NumberBlocks** property holds a number of blocks where the file is stored in.

5.155.2.11 OwnerName

```
property OwnerName: string;
```

Description

The **OwnerName** property holds the name of the file owner.

5.155.2.12 Permissions

```
property Permissions: string;
```

Description

The **Permissions** property contains the flags specifying file permissions.

5.155.2.13 PermissionsDisplay

```
property PermissionsDisplay: string;
```

Description

The **PermissionsDisplay** property contains the flags specifying file permissions in the user-displayable representation.

5.155.2.14 Size

```
property Size: Int64;
```

Description

Use the **Size** property to specify the number of bytes that can be read from the file, or in other words, the location of the end-of-file.

5.155.2.15 SizeAvailable

```
property SizeAvailable: boolean;
```

Description

The **SizeAvailable** property indicates if file size information is available for the resource on the remote FTP server.

5.156 TScFTPClientOptions

5.156.1 Description

Unit

ScFTPClient

Description

The **TScFTPClientOptions** class determines behaviour of an FTP client.

See also

TScFTPClient.Options

5.156.2 Properties

5.156.2.1 BindAddress

```
property BindAddress: string;
```

Description

Determines the TCP/IP address on the local machine as the source address of the connection. Only useful on systems with more than one TCP/IP address.

5.156.2.2 BlockSize

```
property BlockSize: integer; default 65536;
```

Description

Use the **BlockSize** property to determine the maximum size of the data block that will be sent or received as one query to the FTP server when uploading or downloading a file. Use this property to increase the application performance.

The default value is 65536 byte.

5.156.2.3 IgnoreServerPassiveHost

```
property IgnoreServerPassiveHost: boolean; default False;
```

Description

The **IgnoreServerPassiveHost** property determines the IP address to be used in Passive mode of data channel creation.

If **IgnoreServerPassiveHost** is False, then a client when creating a data channel will connect to the IP address obtained from the FTP server as a result of the corresponding request.

If **IgnoreServerPassiveHost** is True, then a client when creating a data channel will connect to the same IP address, which the current FTP connection is established with.

The default value is False.

5.156.2.4 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is `ivIPv4`.

5.156.2.5 SocketReceiveBufferSize

```
property SocketReceiveBufferSize: integer; default 32768;
```

Description

Use the **SocketReceiveBufferSize** property to determine the total per-socket buffer space reserved for receives. This value is set by the OS functions to the socket.

Use this property to increase the application performance.

The default value is 32768.

See also

SocketSendBufferSize

5.156.2.6 SocketSendBufferSize

```
property SocketSendBufferSize: integer; default 32768;
```

Description

Use the **SocketSendBufferSize** property to determine the total per-socket buffer space reserved for sends.This value is set by the OS functions to the socket.

Use this property to increase the application performance.

The default value is 32768.

See also

SocketReceiveBufferSize

5.156.2.7 TCPKeepAlive

```
property TCPKeepAlive: boolean; default True;
```

Description

The **TCPKeepAlive** property specifies whether the system should send TCP keep alive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed.

The default value is True.

5.156.2.8 UseClearingControlChannel

```
property UseClearingControlChannel: boolean; default False;
```

Description

The **UseClearingControlChannel** property determined whether the Clear Command Channel (CCC) command will be used for the FTP client session, as described in the Internet Standards documents RFC 2228.

The default value is False.

5.156.2.9 UseExtendedDataAddress

```
property UseExtendedDataAddress: boolean; default False;
```

Description

The **UseExtendedDataAddress** property declares if RFC 2428 NAT extensions will be available for the FTP client session.

Set **UseExtendedDataAddress** to True, to use the 'EPSV' and 'EPRT' commands when opening a

data channel. Else, the 'PASV' and 'PORT' command will be used.

The default value is False.

5.156.2.1 UseExtList

```
property UseExtList: boolean; default True;
```

Description

The **UseExtList** property specifies whether the TScFTPClient.ExtListDirDetails method will be used instead of TScFTPClient.ListDirDetails.

The TScFTPClient.ExtListDirDetails method is used to support the MLSD extension command.

The default value is True.

5.156.2.1 UseNATFastConnection

```
property UseNATFastConnection: boolean; default False;
```

Description

The **UseNATFastConnection** property declares if FTP Extensions for IPv6 and NATs will be used by the current FTP session.

Using these extensions benefits performance of transfers that traverse firewalls or Network Address Translators (NAT).

The default value is False.

5.157 TScFTPClient

5.157.1 Description

Unit

ScFTPClient

Description

The **TScFTPClient** component implements functionality of FTP/FTPS client.

FTP protocol is used to implement remote file system service, as well as file transfer service.

FTP client supports data transfer over a secure TLS/SSL channel (FTPS protocol). To manage security, you can use the TLSMode property.

To download a file, use the Dowload method, to upload - Upload. To obtain a list of files on the server - the ListDir, ListDirDetails, and ExtListDirDetails methods.

The **TScFTPClient** class throws an EScFTPError exception when an error occurs during execution of

any command to the FTP server. The EScFTPError.FTPErrorCode property contains a value that indicates the source of the error.

See Also

Connected

IsSecure

Dowload

Upload

ListDir

ListDirDetails

5.157.2 Properties

5.157.2.1 AccountInfo

```
property AccountInfo: string;
```

Description

The **AccountInfo** property represents the account information to use for the connection to the FTP server. Account info is used to provide additional login information for the FTP server.

AccountInfo is used in the Login method when sending the FTP 'ACCT' command.

See Also

Login

Username

5.157.2.2 Active

```
property Active: boolean;
```

Description

Determines whether the connection to an FTP server is established.

The property is set to True after a successful connection to the server and to False after a force or unexpected disconnection.

This property is read-only.

See Also

Connect

Disconnect

5.157.2.3 AuthCommand

```
property AuthCommand: TScFTPAuthCommand; default acAuto;
```

Description

AuthCommand represents the argument used with the FTP 'AUTH' command in the TLS handshake protocol exchange. It is used when performing authentication in the Login method.

The default value is acAuto to try each of the AUTH command values.

See Also

Login

5.157.2.4 CEncoding

```
property CEncoding: Encoding;
```

Description

Specifies the `Encoding` object that will be used to encode commands sent to the FTP server and decode responses received from the FTP server.

By default the system locale is used.

See Also

UseUTF8

5.157.2.5 DataIP

```
property DataIP: string;
```

Description

DataIP indicates the IP address to use when binding the data channel for the FTP client session. If a value is empty, a local IP address of the current connection to a FTP server is taken.

DataIP with DataPort are used when uploading and downloading files, as well as when obtaining a list of files to create a data channel that actively listens to a server connection in the non Passive mode. FTP client will send this address using the 'PORT' or 'EPRT' commands.

See Also

DataPort
ExtListDirDetails
ListDir
ListDirDetails
Dowload
Upload

5.157.2.6 DataPort

```
property DataPort: integer; default 0;
```

Description

DataPort indicates the port number to use when binding the data channel for the FTP client session. If value is 0, it means that the data port is assigned when the data channel connection is bound.

DataIP with **DataPort** are used when uploading and downloading files, as well as when obtaining a list of files to create a data channel that actively listens for a server connection in non Passive mode. FTP client will send this port using the 'PORT' or 'EPRT' commands.

The default value is 0.

See Also

DataIP

5.157.2.7 DirectoryFormat

```
property DirectoryFormat: string;
```

Description

DirectoryFormat determines the directory listing format for the parser used to populate the items in the DirectoryListing property.

This property is read-only.

See Also

DirectoryListing

5.157.2.8 DirectoryListing

```
property DirectoryListing: TScFTPDirectoryListing;
```

Description

DirectoryListing holds a list of TScFTPListItem objects that contain information about files and directories returned from the FTP 'LIST', 'NLST', 'MLSD' or 'MLST' commands.

These FTP commands are executed by the ListDir, ListDirDetails and ExtListDirDetails methods, and correspondingly, calling these methods causes changing the **DirectoryListing** list.

This property is read-only.

See Also

AfterParseListing

BeforeParseListing

DirectoryFormat

ExtListDirDetails

ListDir

ListDirDetails

5.157.2.9 EncryptDataChannel

```
property EncryptDataChannel: boolean; default False;
```

Description

The **EncryptDataChannel** property determines whether a data channel connection is protected using a TLS/SSL protocol.

EncryptDataChannel is used when uploading and downloading files, as well as when obtaining a list of files to establish a TLS connection for a data channel that performs data transfer operations.

Set the property to True, to transfer data in a protected mode.

The default value is False.

See Also

ExtListDirDetails

ListDir

ListDirDetails

Dowload

Upload

5.157.2.10 FormattedReply

```
property FormattedReply: TStringList;
```

Description

The **FormattedReply** property represents the formatted response message received during execution of the latest command request to the FTP server.

To obtain the response code, you can use the ReplyCode ptoperty.

This property is read-only.

See Also

OnError

ReplyCode

RaiseLastCommandError

5.157.2.11 HostName

```
property HostName: string;
```

Description

Specifies the host name or the IP address to connect to the FTP server.

The Connect method uses values in the **HostName** and Port properties to establish a connection for the FTP session.

See Also

Connect

Port

5.157.2.12 IsCompressionSupported

```
property IsCompressionSupported: boolean;
```

Description

Determines whether the current FTP server supports data compression. This property is set automatically when the Login method is called while obtaining a list of files, as well as while uploading and downloading files.

This property is read-only.

See Also

Login

ExtListDirDetails

ListDir

ListDirDetails

Dowload

Upload

5.157.2.13 IsSecure

```
property IsSecure: boolean;
```

Description

Determines whether the connection to FTP server is protected.

When **IsSecure** is set to False, data is transferred as plaintext.

When **IsSecure** is set to True, the TLS/SSL protocol is used, and data is transferred in an encrypted form.

This property is read-only.

See also

SSLOptions

5.157.2.14 IsUsedExtendedDataAddress

```
property IsUsedExtendedDataAddress: boolean;
```

Description

The **IsUsedExtendedDataAddress** property indicates if RFC 2428 NAT extensions are available for the FTP client session.

If **IsUsedExtendedDataAddress** is True, the 'EPSV' and 'EPRT' commands are used when opening a data channel. Else, the 'PASV' and 'PORT' command are used.

This property is read-only.

See Also

UsePassive

5.157.2.15 IsUsedNATFastConnection

```
property IsUsedNATFastConnection: boolean;
```

Description

The **IsUsedNATFastConnection** property indicates if FTP Extensions for IPv6 and NATs is supported by the current FTP server.

Using these extensions benefits performance of transfers that traverse firewalls or Network Address Translators (NAT).

This property is read-only.

See Also

TScFTPClientOptions.UseNATFastConnection

5.157.2.16 ListenTimeout

```
property ListenTimeout: integer; default 15;
```

Description

The **ListenTimeout** property determines the time interval in seconds during which the client will actively listen to a server connection when creating a data channel in the non Passive mode.

The default value is 15 seconds.

See Also

Passive

ExtListDirDetails

ListDir

ListDirDetails

Dowload

Upload

5.157.2.17 Options

```
property Options: TScFTPClientOptions;
```

Description

Options determines behaviour of an FTP client.

See Also

Connect

5.157.2.15 Password

```
property Password: string;
```

Description

Password is used to indicate the authentication credentials used when logging in to the FTP server. User name is specified in the Username property.

These values are sent to the FTP server using the USER and PASS commands in the Login method.

See Also

Login

Username

5.157.2.16 Port

```
property Port: integer; default 21;
```

Description

Specifies the port number for TCP/IP connection to the FTP server.

The Connect method uses values in the HostName and **Port** properties to establish a connection for the FTP session.

The default value is 21 port number.

See Also

Connect

HostName

5.157.2.17 ProxyOptions

```
property ProxyOptions: TProxyOptions;
```

Description

The **ProxyOptions** property holds a TProxyOptions object that contains settings for proxy connection.

If it is necessary to connect to server in another network, sometimes the client can reach it only through proxy. In this case you have to setup **ProxyOptions**.

See Also

Connect

5.157.2.21 ReplyCode

```
property ReplyCode: integer;
```

Description

The **ReplyCode** property represents the response code received during execution of the latest command request to the FTP server.

To get the formatted response message, you can use the FormattedReply ptoperty.

This property is read-only.

See Also

FormattedReply

OnError

RaiseLastCommandError

5.157.2.22 ServerDescription

```
property ServerDescription: string;
```

Description

The **ServerDescription** property contains a description for the remote FTP server. **ServerDescription** is automatically set in the Connect method.

This property is read-only.

See Also

Connect

SystemDescription

5.157.2.23 SSLOptions

```
property SSLOptions: TScSSLClientOptions;
```

Description

SSLOptions determines behaviour of a TLS/SSL connection.

See also

Connect

IsSecure

5.157.2.24 SystemDescription

```
property SystemDescription: string;
```

Description

The **SystemDescription** property contains a description of the operating system for the remote FTP server.

SystemDescription is automatically set in the Connect method using the result of the FTP 'SYST' command.

This property is read-only.

See Also

Connect

ServerDescription

5.157.2.25 Timeout

```
property Timeout: integer; default 15;
```

Description

Determines the time interval in seconds during which the client will wait for a response from the server when connecting, or wait for any data from the server when reading, or passing data to the server. After the time has expired, methods return the result and control to the program.

The default value is 15 seconds.

5.157.2.26 TLSMode

```
property TLSMode: TScTLSMode; default tmDisableTLS;
```

Description

The **TLSMode** property indicates the level of Transport Layer Security (TLS) required for control channel and data channel connections in the FTP client.

If **TLSMode** is set to the `tmImplicitTLS` value, TLS protocol will be set in the Connect method.

If **TLSMode** is set to the `tmRequireExplicitTLS` or `tmAllowExplicitTLS` values, TLS

protocol will set in the Login method.

The default value is the `tmDisableTLS` value that means not to use a TLS protocol.

See Also

Connect

Login

5.157.2.27 TransferType

```
property TransferType: TScFTPTransferType; default ttASCII;
```

Description

The **TransferType** property determines the file transfer type currently in use for the FTP client.

TransferType is used when uploading and downloading files, as well as when obtaining a list of files.

The default value is ttASCII.

See Also

ExtListDirDetails

ListDir

ListDirDetails

Dowload

Upload

5.157.2.28 Uri

```
property Uri: string;
```

Description

Gets or sets the Uniform Resource Identifier (URI) of the FTP server.

The **Uri** property can consist of a hostname and an optional port number. **Uri** without port information implies the default port (port 21).

An example complying with requirements that specifies a port of 2121 would be the following value for the **Uri** property: 'ftp://host.com:2121'.

When setting the **Uri** property, there is value parsing and the HostName, Port, Username and Password properties are reset.

See Also

Connect

HostName

Port

5.157.2.2 UseCompression

```
property UseCompression: boolean; default False;
```

Description

The **UseCompression** property determines whether data compression will be used when transferring data via the data channel.

UseCompression is used when uploading and downloading files, as well as when obtaining a list of files.

Set the property to True, to compress the transmitted data.

The default value is False.

See Also

ExtListDirDetails

ListDir

ListDirDetails

Dowload

Upload

5.157.2.3 UsePassive

```
property UsePassive: boolean; default False;
```

Description

The **UsePassive** property indicates how the data channel connection for a FTP session is established.

UsePassive is used when uploading and downloading files, as well as when obtaining a list of files.

When **UsePassive** is True the FTP server will listen for the FTP client to connect on a data port specified in the response to the FTP 'PASV' or 'EPSV' commands. 'EPSV' is used when IsUsedExtendedDataAddress is True.

When **UsePassive** is False the FTP client will listen to the FTP server to connect on a data port specified in the FTP 'PORT' or 'EPRT' commands. 'EPRT' is used when IsUsedExtendedDataAddress is True. In this mode the DataIP and DataPort properties determine the address where a FTP client will listen to the server. The ListenTimeout property determines the time interval during which the client will actively listen for a server connection.

The default value is False.

See Also

IsUsedExtendedDataAddress
DataIP
DataPort
ListenTimeout
ExtListDirDetails
ListDir
ListDirDetails
Dowload
Upload

5.157.2.31 Username

```
property Username: string;
```

Description

User name is used as an authentication identity when logging into the FTP server.

Password for user authentication is specified in the Password property.

These values are sent to the FTP server using the USER and PASS commands in the Login method.

See Also

Login
Password

5.157.2.32 UseUTF8

```
property UseUTF8: boolean;
```

Description

Determines whether the UTF8 encoding will be used to encode commands sent to the FTP server and decode responses received from the FTP server.

When the property is True, the UTF8 encoding is used. Otherwise, the system locale is used.

Changing the value of this property may affect CEncoding.

See Also

CEncoding

5.157.3 Methods

5.157.3.1 Abort

```
procedure Abort;
```

Description

Call **Abort** to interrupt the current FTP data channel. Abort sends the FTP 'ABOR' command to the server to halt the current operation, and after that closes the data channel.

Data channel is used when uploading and downloading files, as well as when obtaining a list of files.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ExtListDirDetails

ListDir

ListDirDetails

Dowload

Upload

5.157.3.2 Account

```
procedure Account(const AAccountInfo: string);
```

Description

Call the **Account** method to provide additional account information for the connection to the FTP server.

`AAccountInfo` is used as an argument when sending the FTP 'ACCT' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

AccountInfo

5.157.3.3 Allocate

```
procedure Allocate(Size: integer);
```

Description

The **Allocate** method call pre-allocates space on an FTP server before uploading a file.

`Size` is used as an argument when sending the FTP 'ALLO' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

Upload

5.157.3.4 ChangeDir

```
procedure ChangeDir(const Path: string);
```

Description

Call the **ChangeDir** method to change the current directory on the FTP server file system to the directory specified in the `Path` parameter.

`Path` is used as an argument when sending the FTP 'CWD' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ChangeDirUp

Dowload

Upload

5.157.3.5 ChangeDirUp

```
procedure ChangeDirUp;
```

Description

Call the **ChangeDirUp** method to change the current directory to the parent directory in the file system for the FTP server.

ChangeDirUp sends the FTP 'CDUP' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ChangeDir

Dowload

Upload

5.157.3.6 Connect

```
procedure Connect;
```

Description

Establishes a connection to the specified FTP server. **Connect** sets the Active property to True.

The **Connect** method uses values in the HostName and Port properties to establish a connection for the FTP session.

If it is necessary to connect to a server in another network, sometimes the client can reach it only through proxy. In this case you have to setup ProxyOptions.

See Also

Disconnect

HostName

Port

ProxyOptions

5.157.3.7 Delete

```
procedure Delete(const Filename: string);
```

Description

Call the **Delete** method to remove the file specified in the `Filename` parameter from the file system on the FTP server.

`Filename` is used as an argument when sending the FTP 'DELE' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

Rename

5.157.3.8 Disconnect

```
procedure Disconnect;
```

Description

Closes an existent connection to the FTP server. **Disconnect** sets the Active property to False.

See Also

Connect

5.157.3.9 Dowload

```
procedure Dowload(const SourceFile, DestFile: string; Overwrite: boolean
= False; StartPos: Int64 = -1); overload;
procedure Dowload(const SourceFile: string; Dest: TStream; StartPos:
Int64 = -1); overload;
```

Description

Call the **Download** method to copy a file from the FTP server to the local machine.

`SourceFile` is used as an argument when sending the FTP 'RETR' command.

Parameters:

- `SourceFile` - holds the initial path to the file that is being copied.
- `DestFile` - holds the destination path to copy the file to.
- `Dest` - holds the destination data stream to copy the file to.
- `Overwrite` - specifies whether to overwrite the file with the same name if it exists.
- `StartPos` - the offset in bytes relative to the beginning of the file that the downloading starts at.

File downloading can occur in an active and passive modes (see UsePassive).

The EncryptDataChannel property determines whether the connection will be protected using a TLS/ SSL protocol.

The UseCompression property determines whether data compression will be used when transferring a file.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

AfterDownloadFile

BeforeDownloadFile

EncryptDataChannel

UseCompression

Upload

5.157.3.10 ExtListDirDetails

```
procedure ExtListDirDetails(List: TStrings = nil; const Path: string =
'');
```

Description

Call the **ExtListDirDetails** method to retrieve a detail list of files or directories in the directory on the FTP server. **ExtListDirDetails** is used to support the MLSD extension command.

`Path` is used as an argument when sending the FTP 'MLSD' command. If `Path` is an empty string, the current directory is used.

If `List` is not nil, the resulting list will be assigned to this object in a text format.

ExtListDirDetails fills the DirectoryListing list of objects that contain detailed information about files and directories returned by this request.

Getting a list of files can occur in an active and a passive modes (see UsePassive).

The EncryptDataChannel property determines whether the connection will be protected using a TLS/SSL protocol.

The UseCompression property determines whether data compression will be used when transferring data.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

AfterRetrieveList

BeforeRetrieveList

DirectoryListing

EncryptDataChannel

ListDir

ListDirDetails

UseCompression

5.157.3.11 GetCurrentDir

```
function GetCurrentDir: string;
```

Description

Call the **GetCurrentDir** method to retrieve the name of the current working directory from the file system on the FTP server.

See Also

ChangeDir

5.157.3.12 GetListItem

```
procedure GetListItem(List: TStrings; DirList: TScFTPDirectoryListing;
const Item: string);
```

Description

Call the **GetListItem** method to retrieve data about the exact object (file, directory, etc.) from the file system on the FTP server named by the `Item` parameter. **GetListItem** is used to support the MLST extension command.

`Item` is used as an argument when sending the FTP 'MLST' command.

The resulting list will be assigned to the List object in a text format.

If `DirList` is not nil, the information parsed in the corresponding object will be assigned to this object.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ChangeDirUp

5.157.3.13 Help

```
procedure Help(const Command: string = ''; HelpInfo: TStrings = nil);
```

Description

Call the **Help** method to retrieve helpful information about FTP server capabilities.

`Command` is used as an argument when sending the FTP 'HELP' command to request more specific information.

The request result will be assigned to the `HelpInfo` object.

The response code received during execution of this command request is contained in the ReplyCode property.

5.157.3.14 IsExtSupported

```
function IsExtSupported(const Command: string): boolean;
```

Description

Call the **IsExtSupported** method to determine if the specified `Command` is a supported FTP extension command for the current FTP server.

5.157.3.13 IsTLSSupported

```
function IsTLSSupported: boolean;
```

Description

Call the **IsTLSSupported** method to determine if the explicit TLS mode is supported for the current FTP server.

See Also

AuthCommand

TLSMode

5.157.3.12 ListDir

```
procedure ListDir(List: TStrings = nil; const Path: string = '');
```

Description

Call the **ListDir** method to retrieve a list of files or directories only in the directory on the FTP server without any describing information.

`Path` is used as an argument when sending the FTP 'NLST' command. If `Path` is an empty string, the current directory is used.

If `List` is not nil, the resulting list will be assigned to this object in a text format.

ListDir fills the DirectoryListing list of objects that contain names of files and directories returned by this request.

Getting a list of files can occur in an active and a passive modes (see UsePassive).

The EncryptDataChannel property determines whether the connection will be protected using a TLS/SSL protocol.

The UseCompression property determines whether data compression will be used when transferring data.

The response code received during execution of this command request is contained in the ReplyCode

property.

See Also

AfterRetrieveList

BeforeRetrieveList

DirectoryListing

EncryptDataChannel

ExtListDirDetails

ListDirDetails

UseCompression

5.157.3.17 ListDirDetails

```
procedure ListDirDetails(List: TStrings = nil; const Path: string = '');
```

Description

Call the **ListDirDetails** method to retrieve a detail list of files or directories in the directory on the FTP server.

