SeeBeyond™ eBusiness Integration Suite

Secure Messaging Extension User's Guide

Release 4.5.3

Java Version

Version 20021113090324.

Contents

Preface

This Preface contains information regarding the Secure Messaging Extension User's Guide.

P.1 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the SeeBeyond™ e*Gate™ Integrator system, and have a working knowledge of:

- Windows NT/Windows 2000 and/or UNIX operations and administration
- Windows-style GUI operations
- Public Key Infrastructure (PKI)

P.2 Nomenclature

For purposes of brevity, Secure Messaging Extension is referred to as SME throughout the User's Guide.

P.3 Organization

This User's Guide is organized into two parts. The first part, consisting of Chapters 1-2, introduces SME and describes the procedures for installing, setting up, and implementing a working system incorporating SME. This part should be of particular interest to a System Administrator or other user charged with the task of getting the system up and running.

The second part, consisting of Chapters 3-6, describes the details of SME operation and configuration, including descriptions of the exposed Java methods. This part should be of particular interest to a Developer involved in customizing SME for a specific purpose. The information in this part that is required for the initial setup of SME is cross-referenced in the first part of the guide.

Chapter 1

Introduction

This document describes how to install and configure SME.

1.1 Overview

SME enables e*Gate to process Events utilizing the S/MIME (Secure Multipurpose Internet Mail Extensions) message format. This format is the IETF RFC 2311 specification for encrypting and/or signing types of data.

SME supports encryption, decryption and authentication of messages and is interoperable with any other client applications that support the S/MIME standard.

SME adds the following features to transactions:

- privacy
- message (Event) authentication
- sender authentication
- nonrepudiation

1.1.1 Components

The following components comprise SME:

- stcsme.jar
- smime.jar
- Java collaborations that load and run the Java methods

A complete list of the installed files appears in **Table 1 on page 16**.

1.2 Introducing Secure Messaging Extension (SME)

SME provides security features, allowing the protected transmission of exchanges over public domains such as the Internet. SME enables using Public Key Infrastructure (PKI) technology to ensure the confidentiality of exchanges. This is done by digitally signing and encrypting messages as they are sent, and decrypting and authenticating messages when they are received.

SME performs the encryption and decryption of messages using the S/MIME format. This format's one-way hash algorithms ensure data integrity by verifying that no modifications are made to the message while in transit. In addition, the message sender's identity is verified through the use of digital signatures, proving that the message actually originated from the entity who claims to have sent it.

The S/MIME format is described in detail in the following section.

1.3 Introducing Multipurpose Internet Mail Extension (MIME) and Secure Multipurpose Internet Mail Extension (S/MIME)

S/MIME is based on the Public Key Cryptography Standards (PKCS), which specify how the RSA public-key cryptographic algorithm should be used to implement enveloped encryption and digital signatures.

The RSA public-key system makes use of two related keys to perform the mathematical algorithms necessary to encrypt or decrypt data: a public key, which may be made available to any prospective correspondent, and a private key known only to the key's owner. A public key can be published openly, thereby assuring the ability of anyone to send secure messages that can only be decrypted by the owner of the respective private key.

Encryption can also be performed using one's private key, and decrypted with the corresponding public key. In this case, the encryption result is known as a digital signature, which guarantees to the intended recipient that the signed message is authentic and genuinely came from the stated originator of the message.

Digital signatures provide data integrity, authentication and non-repudiation of an electronic document. Successful verification of a digital signature ensures the recipient that the "document received" is identical to the "document sent" (data integrity) and confirms the identity of the sender (authentication). It also prevents any subsequent denial by the sender that the document originated with them (non-repudiation).

In practice, public keys are stored as certificates that comply with the X.509 standard. In addition to the public key, a certificate also contains information about the key owner's identity, the key's validity, and the issuer of the certificate, also known as a Certificate Authority.

MIME Message Format

MIME-compliant messages may contain any type of data, including the following:

- Text messages in US-ASCII
- Messages of unlimited length
- Binary files

- Character sets other than US-ASCII
- Multi-media: Image, Audio, and Video objects
- Multiple, nested objects in a single message

When later sent over a protocol such as HTTP or FTP, which provide a "binary clean" data path, MIME messages may be left in binary format. However, if the MIME message is sent via SMTP (email) or other text-only protocols, binary objects must be encoded using the Base64 content transfer encoding format, which produces a textual representation of the original binary data.