`Path` is used as an argument when sending the FTP 'LIST' command. If `Path` is an empty string, the current directory is used.

If `List` is not nil, the resulting list will be assigned to this object in a text format.

ListDirDetails fills the DirectoryListing list of objects that contain detail information about files and directories returned by this request.

Getting a list of files can occur in an active and a passive modes (see UsePassive).

The EncryptDataChannel property determines whether the connection will be protected using a TLS/ SSL protocol.

The UseCompression property determines whether data compression will be used when transferring data.

The response code received during execution of this command request is contains in the ReplyCode property.

See Also

AfterRetrieveList

BeforeRetrieveList

DirectoryListing

EncryptDataChannel

ExtListDirDetails

ListDir

UseCompression

5.157.3.18 Login

```
procedure Login; overload;
procedure Login(const AUsername, APassword, AAccountInfo: string);
overload;
```

Description

Call the **Login** method to provide authentication for the FTP client connection to the remote FTP server.

`AUsername` is used as an argument when sending the FTP 'USER' command.

`APassword` is used as an argument when sending the FTP 'PASS' command.

`AAccountInfo` is used as an argument when sending the FTP 'ACCT' command.

When calling the method without parameters, the Username, Password, and AccountInfo properties are used.

If the TLSMode proeprty is equal to tmRequireExplicitTLS or tmAllowExplicitTLS values, the client first tries to explicitly establish a TLS connection using the 'AUTH' argument from the AuthCommand property for the command.

See Also

Account

AccountInfo

AuthCommand

Password

Username

5.157.3.19 MakeDir

```
procedure MakeDir(const Path: string);
```

Description

Call the **MakeDir** method to create a directory in the file system on the FTP server.

`Path` is used as an argument when sending the FTP 'MKD' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

RemoveDir

5.157.3.2 MountStructure

```
procedure MountStructure(const Path: string);
```

Description

Call the **MountStructure** method to mount a different file system data structure without altering its login or accounting information.

`Path` is used as an argument when sending the FTP 'SMNT' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

Login

5.157.3.2 Noop

```
procedure Noop;
```

Description

Noop does not affect any parameters or previously entered commands and can be used to keep-alive connection with the FTP server.

5.157.3.2 RaiseLastCommandError

```
procedure RaiseLastCommandError;
```

Description

Raises the EScFTPError exception by setting the ReplyCode value as an error code, and as a message a result obtained during execution of the latest command to the FTP server.

See Also

ReplyCode

FormattedReply

5.157.3.23 Reinitialize

```
procedure Reinitialize;
```

Description

Call the **Reinitialize** method to clear the user account info flushing all I/O except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the control connection is left open.

The response code received during execution of this command request is contained in the ReplyCode property.

Reinitialize clears the DirectoryListing list. Also, if TLSMode <> tmImplicitTLS, TLS protocol finishes working and the connection becomes insecure.

See Also

Connect

5.157.3.24 RemoveDir

```
procedure RemoveDir(const Path: string);
```

Description

Call the **RemoveDir** method to remove a specified directory from the file system on the FTP server.

`Path` is used as an argument when sending the FTP 'RMD' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

MakeDir

5.157.3.25 Rename

```
procedure Rename(const OldPath, NewPath: string);
```

Description

Call the **Rename** method to rename the existing file specified in the `OldPath` parameter to the new `NewPath` name for the file system on the FTP server.

`OldPath` is used as an argument when sending the FTP 'RNFR' command.

`NewPath` is used as an argument when sending the FTP 'RNTO' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

Delete

5.157.3.26 SendCmd

```
function SendCmd(const Command: string; const AllowedResponses: array of integer): integer;
```

Description

The **SendCmd** method sends FTP command specified in the `Command` parameter to an FTP server.

You can specify a list of available response codes in the `AllowedResponses` parameter. If a server returns a response code not included in this list, the method will generate a corresponding exception.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ReplyCode

5.157.3.27 SendFileStructure

```
procedure SendFileStructure(const Value: TScFTPFileStructure);
```

Description

Call the **SendFileStructure** method to define the way in which data is represented in FTP data transfer operations.

`Value` is used as an argument when sending the FTP 'STRU' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ListDir

5.157.3.28 SetCommandOptions

```
procedure SetCommandOptions(const Command, Options: string);
```

Description

Call the **SetCommandOptions** method to allow specifying the desired behavior of a server process when the FTP command is executed.

`Command` and `Options` are used as arguments when sending the FTP 'OPTS' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

IsExtSupported

5.157.3.25 Site

```
procedure Site(const Command: string);
```

Description

Call the **Site** method to request services specific for the host system and essential for file transfer, but not sufficiently universal to be included as commands in the protocol.

`Command` is used as an argument when sending the FTP 'SITE' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

Help

5.157.3.26 SiteToSiteTransfer

```
class procedure SiteToSiteTransfer(SourceClient, DestClient:
TScFTPClient; const SourceFile, DestFile: string);
```

Description

Transfers files between FTP servers.

To copy a file from one FTP server to another, you establish a connection to the required servers using two TScFTPClient objects.

Files are transferred in the passive mode (see UsePassive), i.e. a data connection is established directly between the two servers that the clients have connected to.

Parameters:

- `SourceClient` - the FTP server that hosts the file to be copied to another FTP server;

- `DestClient` - the FTP server that the file will be copied to;
- `SourceFile` - the path to the file that will be copied;
- `DestFile` - the destination path to copy the file to.

The SourceClient.EncryptDataChannel property determines whether the connection will be protected using the TLS/SSL protocol.

See Also

Dowload

Upload

5.157.3.33 Size

```
function Size(const FileName: string): Int64;
```

Description

Call the **Size** method to retrieve a file size information for the specified file from the FTP server.

`FileName` is used as an argument when sending the FTP 'SIZE' command.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ListDir

5.157.3.34 Status

```
procedure Status(const Path: string = ''; StatusResponse: TStrings =
nil);
```

Description

Call the **Status** method to get the status of the transfer operation in progress during a file transfer.

`Path` is used as an argument when sending the FTP 'STAT' command.

The content of the server response will be assigned to the `StatusResponse` parameter.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

Dowload

Upload

5.157.3.33 Upload

```
procedure Upload(const SourceFile, DestFile: string; Append: boolean =
False; StartPos: Int64 = -1); overload;
procedure Upload(Source: TStream; const DestFile: string; Append: boolean
= False; StartPos: Int64 = -1); overload;
```

Description

Call the **Upload** method to copy a file from the local machine to the FTP server.

Parameters:

- `SourceFile` - holds the initial path to the file that is being copied.
- `Source` - holds the source data stream that is being copied.
- `DestFile` - holds the destination path to copy the file to.
- `Append` - specifies whether the data will be added to the end of the file or the data of the existing file will be overwritten.
- `StartPos` - the offset in bytes relative to the beginning of the file that the uploading started at.

File uploading can occur in active and passive modes (see UsePassive).

The EncryptDataChannel property determines whether the connection will be secured using TLS/SSL protocol.

The UseCompression property determines whether data compression will be used when transferring data.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

AfterUploadFile

BeforeUploadFile

Dowload

EncryptDataChannel

UseCompression

UploadWithUniqueName

5.157.3.34 UploadWithUniqueName

```
procedure UploadWithUniqueName(const SourceFile: string; StartPos: Int64
= -1); overload;
```

```
procedure UploadWithUniqueName(Source: TStream; StartPos: Int64 = -1);
overload;
```

Description

Call the **UploadWithUniqueName** method to copy a file from the local machine to the FTP server, where resultant file is to be created in the current directory under a name unique to that directory.

Parameters:

- `SourceFile` - holds the initial path to the file that is being copied.
- `Source` - holds the source data stream that is being copied.
- `StartPos` - the offset in bytes relative to the beginning of the file that the uploading started at.

File uploading can occur in active and passive modes (see UsePassive).

The EncryptDataChannel property determines whether the connection will be secured using TLS/SSL protocol.

The UseCompression property determines whether data compression will be used when transferring data.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

AfterUploadFile

BeforeUploadFile

Dowload

EncryptDataChannel

UseCompression

Upload

5.157.4 Events

5.157.4.1 AfterConnect

```
property AfterConnect: TNotifyEvent;
```

Description

Occurs after a connection to an FTP/FTPS server is established.

See Also

Active

AfterDisconnect

BeforeConnect

BeforeDisconnect

5.157.4.2 AfterDisconnect

```
property AfterDisconnect: TNotifyEvent;
```

Description

Occurs after the connection to an FTP/FTPS server becomes closed.

See Also

Active
AfterConnect
BeforeConnect
BeforeDisconnect

5.157.4.3 AfterDownloadFile

```
type
  TScFTPAfterDownloadFileEvent = procedure(Sender: TObject; const
SourceFile: string; Dest: TStream; StartPos: Int64) of object;

property AfterDownloadFile: TScFTPAfterDownloadFileEvent;
```

Description

Occurs after a file was downloaded from the FTP server to the local machine.

Parameters:

- `Sender` - the object that raised the event;
- `SourceFile` - holds the initial path to the file that was being copied;
- `Dest` - holds the destination data stream where the file was copied;
- `StartPos` - the offset in bytes relative to the beginning of the downloaded file.

See Also

Dowload

5.157.4.4 AfterParseListing

```
property AfterParseListing: TNotifyEvent;
```

Description

Occurs when reading a list of files and directories using the DirectoryListing property after parsing text information returned by the server.

See Also

DirectoryListing

5.157.4.5 AfterRetrieveList

```
type
  TScFTPOnRetrieveListEvent = procedure(Sender: TObject; const Path:
string) of object;

property AfterRetrieveList: TScFTPOnRetrieveListEvent;
```

Description

Occurs after retrieving a list of files and directories in the directory on the FTP server using the ExtListDirDetails, ListDir and ListDirDetails methods.

Parameters:

- `Sender` - the object that raised the event;
- `Path` - holds the directory on the FTP server.

See Also

ExtListDirDetails

ListDir

ListDirDetails

5.157.4.6 AfterUploadFile

```
type
  TScFTPAfterUploadFileEvent = procedure(Sender: TObject; Source:
TStream; const DestFile: string; StartPos: Int64) of object;

property AfterUploadFile: TScFTPAfterUploadFileEvent;
```

Description

Occurs after a file was uploaded from the local machine to the FTP server.

Parameters:

- `Sender` - the object that raised the event;
- `Source` - holds the source data stream that was being copied;
- `DestFile` - holds the destination path where the file was copied;
- `StartPos` - the offset in bytes relative to the beginning of the file that the uploading started at.

See Also

Upload

5.157.4.7 BeforeConnect

```
property BeforeConnect: TNotifyEvent;
```

Description

Occurs immediately before establishing a connection to an FTP/FTPS server.

See Also

Active
AfterConnect
AfterDisconnect
BeforeDisconnect

5.157.4.8 BeforeDisconnect

```
property BeforeDisconnect: TNotifyEvent;
```

Description

Occurs immediately before the connection to an FTP/FTPS server becomes closed.

See Also

Active
AfterConnect
AfterDisconnect

BeforeConnect

5.157.4.9 BeforeDownloadFile

```
type
  TScFTPBeforeDownloadFileEvent = procedure(Sender: TObject; const
SourceFile: string; Dest: TStream; var StartPos: Int64) of object;

property BeforeDownloadFile: TScFTPBeforeDownloadFileEvent;
```

Description

Occurs before a file will be downloaded from the FTP server to the local machine.

Parameters:

- `Sender` - the object that raised the event;
- `SourceFile` - holds the initial path to the file that is being copied;
- `Dest` - holds the destination data stream to copy the file to;
- `StartPos` - the offset in bytes relative to the beginning of the file that the downloading starts at.

See Also

Dowload

5.157.4.10 BeforeParseListing

```
property BeforeParseListing: TNotifyEvent;
```

Description

Occurs when reading a list of files and directories using the DirectoryListing proeprty before parsing text information returned by the server.

See Also

DirectoryListing

5.157.4.11 BeforeRetrieveList

```
type
  TScFTPOnRetrieveListEvent = procedure(Sender: TObject; const Path:
```

```
string) of object;

property BeforeRetrieveList: TScFTPOnRetrieveListEvent;
```

Description

Occurs before retrieving a list of files and directories in the directory on the FTP server using the ExtListDirDetails, ListDir and ListDirDetails methods.

Parameters:

- `Sender` - the object that raised the event;
- `Path` - holds the directory on the FTP server.

See Also

ExtListDirDetails

ListDir

ListDirDetails

5.157.4.11 BeforeUploadFile

```
type
  TScFTPBeforeUploadFileEvent = procedure(Sender: TObject; Source:
TStream; const DestFile: string; var StartPos: Int64) of object;

property BeforeUploadFile: TScFTPBeforeUploadFileEvent;
```

Description

Occurs before a file will be uploaded from the local machine to the FTP server.

Parameters:

- `Sender` - the object that raised the event;
- `Source` - holds the source data stream that is being copied;
- `DestFile` - holds the destination path to copy the file to;
- `StartPos` - the offset in bytes relative to the beginning of the file that the uploading started at.

See Also

Upload

5.157.4.13 OnBanner

```
type
  TScBannerEvent = procedure(Sender: TObject; const Banner: string) of
object;

property OnBanner: TScBannerEvent;
```

Description

Occurs if FTP server returns a banner when authenticating. The `Banner` holds the received banner.

The banner may contain a warning message or any other information message.

See Also

Connect

5.157.4.14 OnError

```
type
  TScFTPErrorEvent = procedure(Sender: TObject; ErrorCode: integer; const
ErrorMessage: string; var Fail: boolean) of object;

property OnError: TScFTPErrorEvent;
```

Description

The **OnError** event occurs when the server returns an error when executing some operation.

Parameters:

- `Sender` - the object that raised the event;
- `ErrorCode` - holds the error code;
- `ErrorMessage` - holds the readable description of the error;
- `Fail` - set the `Fail` parameter to False to prevent raising an exception, and set this parameter to True to raise the EScFTPError exception.

See Also

ReplyCode

5.157.4.15 OnProgress

```
type
  TScOnProgressEvent = procedure(Sender: TObject; Total, Current: Int64;
var Cancel: boolean) of object;

property OnProgress: TScOnProgressEvent;
```

Description

Occurs when uploading and downloading files each time when the next piece of data is sent to the server or received from it.

Set `Cancel` to True if you want to abort the current operation. In this case, the client will stop data transfer and generate a corresponding exception.

See Also

Dowload

Upload

5.157.4.16 OnReadReply

```
type
  TScReadLineEvent = procedure (Sender: TObject; const Line: string) of
object;

property OnReadReply: TScReadLineEvent;
```

Description

Occurs when the client has received a response from the FTP server. This event can be used to log responses from the server.

Parameters:

- `Sender` - the object that raised the event;
- `Line` - the response message from the FTP server.

See Also

OnSendCommand

5.157.4.17 OnSendCommand

```
type
  TScSendLineEvent = procedure (Sender: TObject; const Line: string) of
object;

property OnSendCommand: TScSendLineEvent;
```

Description

Occurs when the client has sent a command to the FTP server. This event can be used to log commands sent to the server.

Parameters:

- `Sender` - the object that raised the event;
- `Line` - the FTP command sent to the server.

See Also

OnReadReply

5.158 TScMailAddressItem

5.158.1 Description

Unit

ScMailMessage

Description

TScMailAddressItem is a TCollectionItem descendant that represents an item of the address of an electronic mail sender or recipient.

The **TScMailAddressItem** class is used by the TScSMTPClient and TScMailMessage classes to store address information for email messages.

An email address consists of a User name, Host, and optionally, DisplayName. The DisplayName can contain non-ASCII characters if you encode them.

See Also

TScMailAddressList

TScMailMessage

5.158.2 Properties

5.158.2.1 Address

```
property Address: string;
```

Description

Holds an email address in the RFC 822-compliant format. When you change the value of this property, the AsString, Host, and User properties are changed automatically, and vice versa.

Example: 'support@devart.com'

See Also

AsString

DisplayName

5.158.2.2 AsString

```
property AsString: string;
```

Description

Holds the display name and email address in the RFC 822-compliant format. The value of this property depends on the values of Address, DisplayName, Host, and User.

When you change the value of **AsString**, the Address, DisplayName, Host, and User properties are changed automatically, and vice versa.

AsString uses the following fomat: 'DisplayName <User@Host>'

If a DisplayName is not set, the following format is used: 'User@Host'.

See Also

Address

DisplayName

Host

User

5.158.2.3 DisplayName

```
property DisplayName: string;
```

Description

Holds the display name of the email address. When you change the value of this property, the AsString property is changed automatically, and vice versa.

Example: 'Devart Support'.

See Also

Address

AsString

5.158.2.4 Host

```
property Host: string;
```

Description

Holds the host portion of an email address. When you change the value of this property, the Address and AsString properties are changed automatically, and vice versa.

In a typical email address, the host string comes after the @ sign — for example, in 'support@devart.com', the host is 'devart.com'.

See Also

Address

AsString

5.158.2.5 User

```
property User: string;
```

Description

Holds the user portion of an email address. When you change the value of this property, the Address and AsString properties are changed automatically, and vice versa.

In a typical email address, the user string comes before the @ sign — for example, in 'support@devart.com', the user is 'support'.

See Also

Address

AsString

5.159 TScMailAddressList

5.159.1 Description

Unit

ScMailMessage

Description

The **TScMailAddressList** class is a descendant of the TOwnedCollection class, and is a container for TScMailAddressItem objects.

TScMailAddressList allows Items to be added or maintained in a collection, and can be used to access the collection as a comma-delimited list which is used in email messages.

The TScMailMessage.ToAddress, TScMailMessage.CC, and TScMailMessage.Bcc properties return instances of this class that are used to hold respective email addresses.

See Also

TScMailAddressItem

TScMailMessage

5.159.2 Properties

5.159.2.1 AsString

```
property AsString: string;
```

Description

Holds a collection of email addresses. If a collection contains multiple addresses, they are separated by commas.

When you set this property, the collection is cleared, the specified string is parsed, and each email is added to the list as a TScMailAddressItem.

5.159.2.2 Items

```
property Items[Index: integer]: TScMailAddressItem; default;
```

Description

Lists the TScMailAddressItem object references.

Use **Items** to access objects in the list. **Items** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read the value at a specific index, or use **Items** with the Count property to iterate over the list.

Note: **Items** is the default property of TScMailAddressList, which means you can omit the property name.

5.159.3 Methods

5.159.3.1 GetDomains

```
procedure GetDomains(AStrings: TStrings);
```

Description

Fills a TString object with the domain addresses from a collection.

The method iterates over Items and adds TScMailAddressItem.Host values to `AStrings` if a value has not been added yet.

See Also

AsString

5.160 TScMimeBoundary

5.160.1 Description

Unit

ScMailMessage

Description

The **TScMimeBoundary** class represents MIME boundary markers from email headers.

TScMimeBoundary is a stack-based container for MIME boundary markers found in the "Content-Type" email header.

New boundary markers are added to the top of the stack using the Push method, and removed from the top of the marker list using the Pop method.

See Also

TScMimeBoundary.Pop

TScMimeBoundary.Push

TScMailMessage.MimeBoundary

5.160.2 Properties

5.160.2.1 Boundary

```
property Boundary: string;
```

Description

Holds the most recent MIME boundary market in the list. This property can be used to read the last boundary marker encountered during message header processing.

This is a read-only property.

5.160.3 Methods

5.160.3.1 Assign

```
procedure Assign(Source: TScMimeBoundary);
```

Description

Copies properties and other attributes of the `Source` object to the current object.

5.160.3.2 Clear

```
procedure Clear;
```

Description

Removes all boundary markers from the list accumulated by the current object. **Clear** can be called when a TScMailMessage calls the Clear method.

See Also

Count

Pop

Push

5.160.3.3 Count

```
function Count: integer;
```

Description

Determines the number of boundary markers in the list accumulated by the current object.

See Also

Clear

Pop

Push

5.160.3.4 Pop

```
procedure Pop;
```

Description

Removes the boundary marker from the top of the marker list. To remove all boundary markers from the list you use Clear.

See Also

Clear

Count

Push

5.160.3.5 Push

```
procedure Push(const Boundary: string; ParentIndex: integer);
```

Description

Adds the `Boundary` marker to the top of the list, using a stack-based mechanism. The method checks that `Boundary` does not already exist in the list, and if not, inserts it at index 0.

See Also

Clear

Count

Pop

5.161 TScMailMessage

5.161.1 Description

Unit

ScMailMessage

Description

TScMailMessage represents an email message that can be sent using the TScSMTPClient class. **TScMailMessage** is used with message-based protocols, such as POP3 and SMTP.

To send a message using the SMTP protocol you create a **TScMailMessage** object and set the sender and recipient addresses in the From, Sender, ToAddress, CC, Bcc, ReplyTo properties.

The message body is set in the Body property, and the required Internet headers are set in the Headers property. You can also set the charset in the ContentCharset property to encode Body, and transfer encoding in the ContentTransferEncoding property.

To add attached objects you use the Attachments collection.

To send a message you call the TScSMTPClient.Send method.

See Also

Body

Headers

From

Sender

ToAddress

CC

Bcc

ReplyTo

ReturnReceiptTo

TScSMTPClient

5.161.2 Properties

5.161.2.1 Attachments

```
property Attachments: TScAttachmentCollection;
```

Description

Holds a collection of objects attached to the email message object. Attached can be text, files, and other data types.

See Also

TScAttachmentCollection

TScTextAttachment

TScAttachment

5.161.2.2 Bcc

```
property Bcc: TScMailAddressList;
```

Description

Holds a list of addresses of the blind carbon copy (BCC) recipients of the email message. The message will be sent to these recipients in a separate list, and the email address of these recipients will not be displayed in the "To:" list.

To add an address to the list you create a TScMailAddressItem object, specify the needed address, and add the object to the list returned by **Bcc**.

See Also

CC

ToAddress

5.161.2.3 Body

```
property Body: TStrings;
```

Description

Holds the message body.

By default the ContentType of data in **Body** is "text/plain".

You can set the charset in the ContentCharset property to encode Body, and transfer encoding in the ContentTransferEncoding property.

You can also add a TScTextAttachment object to the Attachments list to present different versions of the message, e.g. in the HTML format.

See Also

Attachments

ContentCharset

ContentTransferEncoding

ContentType

5.161.2.4 CC

```
property CC: TScMailAddressList;
```

Description

Holds a list of addresses of the carbon copy (CC) recipients of the email message. The list contains additional recipients besides those in the ToAddress property.

To add a new address to the list you create a TScMailAddressItem object, specify the address, and add the object to the list returned by **CC**.

See Also

Bcc

ToAddress

5.161.2.5 ContentCharset

```
property ContentCharset: string;
```

Description

Holds the charset to encode or decode the message body.

ContentCharset sets the "charset" field in the Content-Type header. When you change **ContentCharset**, the ContentType property is changed as well, and vice versa.

The default character set is 'us-ascii'.

If you set **ContentCharset** to 'UTF8', 'Unicode', or 'UTF32', it is recommended that you set ContentTransferEncoding to 'Base64'.

An example of the charset field in the in the Content-Type header: 'Content-type: text/plain; charset=us-ascii; boundary="abcd"'.

Use the HeadersCharset property to encode the headers portion of the email message.

See Also

Body

ContentType

ContentTransferEncoding

HeadersCharset

5.161.2.6 ContentDisposition

```
property ContentDisposition: string;
```

Description

Holds the MIME content disposition for the email message. Additional parameters can be "creation-date", "modification-date", "read-date", and "size".

ContentDisposition gets or sets the value of the "Content-Disposition" Internet Message header.

When you change **ContentDisposition**, the Headers property is changed as well.

See Also

Headers

5.161.2.7 ContentTransferEncoding

```
property ContentTransferEncoding: string;
```

Description

Holds the transfer encoding of the message body.

Common values of **ContentTransferEncoding** include '7bit', '8bit', 'binary', 'base64', 'binhex40', 'quoted-printable', 'xxe', 'uue'. **ContentTransferEncoding** gets or sets the value of the "Content-Transfer-Encoding" Internet Message header.

When you change **ContentTransferEncoding**, the respective header in the Headers property is changed as well.

See Also

Headers

5.161.2.8 ContentType

```
property ContentType: string;
```

Description

Holds the MIME media type of the message.

ContentType is intended to describe the data contained in the message body to allow a user agent to determine how to present the data.

Common values of **ContentType** include text/plain, text/html, text/xml, text/enhanced, image/jpeg, image/gif, audio/basic, audio/au, video/mpeg, application/octet-stream, application/postscript, application/ms-word, application/ms-excel, application/rtf, multipart/mixed, multipart/alternative, and message/external-body.

ContentType gets or sets the value of the "Content-Type" Internet Message header. When you change **ContentType**, the respective header in the Headers property is changed as well.

See Also

Headers

5.161.2.9 Date

```
property Date: TDateTime;
```

Description

Holds a TDateTime value that determines the sender's local time when the message was sent.

Date gets or sets the value of the "Date" Internet Message header. When you change **Date**, the respective header in the Headers property is changed as well.

If **Date** property is not set, the current local time will be used when the message is sent.

See Also

Headers

5.161.2.10 Encoding

```
property Encoding: TScMailEncoding;
```

Description

Holds the encoding scheme of the message Attachments.

The default value is `meDefault`. With the default value, the encoding scheme will be chosen automatically when the message is sent. If the message has no attachments, the `mePlainText` encoding is used.

If the message has attachments, the `meMIME` encoding is used.

See Also

Attachments

5.161.2.11 From

```
property From: TScMailAddressItem;
```

Description

Holds the From address of the email message sender.

The value of this property may be different from Sender when a message is forwarded.

See Also

Sender

ToAddress

5.161.2.11 GeneratedHeaders

```
property GeneratedHeaders: TScHeaderList;
```

Description

Holds a complete list of headers sent with an email message.

When you send or receive an email message, a TScHeaderList is created. The following data is then added to the newly created object: all Headers, all email addresses of senders and recipients in From, Sender, CC, ToAddress, ReplyTo, followed by the required headers such as Content-Type, Content-Transfer-Encoding, Date, Subject, X-Priority, and finally all SpecialHeaders.

This property is read-only.

See Also

Headers

SpecialHeaders

From

Sender

CC

ToAddress

ReplyTo

5.161.2.12 Headers

```
property Headers: TScHeaderList;
```

Description

Holds the email headers that are transmitted with this email message.

You specify the sender, recipient, subject, and body of an email message in the properties of a TScMailMessage object.

To get the full list of headers that will be sent with the message you use the GeneratedHeaders property.

Once a message is received, the data will be parsed and added to the **Headers** list.

To specify the encoding of **Headers** you use the HeadersCharset and HeadersTransferEncoding properties.

See Also

HeadersCharset

HeadersTransferEncoding

GeneratedHeaders

SpecialHeaders

5.161.2.14 HeadersCharset

```
property HeadersCharset: string;
```

Description

Holds the charset to encode custom headers for this email message.

An SMTP message consists of headers and a body. **HeadersCharset** is used to encode Headers and SpecialHeaders.

Use the ContentCharset property to specify the encoding for the body.

See Also

ContentCharset

Headers

HeadersTransferEncoding

SpecialHeaders

5.161.2.15 HeadersTransferEncoding

```
property HeadersTransferEncoding: char;
```

Description

Holds the transfer encoding for the following headers of this email message: From, To, Cc, Bcc, Reply-To, and Return-Receipt-To.

This property can be set to any of the following:

'8' - 8-bit ASCII encoding;

'B' - Base64 encoding;

'Q' - quoted-printable encoding.

See Also

Headers

5.161.2.16 InReplyTo

```
property InReplyTo: string;
```

Description

Holds the value of the "In-Reply-To" Internet Message header.

When you change this property, the respective header in the Headers property is changed as well.

See Also

Headers

5.161.2.17 MessageId

```
property MessageId: string;
```

Description

Holds the unique message identifier for the email message. Message-ID can be assigned by the server that receives this message.

MessageId gets of sets the value of the "Message-Id" Internet Message header. When you change **MessageId**, the respective header in the Headers property is changed as well.

See Also

Headers

5.161.2.18 MimeBoundary

```
property MimeBoundary: TScMimeBoundary;
```

Description

Holds the container for boundary marker strings used in this message.

The property is updated when headers are processed from an incoming message, or when message parts are written to the output stream by the message client.

See Also

TScMimeBoundary

5.161.2.19 Organization

```
property Organization: string;
```

Description

Holds the identity of the organization or institution to which the sender of this message belongs.

Organization gets or sets the value of the "Organization" Internet Message header. When you change **Organization**, the respective header in the Headers property is changed as well.

See Also

Headers

5.161.2.20 Priority

```
property Priority: TScMailPriority;
```

Description

Determines the priority of this email message.

When you change this property, the 'Importance', 'Priority', and 'X-Priority' headers in the Headers property are changed as well.

See Also

Headers

5.161.2.21 References

```
property References: string;
```

Description

Holds a list of message identifiers for which the email message is a reply.

This property gets or sets the value of the "References" Internet Message header. When you change this property, the respective header in the Headers property is changed as well.

See Also

Headers

5.161.2.22 ReplyTo

```
property ReplyTo: TScMailAddressList;
```

Description

Holds a list of email addresses that will receive a reply to this email message. The list contains additional recipients besides those in the From property.

To add a new address to the list you create a TScMailAddressItem object, specify the address, and add the object to the list returned by **ReplyTo**.

See Also

From

5.161.2.23 ReturnReceiptTo

```
property ReturnReceiptTo: TScMailAddressItem;
```

Description

Holds the email address where a return receipt for the email message will be delivered.

This property gets or sets the values of the "Return-Receipt-To" and "Disposition-Notification-To" Internet Message headers.

See Also

ReplyTo

5.161.2.24 Sender

```
property Sender: TScMailAddressItem;
```

Description

Holds the sender's address for this email message, if the person is not the message author.

This property gets or sets the value of the "Sender" Internet Message header.

To get the original author of this message you use the From property.

See Also

From

5.161.2.25 SpecialHeaders

```
property SpecialHeaders: TScHeaderList;
```

Description

Holds an additional list of headers to be transmitted with the email message. These headers are not handled in any way, but are appended to the full list of headers when the message is sent.

The user is responsible for making sure that the headers in this list do not duplicate or contradict Headers.

To get a full list of headers that will be sent with the message you use the GeneratedHeaders property.

See Also

Headers

GeneratedHeaders

5.161.2.26 Subject

```
property Subject: string;
```

Description

Holds the subject line of the email message.

Subject gets or sets the value of the "Subject" Internet Message header. When you change **Subject**, the respective header in the Headers property is changed as well.

To specify encoding of "Subject" you use the SubjectCharset and SubjectTransferEncoding property.

See Also

Headers

SubjectCharset

SubjectTransferEncoding

5.161.2.27 SubjectCharset

```
property SubjectCharset: string;
```

Description

Holds the charset to encode the Subject header of the email message.

Use the ContentCharset property to specify the encoding of the body the email message.

Use the HeadersCharset property to specify the encoding of the headers (except for Subject) of the email message.

See Also

ContentCharset

HeadersCharset

Subject
SubjectTransferEncoding

5.161.2.28 SubjectTransferEncoding

```
property SubjectTransferEncoding: char;
```

Description

Holds the transfer encoding for the Subject and Organization headers of the email message.

This property can be set to any of the following:

'8' - 8-bit ASCII encoding;

'B' - Base64 encoding;

'Q' - quoted-printable encoding.

See Also

Organization
Subject
SubjectCharset

5.161.2.29 ToAddress

```
property ToAddress: TScMailAddressList;
```

Description

Holds a list of recipients of the email message. You can specify additional recipients in the CC property.

To add a new address to the list you create a TScMailAddressItem object, specify the address, and add the object to the list returned by **ToAddress**.

See Also

Bcc
CC
From
Sender

5.161.3 Methods

5.161.3.1 Clear

```
procedure Clear;
```

Description

Clears the values from the the Headers and Body of the message, and also all recipients and senders from Sender, From, ToAddress, CC, Bcc, ReplyTo, and ReturnReceiptTo.

See Also

Body

Headers

5.162 TScCustomAttachment

5.162.1 Description

Unit

ScSMTPUtils

Description

TScCustomAttachment is a TCollectionItem descendant that represents an email attachment.

TScCustomAttachment is the base class for common attachment types such as TScAttachment and TScTextAttachment.

The **TScCustomAttachment** class is used by the TScMailMessage class to store attachments information for the message.

TScCustomAttachment is the item added to the TScAttachmentCollection collection.

See Also

TScMailMessage.Attachments

TScAttachment

TScTextAttachment

TScAttachmentCollection

5.162.2 Properties

5.162.2.1 ContentCharset

```
property ContentCharset: string;
```

Description

Holds the charset to encode or decode the email attachment.

This property sets the "charset" field in the Content-Type header. When you change this property, the ContentType property is changed as well, and vice versa.

See Also

ContentType

ContentTransferEncoding

5.162.2.2 ContentDescription

```
property ContentDescription: string;
```

Description

Holds a description of the attachment.

This property holds the value of the "'Content-Description" Internet Message header.

5.162.2.3 ContentDisposition

```
property ContentDisposition: string;
```

Description

Holds the MIME content disposition of the attachment.

ContentDisposition holds the value of the "Content-Disposition" Internet Message header.

ContentDisposition property can contain the parameter "attachment" to indicate that the attachment is separate from the main body of the message.

The parameter "filename" can be used to suggest a filename where the attachment is detached and stored in a separate file. When you change the "filename" parameter, the FileName property is changed as well.

See Also

FileName

5.162.2.4 ContentID

```
property ContentID: string;
```

Description

Holds the MIME content ID of this attachment.

This property holds the value of the "Content-ID" Internet Message header.

5.162.2.5 ContentName

```
property ContentName: string;
```

Description

Holds the name value of the MIME content type from the Content-Type header associated with the attachment.

When you change **ContentName**, the ContentType property is changed as well, and vice versa.

ContentName takes the attachment name when an email with an attachment is received.

See Also

ContentType

5.162.2.6 ContentTransferEncoding

```
property ContentTransferEncoding: string;
```

Description

Holds the transfer encoding of the attachment.

Common values of **ContentTransferEncoding** include the following: '7bit', '8bit', 'binary', 'base64', 'binhex40', 'quoted-printable', 'xxe', 'uue'.

ContentTransferEncoding holds the value of the "Content-Transfer-Encoding" Internet Message header.

See Also

ContentCharset

5.162.2.7 ContentType

```
property ContentType: string;
```

Description

Holds the MIME media type of the attachment

Common values of **ContentType** include text/plain, text/html, text/xml, text/enhanced, image/jpeg, image/gif, audio/basic, audio/au, video/mpeg, application/octet-stream, application/postscript, application/ms-word, application/ms-excel, application/rtf, multipart/mixed, multipart/alternative, message/external-body.

ContentType holds the value of the "Content-Type" Internet Message header.

The "charset" parameter can be used to set the charset for the attachment. When you change the "charset" parameter, the ContentCharset property is changed as well.

The "name" parameter can be used to set the MIME content type name. When you change the "name" parameter, the ContentName property is changed as well.

See Also

ContentCharset

ContentName

5.162.2.8 FileName

```
property FileName: string;
```

Description

Holds the filename to suggest the receiver where the attachment is detached and stored in a separate file.

This property sets the "filename" field in the Content-Dispositon header. When you change the parameter, the ContentDisposition property is changed as well, and vice versa.

An example of the filename field in the Content-Dispositon header: 'Content-Dispositon: attachment; filename="test"'.

See Also

ContentDisposition

5.162.2.9 SpecialHeaders

```
property SpecialHeaders: TScHeaderList;
```

Description

Holds an additional list of headers to be transmitted with the attachment. These headers are not handled in any way, but are appended to the full list of headers when the message is sent.

The user is responsible for making sure that the headers in this list do not duplicate or contradict Headers.

5.163 TScAttachment

5.163.1 Description

Unit

ScSMTPUtils

Description

TScAttachment is a TScCustomAttachment descendant that represents a MIME-encoded attachment.

The **TScAttachment** and **TScTextAttachment** classes are used by the TScMailMessage class to store attachments information for the message.

TScAttachment provides methods and properties for storing the attachment content in a file system or local memory.

To store the attachment content in a file you call the respective constructor by passing the filename or by seting the filename later in the PathName property.

To store the attachment content in local memory you call the constructor with the TStream parameter or set the PathName property later to an empty string.

You call the OpenStream method to read/write the attachment content; the method returns a stream containing the attachment content. When the read/write operation is complete you call the CloseStream method.

To load the attachment content from a file or stream you use the LoadFromFile or LoadFromStream method, respectively.

See Also

TScMailMessage.Attachments

TScCustomAttachment

TScTextAttachment

TScAttachmentCollection

5.163.2 Properties

5.163.2.1 PathName

```
property PathName: string;
```

Description

Holds the name of the file that contains the attachment content. If you set this property to an empty string, the attachment content will be stored in local memory.

When you load data from external objects using the LoadFromFile or LoadFromStream methods, data is saved to a file specified in the property.

You should not open the file yourself, but rather use the OpenStream method.

See Also

LoadFromStream

LoadFromFile

OpenStream

5.163.2.2 TempDirectory

```
property TempDirectory: string;
```

Description

Holds the name of a directory where temporary files containing the attachment content, will be created.

This directory is only used when the Create constructor is invoked by passing an empty string. In this case a random filename is created.

See Also

Create

PathName

5.163.3 Methods

5.163.3.1 CloseStream

```
procedure CloseStream;
```

Description

Specifies that the stream that contains the attachment content returned by the OpenStream method can be closed.

Always call this method when you have finished working with an object returned by the OpenStream method.

See Also

OpenStream

PathName

5.163.3.2 Create

```
constructor Create(Collection: TScAttachmentCollection; const FileName: string); reintroduce; overload;
constructor Create(Collection: TScAttachmentCollection; CopyFrom: TStream); reintroduce; overload;
```

Description

Creates a new TScAttachment object and adds it to the TScAttachmentCollection collection specified by the `Collection` parameter.

The `FileName` parameter indicates the name of the file which attachment content is to be loaded. The value of this of parameter will be also set in the PathName property. When `FileName` is an empty string, the attachment content will be empty, and you will need to write or load the attachment content using the OpenStream, LoadFromFile, or LoadFromStream. In this case the attachment content will be stored in a file with a random name in TempDirectory.

The `CopyFrom` parameter accepts a `TStream` object to load the attachment content from.

See Also

Init

LoadFromFile

LoadFromStream

PathName

OpenStream

TempDirectory

5.163.3.3 Init

```
procedure Init(const Str: string);
```

Description

Initializes the attachment content using a string specified in the `Str` parameter. If an object has some data, it will be overwritten.

See Also

LoadFromFile

LoadFromStream

PathName

SaveToFile

SaveToStream

5.163.3.4 LoadFromFile

```
procedure LoadFromFile(const FileName: string);
```

Description

Loads the attachment content from a file specified in the `FileName` parameter.

When the PathName property is set, the data is copied to the file specified in the PathName property. Otherwise, the data is loaded to a TMemoryStream object that stores the attachment content.

See Also

LoadFromStream

PathName

SaveToFile

SaveToStream

5.163.3.5 LoadFromStream

```
procedure LoadFromStream(AStream: TStream);
```

Description

Loads the attachment content from a stream specified in the `AStream` parameter.

When the PathName property is set, the data from the stream is copied to the file specified in the PathName property. Otherwise, the data is loaded to a TMemoryStream object that stores the attachment content.

See Also

LoadFromFile

PathName

SaveToFile

SaveToStream

5.163.3.6 OpenStream

```
function OpenStream(ForWriting: boolean = False): TStream;
```

Description

Returns a stream containing the attachment content.

When the PathName property is set, the method opens the specified file and grants access to it. Otherwise, a pointer to the TMemoryStream object containing the attachment content is returned.

The `ForWriting` parameter indicates whether write access to the attachment content is required, or the stream is only used for reading data.

You should not free the object returned by this method.

Call the CloseStream method when you have finished reading/writing data.

See Also

CloseStream

PathName

5.163.3.7 SaveToFile

```
procedure SaveToFile(const FileName: string);
```

Description

Saves the attachment content to the file specified in the `FileName` parameter.

See Also

LoadFromFile

LoadFromStream

PathName

SaveToStream

5.163.3.8 SaveToStream

```
procedure SaveToStream(AStream: TStream);
```

Description

Saves the attachment content to the TStream object specified in the `AStream` parameter.

See Also

LoadFromFile

LoadFromStream

PathName

SaveToFile

5.163.3.9 ToString

```
function ToString: string;
```

Description

Represents the email attachment as a string.

5.164 TScTextAttachment

5.164.1 Description

Unit

ScSMTPUtils

Description

TScTextAttachment is a TScCustomAttachment descendant that represents a MIME-encoded textual attachment.

The **TScTextAttachment** and **TScAttachment** classes are used by the TScMailMessage class to store attachments information for the message.

TScTextAttachment provides the Body property that represents the textual content of the attachment.

See Also

TScMailMessage.Attachments

TScAttachment

TScCustomAttachment

TScAttachmentCollection

5.164.2 Properties

5.164.2.1 Body

```
property Body: TStrings;
```

Description

Holds the plain text of the MIME attachment.

You can use the methods and properties of the `TStrings` class, such as Add, Clear, Remove, Text, etc. to work with the list of strings.

See Also

Create

5.164.3 Methods

5.164.3.1 Create

```
constructor Create(Collection: TScAttachmentCollection; Body: TStrings =
nil); reintroduce;
```

Description

Creates a new TScTextAttachment object and adds it to a TScAttachmentCollection specified by the `Collection` parameter.

The Body parameter accepts a `TStrings` object that contains the attachment content. The content will be assigned to the Body property.

See Also

Body

5.165 TScAttachmentCollection

5.165.1 Description

Unit

ScSMTPUtils

Description

The **TScAttachmentCollection** class is a descendant of the TOwnedCollection class, and is a container for ancestors of TScCustomAttachment such as TScAttachment and TScTextAttachment.

TScAttachmentCollection allows Items to be added or maintained in the collection, and can be used to access a list of email attachments.

The TScMailMessage.Attachments property returns instances of this class that are used to hold the collection of objects attached to the email message.

See Also

TScAttachment

TScCustomAttachment

TScTextAttachment

TScMailMessage.Attachments

5.165.2 Properties

5.165.2.1 Encoding

```
property Encoding: string;
```

Description

Holds the default transfer encoding for attachments in the collection.

Common values of **Encoding** include the following: 'base64', 'xxe', 'uue'.

The value of **Encoding** is set in the TScCustomAttachment.ContentTransferEncoding property for all the collection attachments which ContentTransferEncoding is not specified.

See Also

Items

TScCustomAttachment.ContentTransferEncoding

5.165.2.2 Items

```
property Items[Index: integer]: TScCustomAttachment; default;
```

Description

Lists the TScCustomAttachment object references.

Use **Items** to access objects in the list. **Items** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read the value at a specific index, or use **Items** with the Count property to iterate over the list.

Note: **Items** is the default property of TScAttachmentCollection. This means you can omit the property name.

5.166 TScHeaderList

5.166.1 Description

Unit

ScSMTPUtils

Description

TScHeaderList holds a list of key-value pairs that can be represented in the HTTP header format.

You use **TScHeaderList** to store and maintain a list of key-value pairs. **TScHeaderList** provides properties and methods to add, delete, locate, and access items.

To get the entire list of values in the HTTP header format you use the ToString method.

5.166.2 Properties

5.166.2.1 Strings

```
property Strings[Index: Integer]: string;
```

Description

Obtains an item from the list in the HTTP header format: `'Key: Value'`.

The `Index` parameter indicates the index of the key, where 0 is the index of the first item, 1 is the index of the second item, and so on.

You can read the header at a specific index, or use **Strings** with the Count property to iterate over the list.

This property is read-only.

See also

Count

Keys

Values

5.166.3 Methods

5.166.3.1 ToString

```
function ToString: string;
```

Description

Represents the list of key-value pairs as a string in the HTTP header format:

```
'Header1: Value1#13#10Header2: Value2#13#10Header3: Value3'
```

See also

Strings

5.167 TScSASLMechanism

5.167.1 Description

Unit

ScSMTPClient

Description

The **TScSASLMechanism** class implements the SASL mechanism (Simple Authentication and Security Layer) for authentication in various protocols.

The SASL mechanism is supported by the SMTP, POP3, IMAP, and other application protocols.

TScSASLMechanism is an abstract base class that provides methods for authentication. The methods should be overridden in descendant classes.

The SASL authentication flow is as follows.

Some protocols support the Initial Client Response mechanism, which allows to start the authentication process after receiving initialized data from server, without first sending the name of a SASL mechanism. For such protocols, the TryStartAuthenticate method is invoked, and the result is returned to the SASL server.

If a protocol does not support Initial Client Response, an authentication request message containing the name of a SASL mechanism is sent to the server, and if the server supports the mechanism, the StartAuthenticate method is called, which handles the challenge received from the SASL server and returns a response that will be sent to the server.

The server returns a response which is passed to the ContinueAuthenticate method that handles it and returns a response that will be sent to the server.

ContinueAuthenticate can be called in response to each server request until the server confirms successful authorization.

See also

TScSASLAnonymous
TScSASLCRAMMD5
TScSASLCRAMSHA1
TScSASLLogin
TScSASLOTP
TScSASLPlain
TScSASLSKey
TScSASLUserPassMechanism

5.167.2 Properties

5.167.2.1 SecurityLevel

```
property SecurityLevel: cardinal;
```

Description

Holds the assumed security level of a SASL mechanism.

0 means no security. A higher value indicates more security. The highest value is $FFFFFFFF.

This value is advisory only, and the developer may set a different value is a descendant class.

This property is read-only.

See also

ServiceName

5.167.3 Methods

5.167.3.1 ContinueAuthenticate

```
function ContinueAuthenticate(const Response, Host, Protocol: string):
string; virtual;
```

Description

Handles the response received from the SASL server during authentication and returns a response that will be sent to the server.

ContinueAuthenticate can be called in response to each server request until the server confirms successful authorization.

Parameters:

- `Response` - the response received from the server during SASL authentication.
- `Host` - the hostname of the server where the authentication process runs.
- `Protocol` - the name of the application procotol that uses SASL, e.g. 'smtp'.

See also

StartAuthenticate

TryStartAuthenticate

5.167.3.2 IsReadyToStart

```
function IsReadyToStart: boolean; virtual;
```

Description

Checks if authentication is ready to start.