Messages in MIME format consist of two parts: the header and the body. The header forms a collection of metadata in the form of keyword/value pairs structured to provide information necessary for the transmission and interpretation of the message. The body of the message contains the bulk data to be transferred. In turn, S/MIME defines the security services, adding digital signatures and encryption, thus preventing forgery and interception.

For more information regarding MIME, see the Internet Engineering Task Force Text Messages specification (RFC 822) and the MIME Message Body Format (RFC 2045), at http://www.ietf.org.

The S/MIME Version 2 specification (RFC 2311) is also found at http://www.ietf.org.

1.4 Secure Messaging Extension Process

This section illustrates the internal and external flow of the SME's S/MIME processing methods. On the following pages, Figure 1 shows the processing of an inbound signed and/or encrypted message; Figure 2 shows the processing of the data for an outbound signed and/or encrypted message.

Figure 1 Inbound Signed/Encrypted Message

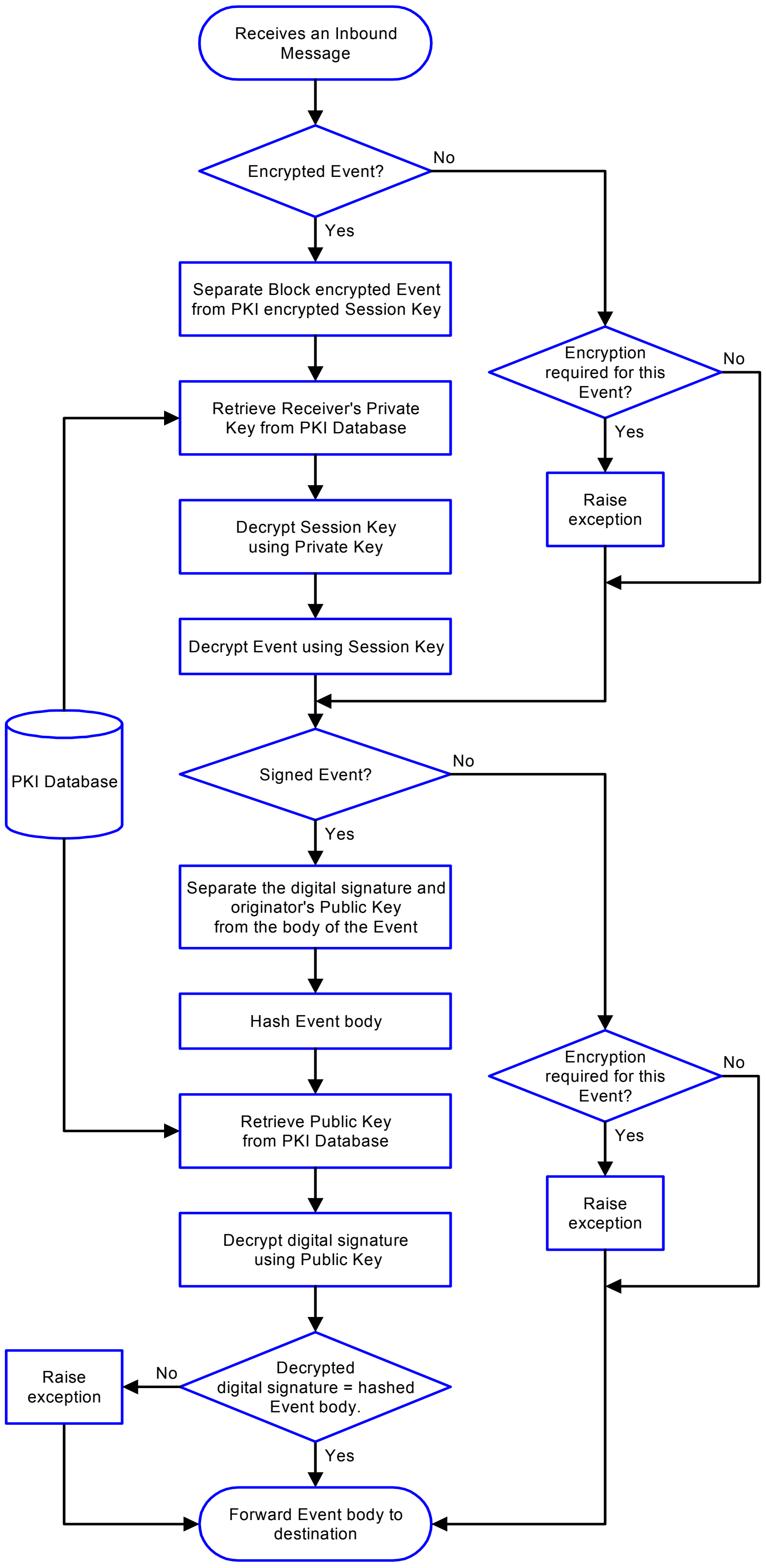
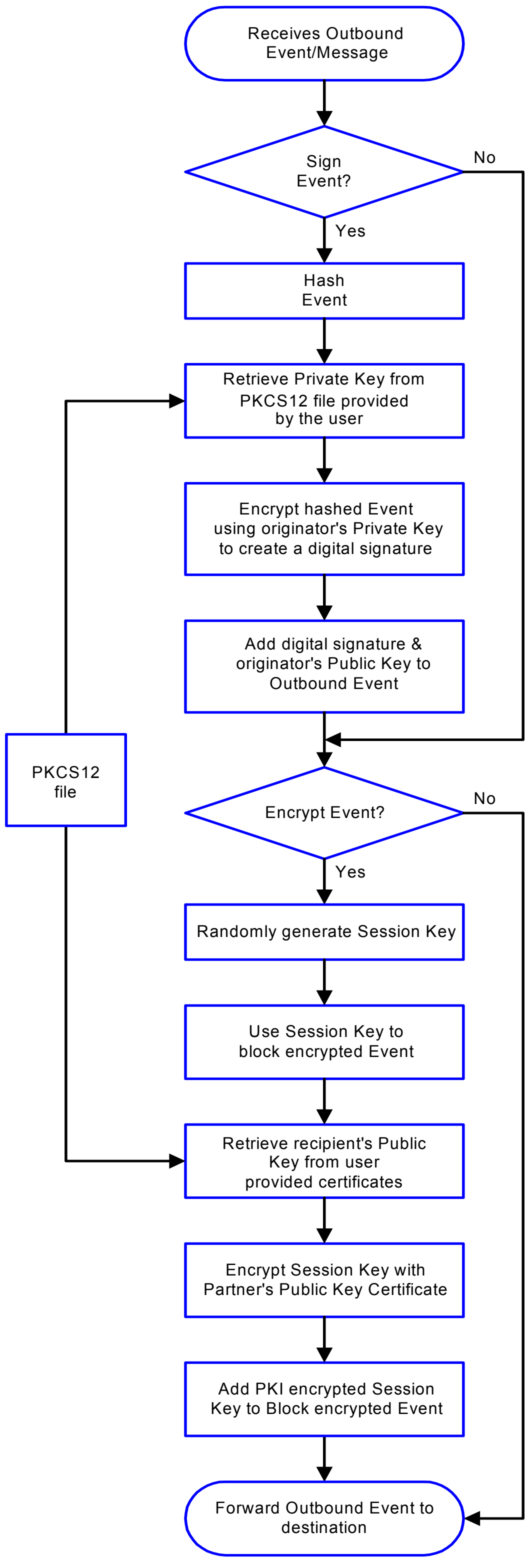

Receives an Inbound Message
Encrypted Event?
No
Yes
Separate Block encrypted Event from PKI encrypted Session Key
Encryption required for this Event?
No
Yes
Raise exception
Retrieve Receiver's Private Key from PKI Database
Decrypt Session Key using Private Key
Decrypt Event using Session Key
PKI Database
Signed Event?
No
Yes
Separate the digital signature and originator's Public Key from the body of the Event
Hash Event body
Encryption required for this Event?
No
Yes
Raise exception
Retrieve Public Key from PKI Database
Decrypt digital signature using Public Key
Raise exception
No
Decrypted digital signature = hashed Event body.
Yes
Forward Event body to destination