IsReadyToStart returns True when an object is initialized and necessary properties are set. Otherwise, returns False.

See also

ServiceName

StartAuthenticate

5.167.3.3 ServiceName

```
class function ServiceName: string; virtual;
```

Description

ServiceName is class method that returns the name of a SASL mechanism in a string that the SASL server can understand.

The string format is defined by the SASL specification.

See also

StartAuthenticate

5.167.3.4 StartAuthenticate

```
function StartAuthenticate(const Challenge, Host, Protocol: string):
string; virtual; abstract;
```

Description

Handles the challenge received from the server in response to the SASL authentication request and returns a response that will be sent to the server.

When the authentication process is started, the SASL server requests initial data for authentication, and **StartAuthenticate** returns a response to this request.

Parameters:

- `Challenge` - the challenge received from the server in response to the SASL authentication request.
- `Host` - the hostname of the server where the authentication process runs.
- `Protocol` - the name of the application procotol that uses SASL, e.g. 'smtp'.

StartAuthenticate is an abstract method and should be overridden in descendant classes that implement a specific SASL mechanism.

See also

ContinueAuthenticate

IsReadyToStart

TryStartAuthenticate

5.167.3.5 TryStartAuthenticate

```
function TryStartAuthenticate(const Host, Protocol: string; out
InitialResponse: string): boolean; virtual;
```

Description

Returns an initial response to the server.

Some protocols support the Initial Client Response mechanism, which allows to start the authentication process after receiving initialized data from server, without first sending the name of a SASL mechanism. This allows reducing the number of round-trips.

The method returns True if a given mechanism supports the Initial Client Response mechanism; otherwise, False.

Parameters:

- `Host` - the hostname of the server where the authentication process runs.
- `Protocol` - the name of the application procotol that uses SASL, e.g. 'smtp'.
- `InitialResponse` - the initial response to the server when an authentication request is received.

See also

ContinueAuthenticate

StartAuthenticate

5.168 TScSASLAnonymous

5.168.1 Description

Unit

ScSMTPClient

Description

TScSASLAnonymous is a TScSASLMechanism descendant that implements the Anonymous SASL mechanism.

This mechanism allows the user to gain anonymous access to services or resources.

The initialization string to be sent to the server in response to the authentication request can be specified in the TraceInfo property.

The ServiceName method of this class returns the value 'ANONYMOUS'.

See also

TScSASLMechanism

5.168.2 Properties

5.168.2.1 TraceInfo

```
property TraceInfo: string;
```

Description

Holds the initialization string that is sent to the server when SASL authentication starts.

5.169 TScSASLUserPassMechanism

5.169.1 Description

Unit

ScSMTPClient

Description

TScSASLUserPassMechanism is a TScSASLMechanism descendant and a base class for

mechanisms that require a username and password for authentication.

Mechanism implementations should be overridden in descendant classes.

A username and password can be specified in the appropriate properties, Username and Password.

See also

TScSASLMechanism
TScSASLCRAMMD5
TScSASLCRAMSHA1
TScSASLLogin
TScSASLOTP
TScSASLPlain
TScSASLSKey

5.169.2 Properties

5.169.2.1 Password

```
property Password: string;
```

Description

Holds the password for authentication with a username and password.

The username is specified in the Username property.

See also

Username

5.169.2.2 Username

```
property Username: string;
```

Description

Holds the username for authentication with a username and password.

The password for user authentication is specified in the Password property.

See also

Password

5.170 TScSASLPlain

5.170.1 Description

Unit

ScSMTPClient

Description

TScSASLPlain is a TScSASLUserPassMechanism descendant that implements the PLAIN SASL mechanism.

PLAIN is a simple clear-text SASL authentication mechanism. PLAIN replaces the obsolete LOGIN mechanism.

The username and password can be specified in the appropriate properties, Username and Password.

The ServiceName method of this class returns the 'PLAIN' value.

See also

TScSASLLogin

TScSASLMechanism

TScSASLUserPassMechanism

5.170.2 Properties

5.170.2.1 Login

```
property Login: string;
```

Description

Holds the authentication identity when logging in with a username.

The password for user authentication is specified in the Password property.

See also

Password

5.171 TScSASLLogin

5.171.1 Description

Unit

ScSMTPClient

Description

TScSASLLogin is a TScSASLUserPassMechanism descendant that implements the LOGIN SASL mechanism.

LOGIN is a simple clear-text user/password authentication mechanism. LOGIN is an obsolete mechanism, and you should use the PLAIN mechanism implemented in the TScSASLPlain class instead.

The username and password can be specified in the appropriate properties, Username and Password.

The ServiceName method of this class returns the 'LOGIN' value.

See also

TScSASLPlain

TScSASLMechanism

TScSASLUserPassMechanism

5.172 TScSASLOTP

5.172.1 Description

Unit

ScSMTPClient

Description

TScSASLOTP is a TScSASLUserPassMechanism descendant that implements the OTP SASL mechanism.

OTP is a one-time password mechanism. OTP replaces the obsolete SKEY mechanism.

The username and password can be specified in the appropriate properties, Username and Password.

The ServiceName method of this class returns the 'OTP' value.

See also

TScSASLMechanism

TScSASLUserPassMechanism

5.173 TScSASLSKey

5.173.1 Description

Unit

ScSMTPClient

Description

TScSASLSKey is a TScSASLUserPassMechanism descendant that implements the SKEY SASL mechanism.

SKEY is a one-time password mechanism. SKEY is an obsolete mechanism, and you should use the OTP mechanism implemented in the TScSASLOTP class instead.

The username and password can be specified in the appropriate properties, Username and Password.

The ServiceName method of this class returns the 'SKEY' value.

See also

TScSASLMechanism

TScSASLOTP

TScSASLUserPassMechanism

5.174 TScSASLCRAMMD5

5.174.1 Description

Unit

ScSMTPClient

Description

TScSASLCRAMMD5 is a TScSASLUserPassMechanism descendant that implements the CRAM-MD5 SASL mechanism.

CRAM-MD5 is challenge-response authentication mechanism (CRAM) that is based on hashing using the HMAC-MD5 algorithm. CRAM-MD5 is an obsolete mechanism, and you should use the CRAM-SHA1 mechanism implemented in the TScSASLCRAMSHA1 class instead.

The username and password can be specified in the appropriate properties, Username and Password.

The ServiceName method of this class returns the 'CRAM-MD5' value.

See also

TScSASLCRAMSHA1

TScSASLMechanism

TScSASLUserPassMechanism

5.175 TScSASLCRAMSHA1

5.175.1 Description

Unit

ScSMTPClient

Description

TScSASLCRAMSHA1 is a TScSASLUserPassMechanism descendant that implements the CRAM-SHA1 SASL mechanism.

CRAM-SHA1 is challenge-response authentication mechanism (CRAM) that is based on hashing using the CRAM-SHA1 algorithm.

The username and password can be specified in the appropriate properties, Username and Password.

The ServiceName method of this class returns the 'CRAM-SHA1' value.

See also

TScSASLMechanism

TScSASLUserPassMechanism

5.176 TScSASLItem

5.176.1 Description

Unit

ScSMTPClient

Description

TScSASLItem is a TCollectionItem descendant that represents an item of a SASL mechanism.

The **TScSASLItem** class is used by the TScSMTPClient class to store information about the available SASL mechanisms.

TScSASLItem holds a descendant of the TScSASLMechanism class in the SASLMechanism property. The descendant class implements a given mechanism and stores information for authentication.

See Also

TScSASLCollection

TScSASLMechanism

TScSMTPClient

5.176.2 Properties

5.176.2.1 SASLMechanism

```
property SASLMechanism: TScSASLMechanism;
```

Description

Holds an object of the class that inherits from TScSASLMechanism. The descendant class implements a given mechanism and stores information for authentication.

See Also

Init

5.176.3 Methods

5.176.3.1 Init

```
procedure Init(const ServiceName: string);
```

Description

Sets a mechanism specified in the `ServiceName` parameter for a given item.

This method removes the existing object from the SASLMechanism property, creates an object of the desired class which implements the SASL mechanism specified in the `ServiceName` parameter, and assigns the newly created object to SASLMechanism.

`ServiceName` supports the following values: 'ANONYMOUS', 'PLAIN', 'LOGIN', 'OTP', 'SKEY', 'CRAM-MD5', and 'CRAM-SHA1'. If any other value is specified, the LOGIN mechanism will be used.

See Also

SASLMechanism

TScSASLAnonymous

TScSASLPlain

TScSASLLogin

TScSASLOTP

TScSASLSKey

TScSASLCRAMMD5

TScSASLCRAMSHA1

5.177 TScSASLCollection

5.177.1 Description

Unit

ScSMTPClient

Description

The **TScSASLCollection** class is a descendant of the TOwnedCollection class, and is a container for TScSASLItem objects.

TScSASLCollection allows Items to be added to or maintained in the collection, and can be used to access the list.

TScSASLCollection iterates over Items and checks if a given mechanism is ready to start the authentication process. If a mechanism is ready, **TScSASLCollection** makes a login attempt using this mechanism.

The **TScSASLCollection** class is used by the TScSMTPClient class to store information about the available SASL mechanisms.

See Also

TScSASLItem

TScSMTPClient.SASLMechanisms

5.177.2 Properties

5.177.2.1 Items

```
property Items[Index: integer]: TScSASLItem;
```

Description

Lists the TScSASLItem object references.

Use **Items** to access objects in the list. **Items** is a zero-based array: the first object is assigned the index 0, the second object is assigned the index 1, and so on. You can read the value at a specific index, or use **Items** with the Count property to iterate over the list.

Note: **Items** is the default property of TScSASLCollection. This means you can omit the property name.

See Also

TScSASLItem

5.177.3 Methods

5.177.3.1 Login

```
procedure Login(const Command, Host, Protocol: string;
  const OkReplies, ContinueReplies: array of integer; Client:
TScSMTPClient;
  Capabilities: TStrings; const AuthType: string = 'AUTH';
  IsInitialResponseSupport: boolean = True);
```

Description

Authenticates the client to the SASL server.

This method iterates over Items and checks if a given mechanism is ready to start the authentication process. If a mechanism is ready, a login attempt is made using this mechanism.

See Also

Items

5.178 TScSMTPClientOptions

5.178.1 Description

Unit

ScSMTPClient

Description

The **TScSMTPClientOptions** class determines behaviour of an SMTP client.

See also

TScSMTPClient.Options

5.178.2 Properties

5.178.2.1 BindAddress

```
property BindAddress: string;
```

Description

BindAddress determines the TCP/IP address on the local machine as the source address of the

connection. Only useful on systems with more than one TCP/IP address.

5.178.2.2 HeloName

```
property HeloName: string;
```

Description

Holds the identity name of the client that establishes a connection to the SMTP server using the EHLO or HELO command.

HeloName is used in TScSMTPClient.Connect to access the SMTP server.

When **HeloName** contains a blank value, the local hostname is used when connecting to the SMTP server.

See also

UseEhlo

5.178.2.3 IPVersion

```
property IPVersion: TIPVersion; default ivIPv4;
```

Description

Use the **IPVersion** property to specify the Internet Protocol version.

The default value is ivIPv4.

See Also

TIPVersion

5.178.2.4 MailAgent

```
property MailAgent: string;
```

Description

Holds the name of an application that is used to create or transmit a message.

MailAgent is placed in the SMTP 'X-Mailer' header in the TScSMTPClient.Send method.

5.178.2.5 Pipelined

```
property Pipelined: boolean; default False;
```

Description

Determines whether command pipelining will be used.

When this property is True and the server supports this mode, the Command Pipelining extension will be used. By default the regular mode is used.

The Command Pipelining mode allows the SMTP client to transmit groups of SMTP commands in batches without waiting for a response to each individual command.

The default value is False.

5.178.2.6 SocketReceiveBufferSize

```
property SocketReceiveBufferSize: integer; default 32768;
```

Description

Use the **SocketReceiveBufferSize** property to determine the total per-socket buffer space reserved for receives. This value is set by the OS functions to the socket.

Use this property to increase the application performance.

The default value is 32768.

See also

SocketSendBufferSize

5.178.2.7 SocketSendBufferSize

```
property SocketSendBufferSize: integer; default 32768;
```

Description

Use the **SocketSendBufferSize** property to determine the total per-socket buffer space reserved for sends.This value is set by the OS functions to the socket.

Use this property to increase the application performance.

The default value is 32768.

See also

SocketReceiveBufferSize

5.178.2.8 TCPKeepAlive

```
property TCPKeepAlive: boolean; default True;
```

Description

The **TCPKeepAlive** property specifies whether the system should send TCP keep alive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed.

The default value is True.

5.178.2.9 UseEhlo

```
property UseEhlo: boolean; default True;
```

Description

Determines whether the client will connect to the server with the SMTP EHLO greeting that instructs the server to send the supported authentication types after the banner greeting.

This property is used in the TScSMTPClient.Connect method prior to sending the HeloName to the SMTP server. When True, the client identification is sent using the EHLO command. Authentication types supported by the server are captured upon successful completion of the EHLO command. When False, the HELO command is used for client identification.

The default value is True.

See also

HeloName

5.178.2.10 UseVerp

```
property UseVerp: boolean; default False;
```

Description

Determines whether the VERP command is issued when sending a message.

VERP is used to enable automatic detection and removal of undeliverable mail recipients. When True, the VERP command is sent with a list of VerpDelimiters.

The default value is False.

See also

VerpDelimiters

5.178.2.11 VerpDelimiters

```
property VerpDelimiters: string;
```

Description

Holds a list of delimiters that is sent as the parameter of the VERP command.

The UseVerp property determines whether the VERP command will be sent.

The VERP command is used to enable automatic detection and removal of undeliverable mail recipients.

See also

UseVerp

5.179 TScSMTPClient

5.179.1 Description

Unit

ScSMTPClient

Description

The **TScSMTPClient** component implements functionality of the SMTP/SMTPS client.

The SMTP protocol is used to send email messages to recipients through the SMTP server.

The SMTP client supports data transfer over a secure TLS/SSL channel (the SMTPS protocol). You use the TLSMode property to manage security.

To send a message using the SMTP protocol you create a TScMailMessage object and set the sender and recipient addresses in the From, Sender, ToAddress, CC, Bcc, ReplyTo properties.

The message body is set in the Body property, and the required Internet headers are set in the Headers property. You can also set the charset in the ContentCharset property to encode Body, and transfer encoding in the ContentTransferEncoding property. To add attached objects you use the Attachments collection.

To send a message you call the TScSMTPClient.Send method. You can also set the subject, body, sender, and recipient directly as overloaded parameters in the TScSMTPClient.Send method.

The **TScSMTPClient** class throws an EScSMTPError exception when an error occurs during execution of any command on the SMTP server.

See Also

Active

IsSecure

Send

TScMailMessage

5.179.2 Properties

5.179.2.1 Active

```
property Active: boolean;
```

Description

Determines whether the connection to an SMTP server is established.

The property is set to True after a successful connection to the server and to False after a force or unexpected disconnection.

This property is read-only.

See Also

Connect

Disconnect

5.179.2.2 AuthenticationType

```
property AuthenticationType: TScSMTPAuthenticationType; default
satDefault;
```

Description

Determines the authentication type to use when accessing the e-mail server.

To use simple password authentication you set **AuthenticationType** to `satDefault`.

To use SASL authentication you set **AuthenticationType** to `satSASLMechanism`.

The list of used SASL mechanisms is set in the SASLMechanisms property.

See Also

Authenticate

SASLMechanisms

5.179.2.3 CEncoding

```
property CEncoding: Encoding;
```

Description

Holds the `Encoding` object that is used to encode commands sent to the SMTP server and decode responses received from the server.

By default the system locale is used.

See Also

UseUTF8

5.179.2.4 FormattedReply

```
property FormattedReply: TStringList;
```

Description

The **FormattedReply** property represents the formatted response message received during execution of the latest command request to the SMTP server.

To obtain the response code, you can use the ReplyCode ptoperty.

This property is read-only.

See Also

OnError

ReplyCode

RaiseReplyError

5.179.2.5 HostName

```
property HostName: string;
```

Description

Specifies the host name or the IP address to connect to the SMTP server.

The Connect method uses values in the **HostName** and Port properties to establish a connection for the SMTP session.

See Also

Connect

Port

5.179.2.6 IsSecure

```
property IsSecure: boolean;
```

Description

Determines whether the connection to SMTP server is protected.

When **IsSecure** is set to False, data is transferred as plaintext.

When **IsSecure** is set to True, the TLS/SSL protocol is used, and data is transferred in an encrypted form.

This property is read-only.

See also

SSLOptions

5.179.2.7 Options

```
property Options: TScSMTPClientOptions;
```

Description

Options determines behaviour of an SMTP client.

See Also

Connect

5.179.2.8 Password

```
property Password: string;
```

Description

Password is used to indicate the authentication credentials used when logging in to the SMTP server. User name is specified in the Username property.

These values are sent to the SMTP server in the Authenticate method.

See Also

Authenticate

Username

5.179.2.9 Port

```
property Port: integer; default 25;
```

Description

Specifies the port number for TCP/IP connection to the SMTP server.

The Connect method uses values in the HostName and **Port** properties to establish a connection for the SMTP session.

The default value is 25 port number.

See Also

Connect

HostName

5.179.2.10 ProxyOptions

```
property ProxyOptions: TProxyOptions;
```

Description

The **ProxyOptions** property holds a TProxyOptions object that contains settings for proxy connection.

If it is necessary to connect to server in another network, sometimes the client can reach it only through proxy. In this case you have to setup **ProxyOptions**.

See Also

Connect

5.179.2.11 ReplyCode

```
property ReplyCode: integer;
```

Description

The **ReplyCode** property represents the response code received during execution of the latest command request to the SMTP server.

To get the formatted response message, you can use the FormattedReply ptoperty.

This property is read-only.

See Also

FormattedReply

OnError

RaiseReplyError

5.179.2.12 SASLMechanisms

```
property SASLMechanisms: TScSASLCollection;
```

Description

Holds a collection of TScSASLItem objects that implement SASL mechanisms and store information for authentication.

SASL authentication is only employed when the AuthenticationType property is set to `satSASLMechanism`.

The Authenticate method iterates over Items and checks if a given mechanism is ready to start the authentication process. If a mechanism is ready, a login attempt is made using this mechanism.

See Also

Authenticate

AuthenticationType

5.179.2.13 SSLOptions

```
property SSLOptions: TScSSLClientOptions;
```

Description

SSLOptions determines behaviour of a TLS/SSL connection.

See also

Connect

IsSecure

5.179.2.14 Timeout

```
property Timeout: integer; default 180;
```

Description

Determines the time interval in seconds during which the client will wait for a response from the server when connecting, or wait for any data from the server when reading, or passing data to the server. After the time has expired, methods return the result and control to the program.

The default value is 180 seconds.

5.179.2.15 TLSMode

```
property TLSMode: TScTLSMode; default tmDisableTLS;
```

Description

The **TLSMode** property indicates the level of Transport Layer Security (TLS) required for control channel and data channel connections in the SMTP client.

If **TLSMode** is set to the `tmImplicitTLS` value, TLS protocol will be set in the Connect method.

If **TLSMode** is set to the `tmRequireExplicitTLS` or `tmAllowExplicitTLS` values, TLS protocol will set in the Authenticate method.

The default value is the `tmDisableTLS` value that means not to use a TLS protocol.

See Also

Authenticate

Connect

5.179.2.16 Uri

```
property Uri: string;
```

Description

Gets or sets the Uniform Resource Identifier (URI) of the SMTP server.

The **Uri** property can consist of a hostname and an optional port number. **Uri** without port information implies the default port (port 25).

An example complying with requirements that specifies a port of 465 would be the following value for the **Uri** property: 'smtp://host.com:465'.

When setting the **Uri** property, there is value parsing and the HostName, Port, Username and Password properties are reset.

See Also

Connect

HostName

Port

5.179.2.17 Username

```
property Username: string;
```

Description

User name is used as an authentication identity when logging into the SMTP server.

Password for user authentication is specified in the Password property.

These values are sent to the SMTP server in the Authenticate method.

See Also

Authenticate

Password

5.179.2.18 UseUTF8

```
property UseUTF8: boolean;
```

Description

Determines whether the UTF8 encoding will be used to encode commands sent to the SMTP server and decode responses received from the SMTP server.

When the property is True, the UTF8 encoding is used. Otherwise, the system locale is used.

Changing the value of this property may affect CEncoding.

See Also

CEncoding

5.179.3 Methods

5.179.3.1 Authenticate

```
function Authenticate: boolean;
```

Description

Authenticates a mail account with the SMTP server. Returns True is authentication was successful, otherwise False.

Use the AuthenticationType property to specify the authentication type to use when accessing an SMTP/SMTPS server.

To use SASL authentication you set AuthenticationType to `satSASLMechanism`.

Authenticate iterates over a list of SASLMechanisms and checks if a given mechanism is ready to start the authentication process. If a mechanism is ready, a login attempt is made using this mechanism.

You call **Authenticate** only once for an SMTP session.

See Also

AuthenticationType
SASLMechanisms

5.179.3.2 Connect

```
procedure Connect;
```

Description

Creates a connection to the specified SMTP server and sets the Active property to True.

This method uses values in the HostName and Port properties to establish a connection for an SMTP session.

If a server can only be accessed through proxy, you need to set up the ProxyOptions.

See Also

Active
Disconnect
HostName
Port
ProxyOptions

5.179.3.3 Disconnect

```
procedure Disconnect;
```

Description

Closes the existing connection to the SMTP server and sets the Active property to False.

See Also

Active
Connect

5.179.3.4 Expand

```
procedure Expand(const UserName: string);
```

Description

Verifies that the argument in `Username` is a valid mailing list or mailbox alias on the server. This

method uses the EXPN command to retrieve the results, with one name and email address per line.

You use the Verify method to validate an individual name or email address on the SMTP server.

See Also

Verify

5.179.3.5 RaiseReplyError

```
procedure RaiseReplyError;
```

Description

Raises the EScSMTPError exception by setting the ReplyCode value as an error code, and as a message a result obtained during execution of the latest command to the SMTP server.

See Also

ReplyCode

FormattedReply

5.179.3.6 Send

```
procedure Send(const From, Recipients, Subject, Body: string); overload;
procedure Send(Message: TScMailMessage); overload;
procedure Send(Message: TScMailMessage; const From: string); overload;
procedure Send(Message: TScMailMessage; const From: string; Recipients:
TScMailAddressList); overload;
procedure Send(Message: TScMailMessage; Recipients: TScMailAddressList);
overload;
```

Description

Sends the email message to the recipient or a list of recipients through the SMTP server.

If only a TScMailMessage object is specified in **Send**, the sender and recipients are taken from this object.

You can also set the subject, body, sender, and recipient directly as overloaded parameters in **Send**.

To add attachments to the email message you use the TScMailMessage.Attachments property.

The response code received during execution of this command is contained in the ReplyCode property.

Parameters:

- `From` - the from address of the email message;

- `Subject` - the subject line of the email message;
- `Body` - the message body as plaintext;
- `Message` - the message object that is sent to the server;
- `Recipients` - a list of recipients of the email message.

See Also

ReplyCode

TScMailMessage

TScMailAddressList

5.179.3.7 SendCmd

```
function SendCmd(const Command: string; const AllowedResponses: array of
integer): integer;
```

Description

The **SendCmd** method sends SMTP command specified in the `Command` parameter to an SMTP server.

You can specify a list of available response codes in the `AllowedResponses` parameter. If a server returns a response code not included in this list, the method will generate a corresponding exception.

The response code received during execution of this command request is contained in the ReplyCode property.

See Also

ReplyCode

5.179.3.8 Verify

```
function Verify(const UserName: string): string;
```

Description

Verifies that that the user specified in `Username` is a valid address on the SMTP server. The method returns the User name and email address on success.

You use the Expand method to determine the membership in a mailing list.

See Also

Expand

5.179.4 Events

5.179.4.1 AfterConnect

```
property AfterConnect: TNotifyEvent;
```

Description

Occurs after a connection to an SMTP/SMTPS server is established.

See Also

Active
AfterDisconnect
BeforeConnect
BeforeDisconnect

5.179.4.2 AfterDisconnect

```
property AfterDisconnect: TNotifyEvent;
```

Description

Occurs after the connection to an SMTP/SMTPS server becomes closed.

See Also

Active
AfterConnect
BeforeConnect
BeforeDisconnect

5.179.4.3 BeforeConnect

```
property BeforeConnect: TNotifyEvent;
```

Description

Occurs immediately before establishing a connection to an SMTP/SMTPS server.

See Also

Active

AfterConnect
AfterDisconnect
BeforeDisconnect

5.179.4.4 BeforeDisconnect

```
property BeforeDisconnect: TNotifyEvent;
```

Description

Occurs immediately before the connection to an SMTP/SMTPS server becomes closed.

See Also

Active
AfterConnect
AfterDisconnect
BeforeConnect

5.179.4.5 OnError

```
type
  TScSMTPErrorEvent = procedure (Sender: TObject; ErrorCode: integer;
const ErrorMessage: string; var Fail: boolean) of object;

property OnError: TScSMTPErrorEvent;
```

Description

The **OnError** event occurs when the server returns an error when executing some operation.

Parameters:

- `Sender` - the object that raised the event;
- `ErrorCode` - holds the error code;
- `ErrorMessage` - holds the readable description of the error;
- `Fail` - set the `Fail` parameter to False to prevent raising an exception, and set this parameter to True to raise the EScSMTPError exception.

See Also

ReplyCode

5.179.4.6 OnReadReply

```
type
  TScReadLineEvent = procedure (Sender: TObject; const Line: string) of
object;

property OnReadReply: TScReadLineEvent;
```

Description

Occurs when the client gets a response from the SMTP/SMTPS server. This event can be used to log responses from the server.

Parameters:

- `Sender` - the object that raised the event;
- `Line` - the response from the SMTP server.

See Also

OnSendCommand

5.179.4.7 OnRecipientError

```
type
  TScSMTPRecipientErrorEvent = procedure (Sender: TObject; const
RecipientAddress: string;
    ReplyCode: integer; const ReplyText: string; var Skip: boolean) of
object;

property OnRecipientError: TScSMTPRecipientErrorEvent;
```

Description

Occurs in the Send method if an error occurred on attempt to send an email message to the recipient. The recipient's address is specified in the `RecipientAddress` parameter.

To proceed with sending the message to other recipients you set the `Skip` parameter to True. To terminate the session and return an error you set the `Skip` parameter to False.

Parameters:

- `Sender` - the object that raised the event;

- `RecipientAddress` - delivery to this recipient failed;
- `ReplyCode` - the server reply code on attempt to send a message to this recipient;
- `ReplyText` - a readable description of the reply;
- `Skip` - whether to ignore the recipient specified in the `RecipientAddress` if an error occurs. Set `Skip` to True if you want to skip sending a message to this recipient. Set `Skip` to False if you want to cancel sending the message, and return an error.

See Also

OnError

Send

5.179.4.8 OnSendCommand

```
type
  TScSendLineEvent = procedure (Sender: TObject; const Line: string) of
object;

property OnSendCommand: TScSendLineEvent;
```

Description

Occurs after the client has sent an SMTP command to the SMTP/SMTPS server. This event can be used to log commands sent to the server.

Parameters:

- `Sender` - the object that raised the event;
- `Line` - the SMTP command that was sent to the server.

See Also

OnReadReply

5.180 TScHubConnection

5.180.1 Description

Unit

ScSignalRHubConnection

Description

The **TScHubConnection** component implements the functionality of a SignalR client. A hub connection is used to invoke hub methods on the SignalR server.

Before invoking the hub methods, a connection must be established using the Start method. A connection can be dropped using the Stop method.

The **TScHubConnection** component can be configured to automatically reconnect using the ReconnectPolicy property.

To define the methods, the hub calls the Register method before establishing the connection. This method registers a handler that is invoked when the hub method is invoked.

To call a hub method on the server, use the Invoke or Send methods.

The **TScHubConnection** class raises the HubException exception when an error occurs while processing a message from the SignalR server.

See Also

Invoke

Register

Send

Start

Stop

5.180.2 Properties

5.180.2.1 ConnectionId

```
property ConnectionId: string;
```

Description

The **ConnectionId** property gets the current ID of the connection. **ConnectionId** will be cleared when the connection is stopped and will have a new value every time the connection is (re) established.

This value will be empty if the negotiation step is skipped via TScHttpConnectionOptions.SkipNegotiation.

See Also

AfterConnect

AfterReconnect

Start

Stop

5.180.2.2 EventsCallMode

```
property EventsCallMode: TScEventCallMode; default ecDirectly;
```

Description

The **EventsCallMode** property determines how the event handler will be called. Data coming from the server is processed in a separate thread of the HubConnection connection, while the event handler can be invoked in a different way for synchronization with the main thread of the application.

The default value is `ecAsynchronous`: events are added to a queue and then asynchronously synchronized with the main thread. This prevents a slowdown in the thread where events occur, and at the same allows calling the event handlers in the main thread.

When the property is set to `ecSynchronous`, the event call is immediately synchronized with the main thread.

When the property is set to `ecDirectly` , no synchronization occurs with the main thread.

The default value is `ecDirectly`.

See also

AfterDisconnect

AfterReconnect

BeforeReconnect

Register

5.180.2.3 HandshakeTimeout

```
property HandshakeTimeout: integer; default 15;
```

Description

The **HandshakeTimeout** property gets or sets the timeout for the initial handshake.

HandshakeTimeout determines the time interval in seconds during which the client will wait for a response from the server when an initial handshake is attempted.

The default value is 15 seconds.

5.180.2.4 HttpConnectionOptions

```
property HttpConnectionOptions: TScHttpConnectionOptions;
```

Description

The **HttpConnectionOptions** property determines the behaviour of the SignalR client.

5.180.2.5 KeepAliveInterval

```
property KeepAliveInterval: integer; default 15;
```

Description

The **KeepAliveInterval** property determines the time interval in seconds during which the client sends ping messages. Sending any message resets the interval timer.

The default value is 15 seconds.

5.180.2.6 Logger

```
property Logger: TScLogger;
```

Description

The **Logger** property determines an object to write debug, error, informational and warning messages.

To log various messages, TScHubConnection calls the corresponding events of the TScLogger class. By default, messages are not written anywhere. To handle messages in some way, a user must set the event handlers for the necessary types of messages.

5.180.2.7 ReconnectPolicy

```
property ReconnectPolicy: TScRetryPolicy;
```

Description

The **ReconnectPolicy** property determines the timing and number of automatic reconnect attempts of TScHubConnection.

The default reconnect policy is the **ReconnectPolicy** used by the Create method to initialize the connection.

If you want to have control over the timing and number of automatic reconnect attempts, you should create an inheritor class of the TScRetryPolicy class and assign its object to this property.

You can also use the TScDefaultRetryPolicy class that implements the default behavior: TScHubConnection makes reconnect attempts immediately after connection is lost. If the first reconnect attempt fails, the second reconnect attempt will be made in 2 seconds. If the second reconnect attempt fails, the third reconnect attempt will be made in 10 seconds. If the third reconnect attempt fails, the fourth reconnect attempt will be made in 30 seconds. After the fourth reconnect attempt fails, no reconnect attempts will be made.

See Also

TScHubConnection.AfterReconnect

TScHubConnection.BeforeReconnect

5.180.2.8 ServerTimeout

```
property ServerTimeout: integer; default 30;
```

Description

The **ServerTimeout** property determines the time interval in seconds during which the connection will wait for a response from the server. The client times out if the server does not respond over this period of time.

The default value is 30 seconds.

5.180.2.9 State

```
property State: TScHubConnectionState;
```

Description

The **State** property indicates the state of the TScHubConnection to the server.

5.180.2.1(Url

```
property Url: string;
```

Description

The **Url** property represents the URL used by TScHubConnection. This address will be used by the HTTP-based transports to connect to the specified URL.

The default value is the URL used by the Create method to initialize the connection.

See Also

TScHttpConnectionOptions.Transports

5.180.3 Methods

5.180.3.1 Create

```
constructor Create(Owner: TComponent); overload; override;
constructor Create(const Url: string; const Transports:
```

```
    TScHttpTransportTypes = []; Logger: TScLogger = nil); reintroduce;
    overload;
constructor Create(const Url: string; const Transports:
    TScHttpTransportTypes; HttpConnectionOptions:
    TScHttpConnectionOptions; ReconnectPolicy: TScRetryPolicy; Logger:
    TScLogger); reintroduce; overload;
```

Description

The **Create** method creates a **TScHubConnection** instance and initialize it with the specified properties.

The `Url` parameter is the address that identifies the Internet resource. The Url property is set from the value of this parameter.

The `Transports` parameter specifies what transports the client should use to send HTTP requests. The HttpConnectionOptions.Transports property is set from the value of this parameter.

The `HttpConnectionOptions` parameter determines the behavior of the SignalR client. The HttpConnectionOptions property is set from the value of this parameter.

The `ReconnectPolicy` parameter is the TScRetryPolicy object that determines the timing and number of automatic reconnect attempts. The ReconnectPolicy property is set from the value of this parameter.

The `Logger` parameter is the TScLogger object service that writes debug, error, informational and warning messages. The Logger property is set from the value of this parameter.

See Also

Start

HttpConnectionOptions

Logger

ReconnectPolicy

Url

5.180.3.2 Invoke

```
procedure Invoke(const MethodName: string; const Args: array of Variant;
const ReturnType: TVarType; out Res: Variant); overload;
procedure Invoke(const MethodName: string; const Args: array of Variant;
const ReturnClass: TClass; out Res: TObject); overload;
procedure Invoke(const MethodName: string; const Args: array of Variant);
overload;
```

Description

The **Invoke** method invokes a hub method on the server using the specified method name, arguments and return type. The **Invoke** method, unlike Send, waits till the server method completes

and returns.

`MethodName` is the name of the server method to invoke.

`Args` is the arguments used to invoke the server method.

`ReturnType` is the return type of the server method.

`Res` is the hub method return value.

See Also

InvokeObj

Register

Send

5.180.3.3 InvokeObj

```
procedure InvokeObj(const MethodName: string; const Args: array of
TObject; const ReturnType: TVarType; out Res: Variant); overload;
procedure InvokeObj(const MethodName: string; const Args: array of
TObject; const ReturnClass: TClass; out Res: TObject); overload;
procedure InvokeObj(const MethodName: string; const Args: array of
TObject); overload;
```

Description

The **InvokeObj** method invokes a hub method on the server using the specified method name, arguments and return type. The **InvokeObj** method, unlike Send, waits till the server method completes and returns.

`MethodName` is the name of the server method to invoke.

`Args` is the arguments used to invoke the server method.

`ReturnType` is the return type of the server method.

`Res` is the hub method return value.

See Also

Invoke

Register

Send

5.180.3.4 Register

```
type
  TScInvocationHandlerCallback = procedure (Sender: TObject; const
Values: array of Variant) of object;

procedure Register(const MethodName: string; const Handler:
TScInvocationHandlerCallback; const ParameterTypes: array of TVarType);
overload;
procedure Register(const MethodName: string; const Handler:
TScInvocationHandlerCallback; const ParameterTypes: array of TClass);
overload;
```

Description

The **Register** method registers a handler that will be invoked when the hub method with the specified method name is invoked.

`MethodName` is the name of the hub method to be defined.

`Handler` is the handler that will be raised when the hub method is invoked.

`ParameterTypes` are the parameter types expected by the hub method.

See Also

Invoke

Send

Unregister

5.180.3.5 Send

```
procedure Send(const MethodName: string; const Args: array of Variant);
```

Description

The **Send** method invokes a hub method on the server using the specified method name and arguments. The **Send** method, unlike Invoke, does not wait for a response from the receiver.

`MethodName` is the name of the server method to invoke.

`Args` are the arguments used to invoke the server method.

See Also

Invoke

Register

SendObj

5.180.3.6 SendObj

```
procedure SendObj(const MethodName: string; const Args: array of
TObject);
```

Description

The **SendObj** method invokes a hub method on the server using the specified method name and arguments. The **SendObj** method, unlike Invoke, does not wait for a response from the receiver.

`MethodName` is the name of the server method to invoke.

`Args` are the arguments used to invoke the server method.

See Also

Invoke

Register

Send

5.180.3.7 Start

```
procedure Start;
```

Description

The **Start** method establishes a connection to the server.

See Also

Stop

5.180.3.8 Stop

```
procedure Stop;
```

Description

The **Stop** method drops a connection to the server.

See Also

Start

5.180.3.9 Unregister

```
procedure Unregister(const MethodName: string);
```

Description

The **Unregister** method removes the handler associated with the method with the specified method name.

`MethodName` is the name of the hub method from which handler is being removed.

See Also

Register

5.180.4 Events

5.180.4.1 AfterConnect

```
type
  TScHubAfterConnectEvent = procedure (Sender: TObject; const
ConnectionId: string) of object;

property AfterConnect: TScHubAfterConnectEvent;
```

Description

The **AfterConnect** event occurs after a connection to the SignalR server is established.

Parameters:

- `Sender` - the object that raised the event;
- `ConnectionId` - the ID of the established connection.

See Also

AfterReconnect

BeforeConnect

5.180.4.2 AfterDisconnect

```
type
  TScHubAfterDisconnectEvent = procedure (Sender: TObject; E: Exception)
of object;

property AfterDisconnect: TScHubAfterDisconnectEvent;
```

Description

The **AfterDisconnect** event occurs after the connection to the SignalR server is closed. The connection could be closed due to an error or due to either the server or client intentionally closing the connection without error.

Parameters:

- `Sender` - the object that raised the event;
- `E` - the Exception that describes the cause of the last connection loss. If the event is triggered by a connection error, the Exception will be passed in this argument. If the event is triggered unintentionally by either the client or the server, the argument will be nil.

See Also

AfterConnect

5.180.4.3 AfterReconnect

```
type
  TScHubAfterReconnectEvent = procedure (Sender: TObject; const
ConnectionId: string) of object;

property AfterReconnect: TScHubAfterReconnectEvent;
```

Description

The **AfterReconnect** event occurs when a connection to the SignalR server has been reestablished after it had been lost.

Parameters:

- `Sender` - the object that raised the event;
- `ConnectionId` - the ID of the reestablished connection.

See Also

AfterConnect

BeforeReconnect

5.180.4.4 BeforeConnect

```
property BeforeConnect: TNotifyEvent;
```

Description

The **BeforeConnect** event occurs immediately before establishing a connection to the SignalR server.

See Also

AfterConnect

BeforeReconnect

5.180.4.5 BeforeReconnect

```
type
  TScHubBeforeReconnectEvent = procedure (Sender: TObject; E: Exception)
of object;

property BeforeReconnect: TScHubBeforeReconnectEvent;
```

Description

The **BeforeReconnect** event occurs immediately before a reconnect attempt is made to the SignalR server after the connection has been lost or a reconnect attempt failed.

Parameters:

- `Sender` - the object that raised the event;
- `E` - the Exception that describes the cause of the last connection loss or the reconnect attempt failure.

See Also

AfterReconnect

BeforeConnect

5.181 TScHttpConnectionOptions

5.181.1 Description

Unit

ScSignalRHttpConnection

Description

The **TScHttpConnectionOptions** class defines options used to configure a TScHubConnection instance.

See also

TScHubConnection.HttpConnectionOptions

5.181.2 Properties

5.181.2.1 AccessTokenProvider

```
type
  TScGetString = function: string of object;

property AccessTokenProvider: TScGetString;
```

Description

The **AccessTokenProvider** property gets or sets the method that will be called to return an access token for each HTTP request.

5.181.2.2 CloseTimeout

```
property CloseTimeout: cardinal; default 5000;
```

Description

The **CloseTimeout** property determines the time interval in milliseconds the client will wait while the connection is being closed. After the timeout period elapses, the connection is aborted.

The default value is 5000 milliseconds.

5.181.2.3 Cookies

```
property Cookies: TStringList;
```

Description

The **Cookies** property gets or sets a collection of cookies to be sent with HTTP requests.

5.181.2.4 Credentials

```
property Credentials: TScNetworkCredential;
```

Description

The **Credentials** property gets or sets the credentials for authenticating HTTP requests.

The property contains authentication information to identify the entity that makes the request. The user, password, and domain information contained in the TScNetworkCredential object is used to authenticate the request.

5.181.2.5 Headers

```
property Headers: TScWebHeaderCollection;
```

Description

The **Headers** property defines a collection of the name-value pairs that will be sent with HTTP requests.

The collection contains the protocol headers associated with the request.

5.181.2.6 Proxy

```
property Proxy: TScWebProxy;
```

Description

The **Proxy** property gets or sets the proxy information used when making HTTP requests.

In situations where the client can only reach the server through proxy, you can setup the **Proxy** property that identifies the TScWebProxy object to process requests to the Internet resources.

5.181.2.7 SkipNegotiation

```
property SkipNegotiation: boolean;
```

Description

The **SkipNegotiation** property gets or sets a value indicating whether negotiation is skipped when connecting to the server. Negotiation can only be skipped when using the WebSockets transport.

See also

TScHttpTransportType

5.181.2.8 SSLOptions

```
property SSLOptions: TScSSLClientOptions;
```

Description

The **SSLOptions** property determines the behavior of the TLS/SSL connection.

5.181.2.9 Transports

```
property Transports: TScHttpTransportTypes; default [ttWebSockets,
ttLongPolling];
```

Description

The **Transports** property gets or sets a combination of one or more TScHttpTransportType values that specify what transports the client can use to send HTTP requests.

5.181.2.1(Url

```
property Url: string;
```

Description

The **Url** property gets or sets the URL used to send HTTP requests.

5.182 TScLogger

5.182.1 Description

Unit

ScUtils

Description

The **TScLogger** class defines methods and events to write debug, error, informational, and warning messages.

To log messages of various types, the handler calls the corresponding events of this class. By default, log messages are not written anywhere. To handle messages, a user must setup event

handlers for the required message types.

See also

TScHubConnection.Logger

5.182.2 Events

5.182.2.1 OnLogDebug

```
type
  TScOnLogMessage = procedure (Sender: TObject; const Message: string) of
object;

property OnLogDebug: TScOnLogMessage;
```

Description

The **OnLogDebug** event occurs when logger needs to format and write a debug log message.

Parameters:

- `Sender` - the object that raises the event;
- `Message` - the format string of the log message.

5.182.2.2 OnLogError

```
type
  TScOnLogError = procedure (Sender: TObject; const Message: string; E:
Exception) of object;

property OnLogError: TScOnLogError;
```

Description

The **OnLogError** event occurs when logger needs to format and write an error log message.

Parameters:

- `Sender` - the object that raises the event;
- `Message` - the format string of the log message;
- `E` - the exception to log.

5.182.2.3 OnLogInformation

```
type
  TScOnLogMessage = procedure (Sender: TObject; const Message: string) of
object;

property OnLogInformation: TScOnLogMessage;
```

Description

The **OnLogInformation** event occurs when logger needs to format and write an informational log message.

Parameters:

- `Sender` - the object that raised the event;
- `Message` - the format string of the log message.

5.182.2.4 OnLogWarning

```
type
  TScOnLogMessage = procedure (Sender: TObject; const Message: string) of
object;

property OnLogWarning: TScOnLogMessage;
```

Description

The **OnLogWarning** event occurs when logger needs to format and write a warning log message.

Parameters:

- `Sender` - the object that raised the event;
- `Message` - the format string of the log message.

5.183 TScRetryPolicy

5.183.1 Description

Unit

ScSignalRHubConnection

Description

The **TScRetryPolicy** class defines the timing and number of automatic reconnect attempts of a TScHubConnection.

If you want control over the timing and number of automatic reconnect attempts, you should create an inheritor class of the **TScRetryPolicy** class and assign its object to the TScHubConnection.ReconnectPolicy property.

The **TScRetryPolicy** class has a single method named NextRetryDelay, that is called by TScHubConnection. This is abstract method and it must be overriden by an inheritor class.

See also

TScHubConnection.ReconnectPolicy

5.183.2 Methods

5.183.2.1 NextRetryDelay

```
function NextRetryDelay(const RetryContext: TScRetryContext): cardinal;
virtual; abstract;
```

Description

The **NextRetryDelay** method returns the time to wait before the next reconnect attempt or -1 if the TScHubConnection must stop reconnect attempts.

NextRetryDelay takes a single argument of the TScRetryContext type, that has three properties: PreviousRetryCount, ElapsedTime and RetryReason which are a Int64, a Cardinal and an Exception respectively. Before the first reconnect attempt, both RetryContext.PreviousRetryCount and RetryContext.ElapsedTime are set to zero, and the RetryContext.RetryReason is the Exception that caused the connection loss. After each failed retry attempt, RetryContext.PreviousRetryCount is incremented by one, RetryContext.ElapsedTime is updated to reflect the amount of time spent reconnecting so far, and the RetryContext.RetryReason is the Exception that caused the last reconnect attempt to fail.

This method is called by TScHubConnection to determine the timing and number of automatic reconnect attempts. This is abstract method and it must be overriden by an inheritor class.

See Also

TScHubConnection.ReconnectPolicy

5.184 TScCMSSubjectIdentifier

5.184.1 Description

Unit

ScCMS

Description

The **TScCMSSubjectIdentifier** class defines the identifier of a subject, such as a TScCMSSignerInfo or a TScCMSRecipient. The subject can be identified by the certificate issuer and serial number or the subject key.

Use the Init method to initialize the instance from the X.509 certificate.

See Also

SubjectIdentifierType

5.184.2 Properties

5.184.2.1 Issuer

```
property Issuer: TScDistinguishedName;
```

Description

The **Issuer** property retrieves the Distinguished Name of the certificate issuer of the subject identifier. This property is set only if the SubjectIdentifierType property is set to sitIssuerAndSerialNumber.

This property is read-only.

See Also

Init

SubjectIdentifierType

5.184.2.2 KeyIdentifierDate

```
property KeyIdentifierDate: TDateTime;
```

Description

The **KeyIdentifierDate** property retrieves the date that specifies a key from a set that was previously distributed. This property is set only if the SubjectIdentifierType property is set to sitKeyIdentifier.

This property is read-only.

See Also

Init

SubjectIdentifierType

5.184.2.3 SerialNumber

```
property SerialNumber: string;
```

Description

The **SerialNumber** property retrieves the serial number of the subject identifier. This property is set only if the SubjectIdentifierType property is set to sitIssuerAndSerialNumber.

This property is read-only.

See Also

Init

SubjectIdentifierType

5.184.2.4 SubjectIdentifierType

```
property SubjectIdentifierType: TScCMSSubjectIdentifierType;
```

Description

The **SubjectIdentifierType** property retrieves the type of the subject identifier. The subject can be identified by the certificate issuer and serial number or the subject key.