Figure 2 Outbound Signed/Encrypted Message


Receives Outbound Event/Message
Sign Event?
No
Yes
Hash Event
Retrieve Private Key from PKCS12 file provided by the user
Encrypt hashed Event using originator's Private Key to create a digital signature
Add digital signature & originator's Public Key to Outbound Event
PKCS12 file
Encrypt Event?
No
Yes
Randomly generate Session Key
Use Session Key to block encrypted Event
Retrieve recipient's Public Key from user provided certificates
Encrypt Session Key with Partner's Public Key Certificate
Add PKI encrypted Session Key to Block encrypted Event
Forward Outbound Event to destination

1.5 Supported Operating Systems

Secure Messaging Extension is supported on the following operating systems:

- Windows 2000, Windows 2000 SP1, Windows 2000 SP2, and Windows 2000 SP3
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- AIX 4.3.3
- HP-UX 11.0 and HP-UX 11i

1.6 System Requirements

To use SME, the following are required:

- An e*Gate Participating Host, version 4.5.1 or later
- A TCP/IP network connection
- A computer running Windows, to allow you to use the e*Gate Enterprise Manager and ETD Editor
- Additional disk space for the Add-on executable, configuration, library, and script files. The disk space is required on both the Participating and the Registry Host. Additional disk space is required to process and queue the data that this Add-on processes; the amount necessary varies based on the type and size of the data being processed, and any external applications performing the processing.
- Open and review the **Readme.txt** for SME for any additional requirements prior to installation. The **Readme.txt** is located on the Installation CD_ROM at setup\addons\ewsme.

Installed on the Participating Host

- Java JDK version 1.3.1

Chapter 2

Installation

This chapter describes the procedures for installing SME.

2.1 Installing SME on Windows NT 4.0 and Windows 2000

2.1.1 Pre-installation

- Exit all Windows programs before running the setup program, including any antivirus applications.
- You must have Administrator privileges to install this Add-on.

2.1.2 Installation Procedure

To install SME on a Windows system

1. Log in as an Administrator to the workstation on which you are installing the Add-on.
2. Insert the installation CD-ROM into the CD-ROM drive.
3. If the CD-ROM drive's Autorun feature is enabled, the setup application launches automatically; skip ahead to step 4. Otherwise, use the Windows Explorer or the Control Panel's **Add/Remove Applications** feature to launch the file **setup.exe** on the CD-ROM drive.
4. The InstallShield setup application launches. Follow the installation instructions until you come to the **Please choose the product to install** dialog box.
5. Select e*Gate Integrator, then click **Next**.
6. Follow the on-screen instructions until you come to the second **Please choose the product to install** dialog box.
7. Clear the check boxes for all selections except Add-ons, and then click **Next**.

8. Follow the on-screen instructions until you come to the **Select Components** dialog box.
9. Select (but do not check) **Agents**, and then click **Change**. The **SelectSub-components** dialog box appears.
10. Select **SME**. Click **Continue** to return to the **Select Components** dialog box, then click **Next**.
11. Follow the rest of the on-screen instructions to install SME. Be sure to install the SME files in the suggested client installation directory. The installation utility detects and suggests the appropriate installation directory. Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested installation directory setting.

Note: *Once you have installed and configured SME, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before SME can perform its intended functions. For more information about any of these procedures, see the online Help.*

*For more information about configuring SME, see the **e*Gate Integrator User's Guide**.*

2.2 Installing SME on UNIX

2.2.1 Pre-installation

Root privileges are not required to install SME. Log in under the user name that you wish to own the SME files. Be sure that this user has sufficient privileges to create files in the e*Gate directory tree.

2.2.2 Installation Procedure

To install SME on a UNIX system

1. Log in on the workstation containing the CD-ROM drive, and insert the CD-ROM into the drive.
2. If necessary, mount the CD-ROM drive.
3. At the shell prompt, type

   cd /cdrom
4. Start the installation script by typing

   setup.sh
5. A menu of options will appear. Select the **Install e*Way** option. Then, follow the additional on-screen directions.