This property is read-only.

See Also

Init

TScCMSSubjectIdentifierType

5.184.2.5 SubjectKeyIdentifier

```
property SubjectKeyIdentifier: string;
```

Description

The **SubjectKeyIdentifier** property retrieves the hash of the subject's public key of the subject identifier. The hash algorithm used is determined by the signature algorithm suite in the subject's certificate. This property is set only if the SubjectIdentifierType property is set to sitSubjectKeyIdentifier or sitKeyIdentifier.

This property is read-only.

See Also

Init

SubjectIdentifierType

5.184.3 Methods

5.184.3.1 Assign

```
procedure Assign(Source: TScCMSSubjectIdentifier);
```

Description

Copies the contents of another similar object. **Assign** copies properties of the specified `Source` object to the current object.

5.184.3.2 Init

```
procedure Init(SubjectIdentifierType: TScCMSSubjectIdentifierType;
Certificate: TScCertificate); overload;
procedure Init(Certificate: TScCertificate); overload;
```

Description

Initializes the **TScCMSSubjectIdentifier** instance from the X.509 certificate.

The `SubjectIdentifierType` parameter represents the type of a subject identifier. The SubjectIdentifierType property is set from the value of this parameter. If this parameter is not specified, the SubjectIdentifierType property will be set to the sitIssuerAndSerialNumber value.

The `Certificate` parameter is an object that represents the subject identifier. The Issuer, SerialNumber, SubjectKeyIdentifier, and KeyIdentifierDate properties are imported from this X.509 certificate. The `SubjectIdentifierType` parameter determines which of these properties will be set. If `SubjectIdentifierType` is not specified, it is considered equal to the sitIssuerAndSerialNumber value.

5.185 TScCMSOriginatorIdentifierOrKey

5.185.1 Description

Unit

ScCMS

Description

The **TScCMSOriginatorIdentifierOrKey** class defines the identifier of a TScCMSKeyAgreeRecipientInfo originator. The originator can be identified by the certificate issuer and serial number or the subject key.

Use the Init method to initialize the instance from the X.509 certificate.

See Also

OriginatorIdentifierOrKeyType

5.185.2 Properties

5.185.2.1 Issuer

```
property Issuer: TScDistinguishedName;
```

Description

The **Issuer** property retrieves the Distinguished Name of the certificate issuer of the originator identifier. This property is set only if the OriginatorIdentifierOrKeyType property is set to oitIssuerAndSerialNumber.

This property is read-only.

See Also

Init

OriginatorIdentifierOrKeyType

5.185.2.2 OriginatorIdentifierOrKeyType

```
property OriginatorIdentifierOrKeyType:
TScCMSOriginatorIdentifierOrKeyType;
```

Description

The **OriginatorIdentifierOrKeyType** property retrieves the type of the originator identifier. The originator can be identified by the certificate issuer and serial number or the originator key.

This property is read-only.

See Also

Init

TScCMSOriginatorIdentifierOrKeyType

5.185.2.3 PublicKey

```
property PublicKey: TBytes;
```

Description

The **PublicKey** property retrieves public key of the originator identifier. This property is set only if the OriginatorIdentifierOrKeyType property is set to oitPublicKeyInfo.

This property is read-only.

See Also

Init

OriginatorIdentifierOrKeyType

5.185.2.4 PublicKeyAlgorithmIdentifier

```
property PublicKeyAlgorithmIdentifier: TScASN1AlgorithmIdentifier;
```

Description

The **PublicKeyAlgorithmIdentifier** property retrieves the algorithm identifier of the public key of the originator identifier. This property is set only if the OriginatorIdentifierOrKeyType property is set to oitPublicKeyInfo.

This property is read-only.

See Also

Init

OriginatorIdentifierOrKeyType

5.185.2.5 SerialNumber

```
property SerialNumber: string;
```

Description

The **SerialNumber** property retrieves the serial number of the originator identifier. This property is set only if the OriginatorIdentifierOrKeyType property is set to oitIssuerAndSerialNumber.

This property is read-only.

See Also

Init

OriginatorIdentifierOrKeyType

5.185.2.6 SubjectKeyIdentifier

```
property SubjectKeyIdentifier: string;
```

Description

The **SubjectKeyIdentifier** property retrieves the hash of the originator's public key of the originator identifier. The hash algorithm used is determined by the signature algorithm suite in the originator's certificate. This property is set only if the OriginatorIdentifierOrKeyType property is set to oitSubjectKeyIdentifier.

This property is read-only.

See Also

Init

OriginatorIdentifierOrKeyType

5.185.3 Methods

5.185.3.1 Assign

```
procedure Assign(Source: TScCMSOriginatorIdentifierOrKey);
```

Description

Copies the contents of another similar object. **Assign** copies properties of the specified `Source` object to the current object.

5.185.3.2 Init

```
procedure Init(OriginatorIdentifierOrKeyType:
TScCMSOriginatorIdentifierOrKeyType; Certificate: TScCertificate);
overload;
procedure Init(Certificate: TScCertificate); overload;
```

Description

Initializes the **TScCMSOriginatorIdentifierOrKey** instance from the X.509 certificate.

The `OriginatorIdentifierOrKeyType` parameter represents the type of a originator identifier. The OriginatorIdentifierOrKeyType property is set from the value of this parameter. If this parameter is not specified, the OriginatorIdentifierOrKeyType property will be set to the oitIssuerAndSerialNumber value.

The `Certificate` parameter is an object that represents the originator identifier. The Issuer, SerialNumber, and SubjectKeyIdentifier properties are imported from this X.509 certificate. The `OriginatorIdentifierOrKeyType` parameter determines which of these properties will be set. If `OriginatorIdentifierOrKeyType` is not specified, it is considered equal to the oitIssuerAndSerialNumber value.

5.186 TScCMSSignedAttributes

5.186.1 Description

Unit

ScCMS

Description

The **TScCMSSignedAttributes** class maintains a list of the TScPKCS7Attribute objects.

TScCMSSignedAttributes stores the collection of signed attributes that is associated with the signer information.

5.187 TScCMSUnsignedAttributes

5.187.1 Description

Unit

ScCMS

Description

The **TScCMSUnsignedAttributes** class maintains a list of the TScPKCS7Attribute objects.

TScCMSUnsignedAttributes stores the collection of unsigned attributes that is associated with the signer information.

5.188 TScCMSSMIMEAttributes

5.188.1 Description

Unit

ScCMS

Description

The **TScCMSSMIMEAttributes** class is a descendant of the TScASN1Attributes class.

Use **TScCMSSMIMEAttributes** to store and maintain a list of the TScASN1Attribute objects and to encode the information in the object into the PKCS #7 format.

See also

TScCMSSignerInfo.SMIMEAttribute

5.188.2 Methods

5.188.2.1 Encode

```
function Encode: TBytes;
```

Description

The **Encode** method encodes the list of the TScASN1Attribute objects into the PKCS #7 format.

See Also

Decode

5.188.2.2 Decode

```
procedure Decode(const RawData: TBytes);
```

Description

The **Decode** method decodes the information from the PKCS #7 format into the list of the TScASN1Attribute objects.

See Also

Encode

5.189 TScCMSContentInfo

5.189.1 Description

Unit

ScCMS

Description

The **TScCMSContentInfo** class represents the CMS/PKCS #7 ContentInfo data structure as defined in the CMS/PKCS #7 standards document (RFC 5652). This data structure stores content of a CMS message and it is the basis for all CMS/PKCS #7 messages.

Use the Init method to initialize the instance from the specified data.

5.189.2 Properties

5.189.2.1 ContentBuffer

```
property ContentBuffer: TBytes;
```

Description

The **ContentBuffer** property is an array of byte values that represents the content of the CMS/PKCS #7 message. This property has meaning only if the ContentStream.Stream is nil.

This property is read-only.

See Also

Init

5.189.2.2 ContentStream

```
property ContentStream: TScStreamInfo;
```

Description

The **ContentStream** property is an object that represents the content of the CMS/PKCS #7 message. If the **ContentStream**.Stream is nil, the ContentBuffer property is used.

This property is read-only.

See Also

Init

5.189.2.3 ContentType

```
property ContentType: TScOId;
```

Description

The **ContentType** property retrieves the TScOId object that contains the object identifier (OID) of the content type of the inner content of the CMS/PKCS #7 message. This can be data, digested data, encrypted data, enveloped data, hashed data, signed and enveloped data, or signed data.

This property is read-only.

See Also

Init

5.189.3 Methods

5.189.3.1 Assign

```
procedure Assign(Source: TScCMSContentInfo);
```

Description

Copies the contents of another similar object. **Assign** copies properties of the specified `Source` object to the current object.

5.189.3.2 GetContentData

```
function GetContentData: TBytes;
```

Description

The **GetContentData** method returns an array of byte values that represents the content of the CMS/PKCS #7 message. If the ContentStream.Stream is not nil, data is read from this stream, else data is copied from the ContentBuffer property.

See Also

ContentBuffer

ContentStream

5.189.3.3 Init

```
procedure Init(ContentType: TScOId; const ContentBuffer: TBytes);
overload;
procedure Init(ContentType: TScOId; ContentStream: TStream); overload;
procedure Init(const ContentBuffer: TBytes); overload;
procedure Init(ContentStream: TStream); overload;
```

Description

Initializes the **TScCMSContentInfo** instance from the specified content type and the specified data.

The `ContentType` parameter is TScOId object that contains an object identifier (OID) that specifies the content type of the content. This can be data, digested data, encrypted data, enveloped data, hashed data, signed and enveloped data, or signed data. The ContentType property is set from the value of this parameter. If this parameter is not specified, the ContentType property will be set to the Data content type (1.2.840.113549.1.7.1).

The `ContentBuffer` parameter is an array of byte values that represents the data from which to initialize the **TScCMSContentInfo** object. The ContentBuffer property is set from the value of this parameter.

The `ContentStream` parameter is an object that represents the data from which to initialize the **TScCMSContentInfo** object. The ContentStream property is set from the value of this parameter.

Content type	OID string
Data	1.2.840.113549.1.7.1
DigestedData	1.2.840.113549.1.7.5
EncryptedData	1.2.840.113549.1.7.6
EnvelopedData	1.2.840.113549.1.7.3
HashedData	1.2.840.113549.1.7.5
SignedAndEnvelopedData	1.2.840.113549.1.7.4
SignedData	1.2.840.113549.1.7.2

5.190 TScCMSSignerInfo

5.190.1 Description

Unit

ScCMS

Description

The **TScCMSSignerInfo** class represents a signer associated with a TScCMSSignedData object that represents a CMS/PKCS #7 message, described in RFC 5652.

5.190.2 Properties

5.190.2.1 Certificate

```
property Certificate: TScCertificate;
```

Description

The **Certificate** property sets or retrieves the signing certificate associated with the signer information.

See Also

SignerIdentifier

5.190.2.2 ContentType

```
property ContentType: string;
```

Description

The **ContentType** property specifies the content type attribute that is associated with the signer information. This attribute is signed along with the rest of the message content.

Depending on the IncludedAttributes property this attribute can be automatically generated and placed in the SignedAttributes list.

See Also

IncludedAttributes

SignedAttributes

5.190.2.3 DigestAlgorithm

```
property DigestAlgorithm: TScHashAlgorithm;
```

Description

The **DigestAlgorithm** property represents the hash algorithm used in the computation of the signatures.

The **DigestAlgorithm** value is calculated based on the DigestAlgorithmIdentifier OID. Setting the **DigestAlgorithm** property changes the DigestAlgorithmIdentifier property.

The default value is the haSHA2_256 algorithm.

See Also

DigestAlgorithmIdentifier

5.190.2.4 DigestAlgorithmIdentifier

```
property DigestAlgorithmIdentifier: TScASN1AlgorithmIdentifier;
```

Description

The **DigestAlgorithmIdentifier** property sets or retrieves the TScASN1AlgorithmIdentifier object that represents the hash algorithm used in the computation of the signatures.

Setting the **DigestAlgorithmIdentifier** property changes the DigestAlgorithm property.

The default value is the OID_SHA256 (2.16.840.1.101.3.4.2.1) algorithm identifier.

See Also

DigestAlgorithm

5.190.2.5 IncludedAttributes

```
property IncludedAttributes: TScCMSIncludedAttributes;
```

Description

The **IncludedAttributes** property sets or retrieves signed attributes that will be automatically generated and placed in the SignedAttributes property.

When any TScCMSIncludedAttribute flag is included in the **IncludedAttributes** set, the corresponding object representing the attribute will be added to the SignedAttributes list (if this attribute yet is not included in the list) on the signature calculation.

When any TScCMSIncludedAttribute flag is excluded from the **IncludedAttributes** set, the corresponding object representing the attribute will be deleted from the SignedAttributes list (if this attribute already is included in the list) on the signature calculation.

By default, the content type attribute (ciaContentType), the message digest attribute (ciaMessageDigest), and the signing time attribute (ciaSigningTime) will be included in the signed attributes.

See Also

ContentType

MessageDigest

SigningTime

SignedAttributes

5.190.2.6 MessageDigest

```
property MessageDigest: TBytes;
```

Description

The **MessageDigest** property specifies the message digest attribute that is associated with the signer information. This attribute is signed along with the rest of the message content.

Depending on the IncludedAttributes property this attribute can be automatically generated and placed in the SignedAttributes list.

See Also

IncludedAttributes

SignedAttributes

5.190.2.7 SignatureAlgorithm

```
property SignatureAlgorithm: TScSignatureAlgorithm;
```

Description

The **SignatureAlgorithm** property represents the signing algorithm used in the computation of the signatures.

The **SignatureAlgorithm** value is calculated based on the SignatureAlgorithmIdentifier OID. Setting the **SignatureAlgorithm** property changes the SignatureAlgorithmIdentifier property.

The default value is the saRSA_Encryption algorithm.

See Also

SignatureAlgorithmIdentifier

5.190.2.8 SignatureAlgorithmIdentifier

```
property SignatureAlgorithmIdentifier: TScASN1AlgorithmIdentifier;
```

Description

The **SignatureAlgorithmIdentifier** property sets or retrieves the TScASN1AlgorithmIdentifier object that represents the signing algorithm used in the computation of the signatures.

Setting the **SignatureAlgorithmIdentifier** property changes the SignatureAlgorithm property.

The default value is the OID_RSA_ENCRYPTION (1.2.840.113549.1.1.1) algorithm identifier.

See Also

SignatureAlgorithm

5.190.2.9 SignedAttributes

```
property SignedAttributes: TScCMSSignedAttributes;
```

Description

The **SignedAttributes** property sets or retrieves the TScCMSSignedAttributes list of signed attributes that is associated with the signer information. Signed attributes are signed along with the rest of the message content.

Signed attributes are signed along with the rest of the TScCMSSignedData message content. This means that a party that successfully verifies the signature can have confidence that the contents of these attributes are authentic and have not been altered.

Depending on the IncludedAttributes property the content type attribute, the message digest attribute, the signing time attribute, and the SMIME attribute can be automatically generated and placed in the **SignedAttributes** list.

See Also

IncludedAttributes

ContentType

MessageDigest

SigningTime

SMIMEAttribute

UnsignedAttributes

5.190.2.10 SignerIdentifier

```
property SignerIdentifier: TScCMSSubjectIdentifier;
```

Description

The **SignerIdentifier** property sets or retrieves the certificate identifier of the signer associated with the signer information. A **SignerIdentifier** object uniquely identifies the signer certificate.

See Also

Certificate

5.190.2.11 SigningTime

```
property SigningTime: TDateTime;
```

Description

The **SigningTime** property specifies the signing time attribute that is associated with the signer information. This attribute is signed along with the rest of the message content.

Depending on the IncludedAttributes property this attribute can be automatically generated and placed in the SignedAttributes list.

See Also

IncludedAttributes

SignedAttributes

5.190.2.12 SMIMEAttribute

```
property SMIMEAttribute: TScCMSSMIMEAttributes;
```

Description

The **SMIMEAttribute** property specifies the SMIME attribute that is associated with the signer information. This attribute is signed along with the rest of the message content.

Depending on the IncludedAttributes property this attribute can be automatically generated and placed in the SignedAttributes list.

See Also

IncludedAttributes

SignedAttributes

5.190.2.13 UnsignedAttributes

```
property UnsignedAttributes: TScCMSUnsignedAttributes;
```

Description

The **UnsignedAttributes** property sets or retrieves the TScCMSUnsignedAttributes list of unsigned attributes that is associated with the TScCMSSignedData content. Unsigned attributes can be modified without invalidating the signature.

Unsigned attributes are not signed along with the rest of the TScCMSSignedData message content. Even though a party successfully verifies the signature, the unsigned attributes may have been altered and should not be considered to have authenticity or integrity.

See Also

SignedAttributes

5.190.2.1 Version

```
property Version: integer;
```

Description

The **Version** property retrieves the signer information version. The version determines whether the message is a PKCS #7 message or a Cryptographic Message Syntax (CMS) message. CMS is a newer superset of PKCS #7.

This property is read-only.

5.190.3 Methods

5.190.3.1 CalcHash

```
function CalcHash(const Content: TBytes): TBytes; overload;
function CalcHash(Stream: TStream; Count: Int64 = 0): TBytes; overload;
```

Description

The **CalcHash** method computes the hash value for the input data using the hash algorithm specified in the DigestAlgorithm property. The input data can be specified in the `Content` parameter as a byte array, or in the `Stream` parameter as a TStream object.

If the `Stream` parameter is used and the `Count` parameter is equal to 0, `Stream.Position` is set to 0 and the `Stream.Size` data count is used for computing the hash value.

If the `Stream` parameter is used and the `Count` parameter is more than 0, `Stream.Position` is not changed and the data count specified in the `Count` parameter is used for computing the hash value.

See Also

CheckHash

DigestAlgorithm

5.190.3.2 CheckHash

```
procedure CheckHash(const Content: TBytes); overload;
procedure CheckHash(Stream: TStream; Count: Int64 = 0); overload;
```

Description

The **CheckHash** method verifies the data integrity of the input data using the hash algorithm specified in the DigestAlgorithm property. The input data can be specified in the `Content` parameter as a byte array, or in the `Stream` parameter as a TStream object.

If the `Stream` parameter is used and the `Count` parameter is equal to 0, `Stream.Position` is set to 0 and the `Stream.Size` data count is used for computing the hash value.

If the `Stream` parameter is used and the `Count` parameter is more than 0, `Stream.Position` is not changed and the data count specified in the `Count` parameter is used for computing the hash value.

This method raises an exception if the verification of the data integrity fails.

Note: CheckHash does not authenticate the signer information because this method does not involve verifying a digital signature. For purpose checking of the integrity and authenticity of CMS/PKCS #7 message signer information, use the CheckSignature method.

See Also

CalcHash

DigestAlgorithm

5.190.3.3 Create

```
constructor Create; overload;
constructor Create(SignerIdentifierType: TScCMSSubjectIdentifierType;
Certificate: TScCertificate); overload;
constructor Create(Certificate: TScCertificate); overload;
```

Description

Create **TScCMSSignerInfo** instance.

The `SubjectIdentifierType` parameter represents the type of a certificate identifier. The SignerIdentifier.SubjectIdentifierType property is set from the value of this parameter.

The `Certificate` parameter is an object that represents the signing certificate associated with the signer information. The Certificate property is set from the value of this parameter. Also properties of the SignerIdentifier object are imported from this X.509 certificate.

5.191 TScCMSSignature

5.191.1 Description

Unit

ScCMS

Description

The **TScCMSSignature** class represents a signer associated with a TScCMSSignedData object that represents a CMS/PKCS #7 message, described in RFC 5652.

TScCMSSignature stores the required information and provides functionality to validate the CMS signature or sign the CMS message.

The signatures represented by the **TScCMSSignature** class can be either over message content or a signature. This class should not be publicly instantiated. It is a read-only class accessible from the TScCMSSignedData.Signatures property.

5.191.2 Properties

5.191.2.1 Signature

```
property Signature: TBytes;
```

Description

The **Signature** property retrieves the digital signature of the CMS message. This property can be set by calling the ComputeSignature method, or it can be set automatically on decoding a CMS message.

This property is read-only.

See Also

CheckSignature

ComputeSignature

5.191.3 Methods

5.191.3.1 CheckSignature

```
procedure CheckSignature;
```

Description

The **CheckSignature** method verifies the digital signature of the CMS message by using the signing certificate specified in the Certificate property, and the signing algorithm specified in the SignatureAlgorithm property.

CheckSignature computes hash value for the CMS message of a TScCMSSignedData object and verifies it using the signature specified in the Signature property.

This method raises an exception if the verification of the digital signature fails.

See Also

ComputeSignature

Signature

5.191.3.2 ComputeSignature

```
procedure ComputeSignature;
```

Description

The **ComputeSignature** method computes a digital signature of the CMS message by using the signing certificate specified in the Certificate property, and the signing algorithm specified in the SignatureAlgorithm property.

The computed digital signature can be retrieved from the Signature property.

See Also

CheckSignature

Signature

5.192 TScCMSSignatures

5.192.1 Description

Unit

ScCMS

Description

TScCMSSignatures maintains a list of the TScCMSSignature objects.

Use **TScCMSSignatures** to store and maintain a list of objects. **TScCMSSignatures** provides properties and methods to add, delete, locate, and access objects. **TScCMSSignatures** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScCMSSignatures** instance is itself destroyed.

See also

TScCMSSignature

5.192.2 Properties

5.192.2.1 Signatures

```
property Signatures[Index: integer]: TScCMSSignature; default;
```

Description

Lists the TScCMSSignature object references.

Use **Signatures** to access objects in the list. **Signatures** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **Signatures** with the Count property to iterate through the list.

Reassigning a **Signatures** index frees the object that previously occupied that position in the list.

Note: Signatures is the default property of TScCMSSignatures. This means you can omit the property name.

See also

Count

TScCMSSignature

5.193 TScCMSData

5.193.1 Description

Unit

ScCMS

Description

TScCMSData is an abstract base class, which is the ancestor for all CMS messages classes, like TScCMSSignedData and TScCMSEnvelopedData.

See also

TScCMSSignedData

TScCMSEnvelopedData

5.194 TScCMSSignedData

5.194.1 Description

Unit

ScCMS

Description

The **TScCMSSignedData** class represents a signed message according to CMS/PKCS #7, described in RFC 5652.

TScCMSSignedData stores the required information and enables signing and verifying of CMS/

PKCS #7 messages.

To sign content of any type, in the beginning initialize the **TScCMSSignedData** object with the required content using the Init method. After this, for each signer, create and initialize the TScCMSSignerInfo object specifying a signer's certificate with a private key, required algorithms, signed and unsigned attributes, and other parameters, and sign the information using the ComputeSignature method. And finally, call the Encode method to get an encoded CMS message.

To verify CMS/PKCS #7 message, in the beginning, decode the message using the Decode method. This method sets all properties of the **TScCMSSignedData** object by using the decoded message and a user can retrieve the required properties. The inner contents of the decoded message can be retrieved from the ContentInfo property if it was included in the encoded message. To verify the decoded message use the CheckSignature method specifying the signer's certificate.

See Also

Init

CheckSignature

ComputeSignature

5.194.2 Properties

5.194.2.1 Certificates

```
property Certificates: TScCertificateList;
```

Description

The **Certificates** property represents the list of certificates associated with the encoded CMS/PKCS #7 message.

5.194.2.2 ContentInfo

```
property ContentInfo: TScCMSContentInfo;
```

Description

The **ContentInfo** property retrieves the inner contents of the encoded CMS/PKCS #7 message.

This property is read-only.

ContentInfo is set as a result of calling the Decode and Init methods.

See Also

ComputeSignature

Decode

Init

5.194.2.3 Signatures

```
property Signatures: TScCMSSignatures;
```

Description

The **Signatures** property retrieves the TScCMSSignatures list associated with the CMS/PKCS #7 message.

This property is read-only.

The **Signatures** list is populated as a result of calling the Decode and ComputeSignature methods.

See Also

ComputeSignature

5.194.3 Methods

5.194.3.1 CheckSignature

```
procedure CheckSignature(Certificate: TScCertificate);
```

Description

The **CheckSignature** method verifies the digital signature on the signed CMS/PKCS #7 message by using the certificate specified in the `Certificate` parameter.

The method finds the signature corresponding to the specified certificate and verifies it. If there are signed attributes included with the message, these attributes are also verified.

CheckSignature raises an exception if the verification of a digital signature fails.

See Also

ComputeSignature

5.194.3.2 ComputeSignature

```
procedure ComputeSignature(SignerInfo: TScCMSSignerInfo);
```

Description

The **ComputeSignature** method creates a signature using the specified signer and adds the

signature to the CMS/PKCS #7 message.

The `SignerInfo` parameter is an object that represents the signer.

The computed signature is added to the Signatures list.

See Also

CheckSignature

5.194.3.3 Decode

```
procedure Decode(const RawData: TBytes); overload;
procedure Decode(Stream: TStream); overload;
```

Description

The **Decode** method decodes an encoded signed CMS/PKCS #7 message. Upon successful decoding, the decoded information can be retrieved from the properties of the TScCMSSignedData object.

`RawData` is an array of byte values, and `Stream` is a TStream object, that represent the encoded CMS/PKCS #7 message to be decoded.

This method resets all properties of the object by using the information obtained from successful decoding.

See Also

Encode

5.194.3.4 Encode

```
function Encode(IncludeContent: boolean = False): TBytes; overload;
procedure Encode(Stream: TStream; IncludeContent: boolean = False);
overload;
```

Description

The **Encode** method encodes the information in the object into a signed CMS/PKCS #7 message. Signing must be done before encoding.

This method can return an array of byte values that represents the encoded message or can write this result to the `Stream` parameter.

The `IncludeContent` parameter is a boolean value that specifies whether the signature of the TScCMSSignedData object is for detached content. If `IncludeContent` is True, the signed content is included in the CMS/PKCS #7 message along with the signature information. If the `IncludeContent` state is False (by default), the message does not contain the signed content, and a client can see the content of the message only if it is sent separately.

The encoded message can be decoded by the Decode method.

See Also

Decode

5.194.3.5 Init

```
procedure Init(ContentInfo: TScCMSContentInfo); overload;
procedure Init(const ContentBuffer: TBytes); overload;
procedure Init(ContentStream: TStream); overload;
```

Description

Initializes the **TScCMSSignedData** instance by using the specified content information as the inner content.

The `ContentInfo`, `ContentBuffer`, and `ContentStream` parameters represent the content information as the inner content of the signed message. The ContentInfo property is set from the value of this parameter.

The **Init** method clears the Signatures list.

See also

ContentInfo

Signatures

5.195 TScCMSRecipient

5.195.1 Description

Unit

ScCMS

Description

The **TScCMSRecipient** class defines the recipient of a CMS/PKCS #7 message, described in RFC 5652.

5.195.2 Properties

5.195.2.1 Certificate

```
property Certificate: TScCertificate;
```

Description

The **Certificate** property retrieves the certificate associated with the recipient.

This property is read-only.

See Also

Init

5.195.2.2 RecipientIdentifierType

```
property RecipientIdentifierType: TScCMSSubjectIdentifierType;
```

Description

The **RecipientIdentifierType** property retrieves the type of the identifier of the recipient.

This property is read-only.

See Also

Init

5.195.3 Methods

5.195.3.1 Create

```
constructor Create; overload;
constructor Create(RecipientIdentifierType: TScCMSSubjectIdentifierType;
Certificate: TScCertificate); overload;
constructor Create(Certificate: TScCertificate); overload;
```

Description

Create **TScCMSRecipient** instance by using the specified recipient identifier type and recipient certificate.

The `RecipientIdentifierType` parameter represents the type of the identifier of the recipient. The RecipientIdentifierType property is set from the value of this parameter. If this parameter is not specified, the RecipientIdentifierType property will be set to the sitIssuerAndSerialNumber value.

The `Certificate` parameter represents the recipient certificate. The Certificate property is set from the value of this parameter.

See Also

Certificate

RecipientIdentifierType

5.195.3.2 Init

```
procedure Init(RecipientIdentifierType: TScCMSSubjectIdentifierType;
Certificate: TScCertificate); overload;
procedure Init(Certificate: TScCertificate); overload;
```

Description

Initializes the **TScCMSRecipient** instance by using the specified recipient identifier type and recipient certificate.

The `RecipientIdentifierType` parameter represents the type of the identifier of the recipient. The RecipientIdentifierType property is set from the value of this parameter. If this parameter is not specified, the RecipientIdentifierType property will be set to the sitIssuerAndSerialNumber value.

The `Certificate` parameter represents the recipient certificate. The Certificate property is set from the value of this parameter.

See Also

Certificate

RecipientIdentifierType

5.196 TScCMSRecipientInfo

5.196.1 Description

Unit

ScCMS

Description

The **TScCMSRecipientInfo** class represents information about a CMS/PKCS #7 message recipient, described in RFC 5652.

TScCMSRecipientInfo is an abstract class inherited by the TScCMSKeyTransRecipientInfo, TScCMSKeyAgreeRecipientInfo, TScCMSKEKRecipientInfo, and TScCMSPasswordRecipientInfo classes.

See also

TScCMSKeyTransRecipientInfo

TScCMSKeyAgreeRecipientInfo

TScCMSKEKRecipientInfo

TScCMSPasswordRecipientInfo

5.196.2 Properties

5.196.2.1 EncryptedKey

```
property EncryptedKey: TBytes;
```

Description

The **EncryptedKey** property retrieves the encrypted recipient keying material.

This property is read-only.

5.196.2.2 KeyEncryptionAlgorithmIdentifier

```
property KeyEncryptionAlgorithmIdentifier: TScASN1AlgorithmIdentifier;
```

Description

The **KeyEncryptionAlgorithmIdentifier** property retrieves the TScASN1AlgorithmIdentifier object that contains the value of the algorithm used to establish the key between the originator and recipient of the CMS/PKCS #7 message.

This property is read-only.

5.196.2.3 RecipientInfoType

```
property RecipientInfoType: TScCMSRecipientInfoType;
```

Description

The **RecipientInfoType** property retrieves the type of the recipient. The type of the recipient determines which of protocols is used to establish a key between the originator and the recipient of a CMS/PKCS #7 message.

This property is read-only.

5.197 TScCMSKeyTransRecipientInfo

5.197.1 Description

Unit

ScCMS

Description

The **TScCMSKeyTransRecipientInfo** class represents key transport recipient information, described in RFC 5652.

Key transport algorithms typically use the RSA algorithm, in which an originator establishes a shared cryptographic key with a recipient by generating that key and then transporting it to the recipient. This is in contrast to key agreement algorithms, in which the two parties that will be using a cryptographic key both take part in its generation, thereby mutually agreeing to that key.

5.197.2 Properties

5.197.2.1 RecipientIdentifier

```
property RecipientIdentifier: TScCMSSubjectIdentifier;
```

Description

The **RecipientIdentifier** property retrieves the recipient identifier associated with the encrypted content.

RecipientIdentifier specifies the recipient's certificate or key that was used by the sender to protect the content-encryption key. The content-encryption key is encrypted with the recipient's public key.

This property is read-only.

See Also

Init

5.197.3 Methods

5.197.3.1 Init

```
procedure Init(Recipient: TScCMSRecipient); overload;
procedure Init(Recipient: TScCMSRecipient; const EncryptedKey: TBytes);
overload;
```

Description

Initializes the **TScCMSKeyTransRecipientInfo** instance by using the specified recipient information and encrypted key.

The `Recipient` parameter is an object that represents the recipient information.

The `EncryptedKey` parameter represents the encrypted key for this key transport recipient. The EncryptedKey property is set from the value of this parameter.

See Also

EncryptedKey

RecipientIdentifier

5.198 TScCMSKeyAgreeRecipientInfo

5.198.1 Description

Unit

ScCMS

Description

The **TScCMSKeyAgreeRecipientInfo** class represents key agreement recipient information, described in RFC 5652.

Key agreement algorithms typically use the Diffie-Hellman key agreement algorithm, in which the two parties that establish a shared cryptographic key both take part in its generation and, by definition, agree on that key. This is in contrast to key transport algorithms, in which one party generates the key unilaterally and sends, or transports it, to the other party.

5.198.2 Properties

5.198.2.1 OriginatorIdentifier

```
property OriginatorIdentifier: TScCMSOriginatorIdentifierOrKey;
```

Description

The **OriginatorIdentifier** property retrieves the object that contains information about the originator of the key agreement for key agreement algorithms that warrant it.

The sender uses the corresponding private key and the recipient's public key to generate a pairwise key. The content-encryption key is encrypted in the pairwise key.

This property is read-only.

5.198.2.2 RecipientIdentifier

```
property RecipientIdentifier: TScCMSSubjectIdentifier;
```

Description

The **RecipientIdentifier** property retrieves the identifier of the recipient's certificate, and thereby the recipient's public key, that was used by the sender to generate a pairwise key-encryption key.

This property is read-only.

5.198.2.3 UserKeyingMaterial

```
property UserKeyingMaterial: TBytes;
```

Description

The **UserKeyingMaterial** property retrieves the User Keying Material (UKM). The sender can provide a UKM to ensure that a different key is generated each time the same two parties generate a pairwise key.

This property is read-only.

5.199 TScCMSKEKRecipientInfo

5.199.1 Description

Unit

ScCMS

Description

The **TScCMSKEKRecipientInfo** class represents KEK recipient information, described in RFC 5652.

KEK algorithms use previously distributed symmetric keys. Each instance of KEK recipient transfers the content-encryption key to one or more recipients who have the previously distributed key-encryption key.

5.199.2 Properties

5.199.2.1 Date

```
property Date: TDateTime;
```

Description

The **Date** property retrieves the date and time that specifies a single key-encryption key from a set that was previously distributed.

This property is read-only.

5.199.2.2 KeyIdentifier

```
property KeyIdentifier: TBytes;
```

Description

The **KeyIdentifier** property retrieves the identifier of the key-encryption key that was previously distributed to the sender.

This property is read-only.

5.200 TScCMSPasswordRecipientInfo

5.200.1 Description

Unit

ScCMS

Description

The **TScCMSPasswordRecipientInfo** class represents password recipient information, described in RFC 5652.

Password algorithms use a password or shared secret value. Each instance of password recipient transfers the content-encryption key to one or more recipients who possess the password or shared secret value.

5.200.2 Properties

5.200.2.1 KeyDerivationAlgorithmIdentifier

```
property KeyDerivationAlgorithmIdentifier: TScASN1AlgorithmIdentifier;
```

Description

The **KeyDerivationAlgorithmIdentifier** property retrieves the key-derivation algorithm, and any associated parameters, used to derive the key-encryption key from the password or shared secret value.

If this property is empty, the key-encryption key is supplied from other external source.

This property is read-only.

5.201 TScCMSRecipientInfos

5.201.1 Description

Unit

ScCMS

Description

TScCMSRecipientInfos maintains a list of the TScCMSRecipientInfo objects.

Use **TScCMSRecipientInfos** to store and maintain a list of objects. **TScCMSRecipientInfos** provides properties and methods to add, delete, locate, and access objects. **TScCMSRecipientInfos** controls the memory of its objects, freeing an object when its index is reassigned; when it is removed from the list with the Delete, Remove, or Clear method; or when the **TScCMSRecipientInfos** instance is itself destroyed.

See also

TScCMSRecipientInfo

5.201.2 Properties

5.201.2.1 RecipientInfos

```
property RecipientInfos[Index: integer]: TScCMSRecipientInfo; default;
```

Description

Lists the TScCMSRecipientInfo object references.

Use **RecipientInfos** to access objects in the list. **RecipientInfos** is a zero-based array: the first object is indexed as 0, the second object is indexed as 1, and so on. You can read or change the value at a specific index, or use **RecipientInfos** with the Count property to iterate through the list.

Reassigning a **RecipientInfos** index frees the object that previously occupied that position in the list.

Note: Signatures is the default property of TScCMSRecipientInfos. This means you can omit the property name.

See also

Count

TScCMSRecipientInfo

5.202 TScCMSEnvelopedData

5.202.1 Description

Unit

ScCMS

Description

The **TScCMSEnvelopedData** class represents a CMS/PKCS #7 structure for enveloped data, described in RFC 5652.

TScCMSEnvelopedData stores the required information and enables encrypting and decrypting of CMS/PKCS #7 messages.

To encrypt content of any type, in the beginning initialize the **TScCMSEnvelopedData** object with the required content using the Init method. After this, for each recipient, create and initialize the TScCMSRecipient object specifying a recipient's certificate with a public key, and encrypt the information using the Encrypt method. And finally, call the Encode method to get an encoded CMS message.

To decrypt CMS/PKCS #7 message, in the beginning, decode the message using the Decode method. This method sets all properties of the **TScCMSEnvelopedData** object by using the decoded message and a user can retrieve the required properties. The inner contents of the decoded message can be retrieved from the ContentInfo property, but the information will be encrypted. To decrypt the content data use the Decrypt method specifying the recipient's certificate with a private key.

See Also

Init

Decrypt

Encrypt

5.202.2 Properties

5.202.2.1 ContentEncryptionAlgorithm

```
property ContentEncryptionAlgorithm: TScASN1AlgorithmIdentifier;
```

Description

The **ContentEncryptionAlgorithm** property retrieves the identifier of the symmetric algorithm used to encrypt the content.

The default value is the OID_DES_EDE3_CBC (1.2.840.113549.3.7) algorithm identifier.

This property is read-only.

ContentEncryptionAlgorithm is set as a result of calling the Decode and Init methods.

See Also

Decrypt

Encrypt

Init

5.202.2.2 ContentInfo

```
property ContentInfo: TScCMSContentInfo;
```

Description

The **ContentInfo** property retrieves the inner content information for the enveloped CMS/PKCS #7 message.

This property is read-only.

ContentInfo is set as a result of calling the Decode and Init methods.

See Also

Decrypt

Encrypt

Init

5.202.2.3 OriginatorCertificates

```
property OriginatorCertificates: TScCertificateList;
```

Description

The **OriginatorCertificates** property represents the list of certificates associated with the enveloped CMS/PKCS #7 message.

See Also

Decrypt

Encrypt

5.202.2.4 RecipientInfos

```
property RecipientInfos: TScCMSRecipientInfos;
```

Description

The **RecipientInfos** property retrieves the TScCMSRecipientInfos list of recipients information associated with the enveloped CMS/PKCS #7 message.

This property is read-only.

The **RecipientInfos** list is populated as a result of calling the Decode and Encrypt methods.

See Also

Decode

Encrypt

5.202.2.5 UnprotectedAttributes

```
property UnprotectedAttributes: TScCMSUnsignedAttributes;
```

Description

The **UnprotectedAttributes** property retrieves the unprotected (unencrypted) attributes associated with the enveloped CMS/PKCS #7 message. Unprotected attributes are not encrypted, and so do not have data confidentiality within a TScCMSEnvelopedData object.

See Also

Decode

5.202.3 Methods

5.202.3.1 Decode

```
procedure Decode(const RawData: TBytes); overload;
procedure Decode(Stream: TStream); overload;
```

Description

The **Decode** method decodes a specified enveloped CMS/PKCS #7 message. Upon successful decoding, the decoded information can be retrieved from the properties of the TScCMSEnvelopedData object.

`RawData` is an array of byte values, and `Stream` is a TStream object, that represent the encoded CMS/PKCS #7 message to be decoded.

This method resets all properties of the object by using the information obtained from successful decoding.

See Also

Encode

5.202.3.2 Decrypt

```
function Decrypt(Certificate: TScCertificate): TBytes; overload;
procedure Decrypt(Certificate: TScCertificate; OutStream: TStream);
overload;
```

Description

The **Decrypt** method decrypts the contents of the decoded enveloped CMS/PKCS #7 message by using the certificate with a private key specified in the `Certificate` parameter.

The method finds the recipient information corresponding to the specified certificate and decrypts the content information setting in the ContentInfo property.

The decrypted data can be returned as an array of byte values or can be written to the `OutStream` parameter.

See Also

Encrypt

5.202.3.3 Encode

```
function Encode: TBytes; overload;
procedure Encode(Stream: TStream); overload;
```

Description

The **Encode** method encodes the contents of the object into an enveloped CMS/PKCS #7 message. Encryption must be done before encoding.

This method can return an array of byte values that represents the encoded message or can write this result to the `Stream` parameter.

The encoded message can be decoded by the Decode method.

See Also

Decode

5.202.3.4 Encrypt

```
procedure Encrypt(Recipient: TScCMSRecipient);
```

Description

The **Encrypt** method encrypts the contents of the CMS/PKCS #7 message by using the specified recipient information.

The `Recipient` parameter is an object that represents the recipient.

The recipient information is added to the RecipientInfos list.

See Also

Decrypt

5.202.3.5 Init

```
procedure Init(ContentInfo: TScCMSContentInfo; EncryptionAlgorithm:
TScASN1AlgorithmIdentifier); overload;
procedure Init(ContentInfo: TScCMSContentInfo; EncryptionAlgorithm:
TScSymmetricAlgorithm = saTripleDES_cbc); overload;
procedure Init(const ContentBuffer: TBytes; EncryptionAlgorithm:
TScSymmetricAlgorithm = saTripleDES_cbc); overload;
procedure Init(ContentStream: TStream; EncryptionAlgorithm:
TScSymmetricAlgorithm = saTripleDES_cbc); overload;
```

Description

Initializes the **TScCMSEnvelopedData** instance by using the specified content information as the inner content.

The `ContentInfo`, `ContentBuffer`, and `ContentStream` parameters represent the content information as the inner content of the encrypted message. The ContentInfo property is set from the value of this parameter.

The `EncryptionAlgorithm` parameter represents the symmetric algorithm used to encrypt the content. The ContentEncryptionAlgorithm property is set from the value of this parameter.

The **Init** method clears the RecipientInfos list.

See Also

ContentEncryptionAlgorithm

ContentInfo

RecipientInfos

5.203 TScCMSProcessor

5.203.1 Description

Unit

ScCMS

Description

The **TScCMSProcessor** class provides a simple interface to encrypt, decrypt, sign, and verify content of any type and store them in CMS/PKCS #7 format. CMS is a common format to store encrypted and signed data, described in RFC 5652.

Before any operation the Certificate or CertificateName property should be set. The specified certificate will be used to encrypt, decrypt, sign, or verify data.

After this to encrypt data just call the Encrypt method, specifying the encrypted data as an input parameter of the method.

To decrypt data call the Decrypt method, specifying the enveloped CMS/PKCS #7 message as an input parameter of the method.

The EnvelopedData object will store the information about the previous encrypted or decrypted enveloped CMS/PKCS #7 message.

To sing data just call the Sign method, specifying the input data as an input parameter of the method.

To verify the digital signature of the data call the CheckSignature method, specifying the signed CMS/PKCS #7 message as an input parameter of the method.

The SignedData object will store the information about the previous signed or verified CMS/PKCS #7 message.

5.203.2 Properties

5.203.2.1 Certificate

```
property Certificate: TScCertificate;
```

Description

The **Certificate** property represents the certificate associated with the CMS/PKCS #7 message. This certificate is used to encrypt, decrypt, sign, or verify data.

This property is related to the CertificateName property. If the **Certificate** property is nil, the CertificateName property is used instead.

If the **Certificate** property is nil and the CertificateName property is empty, an exception will be raised on processing a CMS message.

See also

CertificateName

5.203.2.2 CertificateName

```
property CertificateName: string;
```

Description

The **CertificateName** property represents the name of the certificate associated with the CMS/PKCS #7 message. Specified certificate is stored in the Storage. This certificate is used to encrypt, decrypt, sign, or verify data.

This property is related to the Certificate property. If the Certificate property is not nil, this property is ignored and the Certificate property is used instead.

If the Certificate property is nil and the **CertificateName** property is empty, an exception will be raised on processing a CMS message.

See also

Certificate

Storage

5.203.2.3 DigestAlgorithm

```
property DigestAlgorithm: TScHashAlgorithm; default haSHA2_256;
```

Description

The **DigestAlgorithm** property contains the hash algorithm used in the computation of the signatures.

The default value is the haSHA2_256 algorithm.

5.203.2.4 EncryptionAlgorithm

```
property EncryptionAlgorithm: TScSymmetricAlgorithm; default
saAES192_cbc;
```

Description

The **EncryptionAlgorithm** property contains the symmetric algorithm used to encrypt the data.

The default value is the saAES192_cbc algorithm.

5.203.2.5 EnvelopedData

```
property EnvelopedData: TScCMSEnvelopedData;
```

Description

The **EnvelopedData** property contains an object that stores the information about the previous encrypted or decrypted enveloped CMS/PKCS #7 message.

This property is read-only.

See Also

SignedData

5.203.2.6 SignedData

```
property SignedData: TScCMSSignedData;
```

Description

The **SignedData** property contains an object that stores the information about the previous signed or verified CMS/PKCS #7 message.

This property is read-only.

See Also

EnvelopedData

5.203.2.7 Storage

```
property Storage: TScStorage;
```

Description

The **Storage** property is used to access certificate list in the linked storage. If **Storage** is not assigned and the Certificate property is nil, an exception will be raised on processing a CMS message.

See Also

CertificateName

5.203.3 Methods

5.203.3.1 CheckSignature

```
procedure CheckSignature(const Data: TBytes); overload;
procedure CheckSignature(InStream: TStream; TmpDecodedStream: TStream =
nil); overload;
procedure CheckSignature(const FileName: string; const TmpFileName:
string = ''); overload;
```

Description

The **CheckSignature** method verifies the digital signature on the signed CMS/PKCS #7 message by using the certificate specified in the Certificate property. The method finds the signature corresponding to the specified certificate and verifies it. If there are signed attributes included with the

message, these attributes are also verified.

`Data` is an array of byte values that represents the CMS/PKCS #7 message to be verified.

`InStream` is a TStream object that represents the CMS/PKCS #7 message to be verified.

`FileName` is a name of the file that contains the CMS/PKCS #7 message to be verified.

`TmpDecodedStream` is a TStream object that will contain temporary data in case if source a format of CMS/PKCS #7 message is the PEM or S/MIME format. If the `TmpDecodedStream` parameter is nil, the TMemoryStream object will be created instead.

`TmpFileName` is a name of the file that will contain temporary data in case if a source format of CMS/PKCS #7 message is the PEM or S/MIME format. If the `TmpFileName` parameter is empty, the TMemoryStream object will be created for temporary data.

CheckSignature raises an exception if the verification of a digital signature fails.

CheckSignature resets all properties of the SignedData object that stores the information about the processed CMS/PKCS #7 message.

See Also

Sign

SignedData

5.203.3.2 DecodeData

```
class function DecodeData(const Data: TBytes): TBytes; overload;
class procedure DecodeData(InStream, OutStream: TStream); overload;
```

Description

The **DecodeData** method decodes a CMS/PKCS #7 message encoded in the PEM or S/MIME format to the DER format. The result data can be passed to other methods of CMS message processing, like Decrypt, CheckSignature, and DecryptAndCheckSignature.

The encoded CMS/PKCS #7 message that will be decoded can be passed by the `Data` parameter as a byte array or by the `InStream` parameter as a TStream object.

The decoded data can be returned as an array of byte values or can be written to the `OutStream` parameter.

See Also

EncodeData

5.203.3.3 Decrypt

```
function Decrypt(const Data: TBytes): TBytes; overload;
```

```
procedure Decrypt(InStream, OutStream: TStream); overload;
procedure Decrypt(const InFileName, OutFileName: string); overload;
```

Description

The **Decrypt** method decrypts the contents of the enveloped CMS/PKCS #7 message by using the certificate with a private key specified in the Certificate property. The method finds the recipient information corresponding to the specified certificate and decrypts the content information.

The decrypted data can be returned as an array of byte values or can be written to the `OutStream` stream or to the `OutFileName` file.

`Data` is an array of byte values that represents the CMS/PKCS #7 message to be decrypted.

`InStream` is a TStream object that represents the CMS/PKCS #7 message to be decrypted.

`InFileName` is a name of the file that contains the CMS/PKCS #7 message to be decrypted.

`OutStream` is a TStream object that will contain the decrypted content information.

`OutFileName` is a name of the file that will contain the decrypted content information.

Decrypt resets all properties of the EnvelopedData object that stores the information about the processed CMS/PKCS #7 message.

See Also

Encrypt

EnvelopedData

5.203.3.4 DecryptAndCheckSignature

```
function DecryptAndCheckSignature(const Data: TBytes): TBytes; overload;
procedure DecryptAndCheckSignature(InStream, OutStream: TStream;
TmpDecryptedStream: TStream = nil); overload;
procedure DecryptAndCheckSignature(const InFileName, OutFileName: string;
const TmpFileName: string = ''); overload;
```

Description

The **DecryptAndCheckSignature** method decrypts the contents of the enveloped CMS/PKCS #7 message and verifies it the digital signature by using the certificate with a private key specified in the Certificate property. The method finds the recipient information corresponding to the specified certificate and decrypts the content information. After this the method finds the signature corresponding to the specified certificate and verifies it.

The decrypted data can be returned as an array of byte values or can be written to the `OutStream` stream or to the `OutFileName` file.

`Data` is an array of byte values that represents the CMS/PKCS #7 message to be decrypted and verified.

`InStream` is a TStream object that represents the CMS/PKCS #7 message to be decrypted and verified.

`InFileName` is a name of the file that contains the CMS/PKCS #7 message to be decrypted and verified.

`OutStream` is a TStream object that will contain the decrypted content information.

`OutFileName` is a name of the file that will contain the decrypted content information.

`TmpDecryptedStream` is a TStream object that will contain temporary data. If the `TmpDecryptedStream` parameter is nil, the TMemoryStream object will be created instead.

`TmpFileName` is a name of the file that will contain temporary data. If the `TmpFileName` parameter is empty, the TMemoryStream object will be created for temporary data.

DecryptAndCheckSignature raises an exception if the verification of a digital signature fails.

DecryptAndCheckSignature resets all properties of the SignedData and the EnvelopedData objects that store the information about the processed CMS/PKCS #7 message.

See Also

EnvelopedData

SignedData

SignAndEncrypt

5.203.3.5 EncodeData

```
class function EncodeData(const Data: TBytes; CMSEncoding:
TScCMSEncoding): TBytes; overload;
class procedure EncodeData(InStream, OutStream: TStream; CMSEncoding:
TScCMSEncoding); overload;
```

Description

The **EncodeData** method encodes a CMS/PKCS #7 message from the DER format to the PEM or S/MIME format.

The `CMSEncoding` parameter specifies the output encoded format.

The CMS/PKCS #7 message that will be encoded can be passed by the `Data` parameter as a byte array or by the `InStream` parameter as a TStream object.

The encoded data can be returned as an array of byte values or can be written to the `OutStream` parameter.

See Also

DecodeData

5.203.3.6 Encrypt

```
function Encrypt(const Data: TBytes; Encoding: TScCMSEncoding = ceDER):
TBytes; overload;
procedure Encrypt(InStream, OutStream: TStream; Encoding: TScCMSEncoding
= ceDER); overload;
procedure Encrypt(const InFileName, OutFileName: string; Encoding:
TScCMSEncoding = ceDER); overload;
```

Description

The **Encrypt** method encrypts the input data by using the certificate specified in the Certificate property, and encodes the result information into an enveloped CMS/PKCS #7 message.

This method can return an array of byte values that represents the encoded message or can write this result to the `OutStream` stream or to the `OutFileName` file.

`Data` is an array of byte values that represents the input data to be encrypted.

`InStream` is a TStream object that contains the input data to be encrypted.

`InFileName` is a name of the file that contains the input data to be encrypted.

`OutStream` is a TStream object that will contain the enveloped CMS/PKCS #7 message.

`OutFileName` is a name of the file that will contain the enveloped CMS/PKCS #7 message.

`Encoding` specifies the output encoded format.

The symmetric encryption algorithm can be specified by the EncryptionAlgorithm property.

Encrypt resets all properties of the EnvelopedData object that stores the information about the processed CMS/PKCS #7 message.

See Also

Decrypt

EncryptionAlgorithm

EnvelopedData

5.203.3.7 Sign

```
function Sign(const Data: TBytes; IncludeContent: boolean = False;
Encoding: TScCMSEncoding = ceDER): TBytes; overload;
procedure Sign(InStream, OutStream: TStream; IncludeContent: boolean =
False; Encoding: TScCMSEncoding = ceDER); overload;
procedure Sign(const InFileName, OutFileName: string; IncludeContent:
boolean = False; Encoding: TScCMSEncoding = ceDER); overload;
```

Description

The **Sign** method creates a signature of the input data by using the certificate specified in the Certificate property, and encodes the result information into a signed CMS/PKCS #7 message.

This method can return an array of byte values that represents the encoded message or can write this result to the `OutStream` stream or to the `OutFileName` file.

`Data` is an array of byte values that represents the input data to be signed.

`InStream` is a TStream object that contains the input data to be signed.

`InFileName` is a name of the file that contains the input data to be signed.

`OutStream` is a TStream object that will contain the signed CMS/PKCS #7 message.

`OutFileName` is a name of the file that will contain the signed CMS/PKCS #7 message.

`Encoding` specifies the output encoded format.

`IncludeContent` is a boolean value that specifies whether the signed content is included in the CMS/PKCS #7 message. If `IncludeContent` is True, the signed content is included in the CMS/PKCS #7 message along with the signature information. If the `IncludeContent` state is False (by default), the message does not contain the signed content, and a client can see the content of the message only if it is sent separately.

The hash algorithm can be specified by the DigestAlgorithm property.

Sign resets all properties of the SignedData object that stores the information about the processed CMS/PKCS #7 message.

See Also

CheckSignature

DigestAlgorithm

SignedData

5.203.3.8 SignAndEncrypt

```
function SignAndEncrypt(const Data: TBytes; Encoding: TScCMSEncoding =
ceDER): TBytes; overload;
procedure SignAndEncrypt(InStream, OutStream: TStream; TmpSignedStream:
TStream = nil; Encoding: TScCMSEncoding = ceDER); overload;
procedure SignAndEncrypt(const InFileName, OutFileName: string; const
TmpFileName: string = ''; Encoding: TScCMSEncoding = ceDER); overload;
```

Description

The **SignAndEncrypt** method creates a signature of the input data and encrypts this data by using the certificate specified in the Certificate property, and encodes the result information into an enveloped CMS/PKCS #7 message.

This method can return an array of byte values that represents the encoded message or can write this result to the `OutStream` stream or to the `OutFileName` file.

`Data` is an array of byte values that represents the input data to be signed and encrypted.

`InStream` is a TStream object that contains the input data to be signed and encrypted.

`InFileName` is a name of the file that contains the input data to be signed and encrypted.

`OutStream` is a TStream object that will contain the enveloped CMS/PKCS #7 message.

`OutFileName` is a name of the file that will contain the enveloped CMS/PKCS #7 message.

`TmpSignedStream` is a TStream object that will contain temporary data. If the `TmpSignedStream` parameter is nil, the TMemoryStream object will be created instead.

`TmpFileName` is a name of the file that will contain temporary data. If the `TmpFileName` parameter is empty, the TMemoryStream object will be created for temporary data.

`Encoding` specifies the output encoded format.

The symmetric encryption algorithm can be specified by the EncryptionAlgorithm property.

The hash algorithm can be specified by the DigestAlgorithm property.

SignAndEncrypt resets all properties of the SignedData and the EnvelopedData objects that store the information about the processed CMS/PKCS #7 message.

See Also

DecryptAndCheckSignature

DigestAlgorithm

EncryptionAlgorithm

EnvelopedData

SignedData

5.204 TScStreamInfo

5.204.1 Description

Unit

ScUtils

Description

The **TScStreamInfo** class is wrapper for TStream object that stores encapsulated data.

Properties of **TScStreamInfo** provide information about the stream, count of encapsulated data and offset into the stream for reading.

See Also

TScCMSContentInfo.ContentStream

5.204.2 Properties

5.204.2.1 Count

```
property Count: Int64;
```

Description

The `Count` property indicates the count in bytes of the encapsulated data.

This property is read-only.

See also

Init

5.204.2.2 Position

```
property Position: Int64;
```

Description

The `Position` property indicates the offset into the Stream for reading the encapsulated data.

This property is read-only.

See also

Init

5.204.2.3 Stream

```
property Stream: TStream;
```

Description

The `Stream` property is a reference to a TStream object that stores encapsulated data.

This property is read-only.

See also

Init

5.204.3 Methods

5.204.3.1 Assign

```
procedure Assign(Source: TScStreamInfo);
```

Description

Copies the contents of another similar object. **Assign** copies properties of the specified `Source` object to the current object.

5.204.3.2 Create

```
constructor Create(Stream: TStream; Position, Count: Int64);
```

Description

Create **TScStreamInfo** instance.

The `Stream` parameter is a reference to a TStream object that stores encapsulated data. The Stream property is set from the value of this parameter.

The `Position` parameter represents the offset into the `Stream` for reading the encapsulated data. The Position property is set from the value of this parameter.

The `Count` parameter represents the count in bytes of the encapsulated data. The Count property is set from the value of this parameter.

See also

Init

5.204.3.3 Init

```
procedure Init(Stream: TStream; Position, Count: Int64);
```

Description

Initializes the **TScStreamInfo** instance from the specified TStream object.

The `Stream` parameter is a reference to a TStream object that stores encapsulated data. The Stream property is set from the value of this parameter.

The `Position` parameter represents the offset into the `Stream` for reading the encapsulated data. The Position property is set from the value of this parameter.

The `Count` parameter represents the count in bytes of the encapsulated data. The Count property is set from the value of this parameter.

5.205 TScTerminalInfo

5.205.1 Description

Unit

ScSSHUtils

Description

The **TScTerminalInfo** class represents information about pseudo-terminal, which is created on the server side for correct displaying results of the command execution via TScSSHShell.

See also

TScSSHShell.TerminalInfo

5.205.2 Properties

5.205.2.1 Cols

```
property Cols: Integer; default 80;
```

Description

The width of the terminal window in characters. The **Cols** dimension override the Width dimensions when nonzero.

Default value is 80.

5.205.2.2 Rows

```
property Rows: Integer; default 25;
```

Description

The height of the terminal window in characters. The **Rows** dimensions override the Height dimensions when nonzero.

Default value is 25.

5.205.2.3 Height

```
property Height: Integer; default 480;
```

Description

The height of the terminal window in pixels.

Default value is 480.

5.205.2.4 Width

```
property Width: Integer; default 640;
```

Description

The width of the terminal window in pixels.

Default value is 640.

5.205.2.5 TerminalType

```
property TerminalType: string;
```

Description

The TERM environment variable value, that represents a terminal type. Default value is 'vt100', which is provides a text terminal.

6 SecureBridge Object and Component Listing by Unit

6.1 ScBridge

6.1.1 Classes

ScBridge unit implements the following classes:

TScDHData
TScDSAData
TScECCryptography
TScECData
TScECPoint
TScECParameters
TScRSAData

TScCertificate
TScCertificateList
TScCertificateStatus
TScCRL
TScKey
TScKeyList
TScKeyFormat
TScFileStorage
TScMemoryStorage
TScOAEPParams
TScPSSParams
TScRegStorage
TScStorage

TScStorageItem
TScStorageList
TScUser
TScUserAuthentication
TScUserAuthentications
TScUserList

6.2 ScCertificateExts

6.2.1 Classes

ScCertificateExts unit implements the following classes:

TScASN1AlgorithmIdentifier
TScASN1AlgorithmIdentifiers
TScASN1Attribute
TScASN1Attributes
TScDistinguishedName
TScDistinguishedNameList
TScGeneralName
TScGeneralNames
TScKeyUsageFlag
TScOId
TScOIds
TScPaddingMode
TScPKCS7Attribute
TScPKCS7Attributes
TScPolicy
TScPolicyList
TScPolicyMapping
TScPolicyMappingList
TScQualifier
TScRelativeDistinguishedName
TScSignatureAlgorithmIdentifier

TScCertAlternativeNameExtension
TScCertAuthorityInfoAccessExtension
TScCertAuthorityKeyIdExtension

TScCertBasicConstraintsExtension
TScCertCRLDistributionPointsExtension
TScCertExtendedKeyUsageExtension
TScCertificateExtension
TScCertKeyUsageExtension
TScCertPoliciesExtension
TScCertPolicyMappingsExtension
TScCertSubjectDirectoryAttributesExtension
TScCertSubjectInfoAccessExtension
TScCertSubjectKeyIdExtension
TScCRLCertificateIssuerExtension
TScCRLDeltaIndicatorExtension
TScCRLInvalidityDateExtension
TScCRLIssuingDistributionPointExtension
TScCRLNumberExtension
TScCRLReasonCodeExtension
TScExtensions

6.3 ScCMS

6.3.1 Classes

ScCMS unit implements the following classes:

TScCMSContentInfo
TScCMSData
TScCMSEncoding
TScCMSEnvelopedData
TScCMSIncludedAttribute
TScCMSIncludedAttributes
TScCMSKEKRecipientInfo
TScCMSKeyAgreeRecipientInfo
TScCMSKeyTransRecipientInfo
TScCMSOriginatorIdentifierOrKey
TScCMSOriginatorIdentifierOrKeyType
TScCMSPasswordRecipientInfo
TScCMSProcessor
TScCMSRecipient

TScCMSRecipientInfo
TScCMSRecipientInfos
TScCMSRecipientInfoType
TScCMSSignature
TScCMSSignatures
TScCMSSignedAttributes
TScCMSSignedData
TScCMSSignerInfo
TScCMSSMIMEAttributes
TScCMSSubjectIdentifier
TScCMSSubjectIdentifierType
TScCMSUnsignedAttributes

6.4 ScCryptoAPIStorage

6.4.1 Classes

ScCryptoAPIStorage unit implements the following classes:

TScCryptoAPIStorage

6.5 ScFTPClient

6.5.1 Classes

ScFTPClient unit implements the following classes:

EScFTPError
TScFTPAuthCommand
TScFTPClient
TScFTPClientOptions
TScFTPFileStructure
TScFTPTransferType

6.6 ScFTPListParser

6.6.1 Classes

ScFTPListParser unit implements the following classes:

TScFTPDirectoryListing
TScFTPFileType
TScFTPListItem
TScFTPListParser

6.7 ScHttp

6.7.1 Classes

ScHttp unit implements the following classes:

HttpException
TScHttpStatusCode
TScHttpWebRequest
TScHttpWebResponse
TScRequestCacheLevel
TScRequestCachePolicy
TScRequestMethod
TScWebHeaderCollection
TScWebRequestHeaderCollection
TScWebResponseHeaderCollection

6.8 ScIndy

6.8.1 Classes

ScIndy unit implements the following classes:

TScIdIOHandler

6.9 ScMailMessage

6.9.1 Classes

ScMailMessage unit implements the following classes:

TScMailAddressItem
TScMailAddressList
TScMailEncoding
TScMailMessage
TScMailPriority
TScMimeBoundary

6.10 ScRNG

6.10.1 Classes

ScRNG unit implements the following classes:

TScRandom
TScRandom_LFSR

6.11 ScSecureConnection

6.11.1 Classes

ScSecureConnection unit implements the following classes:

TScNetworkCredential
TScVersion
TScWebProxy

6.12 ScSFTPClient

6.12.1 Classes

ScSFTPClient unit implements the following classes:

TScSFTPClient
TScSFTPOperation
TScSFTPServerProperties

6.13 ScSFTPConsts

6.13.1 Classes

ScSFTPConsts unit implements the following classes:

TScSFTPErrorCode

6.14 ScSFTPServer

6.14.1 Classes

ScSFTPServer unit implements the following classes:

TScHandle
TScSearchRec
TScSFTPServer

6.15 ScSFTPUtils

6.15.1 Classes

ScSFTPUtils unit implements the following classes:

ConvertFilePermissionsToSFTPValue
EScSFTPError
TScCheckFileReplyExtension
TScFilenameTranslationControlExtension

TScSFTPACEItem
TScSFTPAceMask
TScSFTPAceMaskItem
TScSFTPACEs

TScSFTPAttribute
TScSFTPAttributes
TScSFTPBlockMode
TScSFTPBlockModes
TScSFTPCustomExtension
TScSFTPDesiredAccess
TScSFTPDesiredAccessItem
TScSFTPError
TScSFTPExtension
TScSFTPFileAttributes
TScSFTPFileInfo
TScSFTPFileOpenAttributes
TScSFTPFileOpenFlag
TScSFTPFileOpenFlags
TScSFTPFileOpenMode
TScSFTPFileOpenModeItem
TScSFTPFileOpenModes
TScSFTPFileType
TScSFTPRealpathControl
TScSFTPRenameFlag
TScSFTPRenameFlags
TScSFTPSupportedAclExtension
TScSFTPSupportedExtension
TScSFTPVendorExtension
TScSFTPVersion
TScSFTPVersions
TScSFTPVersionsExtension
TScSpaceAvailableReplyExtension

6.16 ScSignalRHttpConnection

6.16.1 Classes

ScSignalRHttpConnection unit implements the following classes:

TScHttpConnectionOptions
TScHttpTransportType
TScHttpTransportTypes

6.17 ScSignalRHubConnection

6.17.1 Classes

ScSignalRHubConnection unit implements the following classes:

TScHubConnection
TScHubConnectionState
TScRetryContext
TScRetryPolicy

6.18 ScSignalRProtocol

6.18.1 Classes

ScSignalRProtocol unit implements the following classes:

HubException

6.19 ScSMTPClient

6.19.1 Classes

ScSMTPClient unit implements the following classes:

EScSMTPError

TScSASLAnonymous
TScSASLCollection
TScSASLCRAMMD5
TScSASLCRAMSHA1
TScSASLItem
TScSASLLogin
TScSASLMechanism

TScSASLOTP
TScSASLPlain
TScSASLSKey
TScSASLUserPassMechanism

TScSMTPAuthenticationType
TScSMTPClient
TScSMTPClientOptions

6.20 ScSMTPUtils

6.20.1 Classes

ScSMTPUtils unit implements the following classes:

TScAttachment
TScAttachmentCollection
TScCustomAttachment
TScHeaderList
TScTextAttachment

6.21 ScSSHChannel

6.21.1 Classes

ScSSHChannel unit implements the following classes:

TScSSHChannel
TScSSHCustomChannel
TScSSHShell
TScSSHStream

6.22 ScSSHClient

6.22.1 Classes

ScSSHClient unit implements the following classes:

TScSSHClient
TScSSHClientOptions
TScSSHCompression

6.23 ScSSHServer

6.23.1 Classes

ScSSHServer unit implements the following classes:

TScSSHServer
TScSSHServerOptions

6.24 ScSSHUtils

6.24.1 Classes

ScSSHUtils unit implements the following classes:

TScSFTPSessionInfo
TScSSHAuthentication
TScSSHChannelInfo
TScSSHCipherItem
TScSSHCiphers
TScSSHClientInfo
TScSSHConnectionInfo
TScSSHHMacAlgorithmItem
TScSSHHMacAlgorithms
TScSSHHostKeyAlgorithmItem
TScSSHHostKeyAlgorithms
TScSSHKeyExchangeAlgorithmItem
TScSSHKeyExchangeAlgorithms
TScTerminalInfo

6.25 ScSSLClient

6.25.1 Classes

ScSSLClient unit implements the following classes:

TScSSLClient
TScSSLClientOptions
TScSSLSecurityOptions

6.26 ScSSLExtensions

6.26.1 Classes

ScSSLExtensions unit implements the following classes:

TScECPointFormat
TScECPointFormats

TTLSApplicationLayerProtocolNegotiationExtension
TTLSEllipticCurvePointFormatsExtension
TTLSExtendedMasterSecretExtension
TTLSHelloExtension
TTLSHelloExtensions
TTLSRenegotiationIndicationExtension
TTLSServerNameExtension
TTLSSessionTicketExtension
TTLSSignatureAlgorithmsExtension
TTLSSupportedGroupsExtension

6.27 ScSSLServer

6.27.1 Classes

ScSSLServer unit implements the following classes:

TScSSLServerConnection
TScSSLServerOptions

6.28 ScSSLTypes

6.28.1 Classes

ScSSLTypes unit implements the following classes:

TScECurveDomainType
TScExtendedMasterSecretMode
TScSSLCipherAlgorithm
TScSSLCipherAlgorithmArray
TScSSLCipherSuiteItem
TScSSLCipherSuites
TScSSLProtocol
TScSSLProtocols
TScSSLSessionInfo
TScSSLSignatureAlgorithm

6.29 ScTCPServer

6.29.1 Classes

ScTCPServer unit implements the following classes:

TScTCPServer

6.30 ScUtils

6.30.1 Classes

ScUtils unit implements the following classes and types:

EScError
TScAsymmetricAlgorithm
TScAsymmetricAlgorithms
TScCollection
TScCollectionItem

TScCollectionItemClass
TScCompressionAlgorithm
TScCompressionAlgorithms
TScECName
TScHashAlgorithm
TScHashAlgorithms
TScHMACAlgorithm
TScHMACAlgorithms
TScKeyExchangeAlgorithm
TScKeyExchangeAlgorithms
TScLogger
TScPersistent
TScPersistentClass
TScPersistentObjectList
TScSignatureAlgorithm
TScStreamInfo
TScSymmetricAlgorithm
TScSymmetricAlgorithms
TScTCPConnection
TScTLSMode

6.31 ScVio

6.31.1 Classes

ScVio unit implements the following classes and types:

TCRSocksVersion
TIPVersion
THttpOptions
TProxyOptions

6.32 ScWebSocketClient

6.32.1 Classes

ScWebSocketClient unit implements the following classes:

TScHeartBeatOptions

TScWatchDogOptions
TScWebSocketClient
TScWebSocketClientOptions
TScWebSocketCloseStatus
TScWebSocketControlMessageType
TScWebSocketMessageType
TScWebSocketState
WebSocketException