Note: *Be sure to install the SME files in the suggested **client** installation directory. The installation utility detects and suggests the appropriate installation directory. **Unless you are directed to do so by SeeBeyond support personnel, do not change the suggested "installation directory" setting.***

6 After installation is complete, exit the installation utility and launch the Enterprise Manager.

Note: *Once you have installed and configured SME, you must incorporate it into a schema by defining and associating the appropriate Collaborations, Collaboration Rules, IQs, and Event Types before SME can perform its intended functions. For more information about any of these procedures, see the online Help system.*

*For more information about configuring SME, see the **e*Gate Integrator User's Guide**.*

2.3 Files/Directories Created by the Installation

The SME installation process installs the files shown in Table 1 within the e*Gate directory tree. Files are installed within the **egate\client** tree on the Participating Host and committed to the **default** schema on the Registry Host.

Table 1 Files Created by the Installation

e*Gate Directories	File(s)
\classes\	stcsme.jar
\etd\	sme.ctl
\etd\smeclient\	smeinputmsg.dtd smemessage.xsc
\client\ThirdParty\baltimore\classes	KeyToolsPro_All1.2.jar smime.jar

.

Chapter 3

Encrypted Message Formats, Digital Signature Formats, and Certificate Formats

This chapter provides an overview of the encrypted message formats, digital signatures and certificates that are handled by SME. In addition, this chapter describes how to use Windows 2000 and Microsoft Internet Explorer tools to transfer certificate formats accepted by the SMIME/C library

3.1 Encrypted Message Formats

This section provides examples of encrypted message formats.

PKCS#7 encrypted message format

The PKCS#7 format, as specified by RFC 2315, is used for basic digitally signed and/or encrypted data. This format does not provide a MIME header, and produces mostly binary data, except for a few character strings in an embedded certificate, as shown in the following example:

```
0     *†H†÷  0  1,$0,  0ˆ0,10UUS10\U

California1\0/UMonrovia1

0

U

STC10UDevelopment1'0%USTC Test Certificate Authority0*†H†÷
 V<±ïíà»,¯‡¾l-êÒTâž|g®<êÆ<õ¢\)Ç‰‡îQt£rµ»Ÿ½TûRP[Myß÷ ×ÚÚh-Íá–Ù¾—áô)Ã|bF©[_ˆHESM†2?k_

z¸~½ ï/ÈÕ+¶>æ³G¨šXK8yÃ!·Âyá—œB4U0 *†H†÷0*†H†÷b

4˜mDY  jE¯††,ë-]2žI¯e´G®†Ö¤ŸQÜ&ZÈX,¶Ê!4`RK”ÆE«9ýìÂPÝ Q- ní\=(-÷þÚïL
```

S/MIME2 encrypted message format (base64)

The S/MIME2 format is also used to represent digitally signed and/or encrypted data. This format provides a MIME header and encrypted results, with the binary data encoded as printable characters using the base64 method, as shown in the following example:

```
Content-Type: application/pkcs7-mime; name = "smime.p7m"
Content-Transfer-Encoding:base64
MIAGCSqGSIb3DQEHA6CAMIACAQAxggEkMIIBIAIBADCBiDCBgjELMAkGA1UEBhMCVVMxEzARBgNV

BAgTCkNhbGlmb3JuaWExETAPBgNVBAcTCE1vbnJvdmlhMQwwCgYDVQQKEwNTVEMxFDASBgNVBAsT

C0RldmVsb3BtZW50MScwJQYDVQQDEx5TVEMgVGVzdCBDZXJ0aWZpY2F0ZSBBdXRob3JpdHkCARMw

DQYJKoZIhvcNAQEBBQAEgYBR3Hwe+1JB2pZuR2XdNFS1DISYbgWHaXcmmpRZE+r35Ar5iaNlfRAj

ipc1RBW0HmidnWz3zBGYOml91btVjy2z6dmoDknnksgTI77YX727hESHgjCpxxcs+1kRzzI5ZUlU

WvvXeX/7wNkx3ZgJOrtIiXjfs6t8zW4edd1/13fQgjCABgkqhkiG9w0BBwEwFAYIKoZIhvcNAwcE

CBUeyy6UZb4koIAECOpD8MyUjNZ/BAjB0O2dStz8HgQIiPOI1H4tpfsECARjsNRDbMpqBAgtC3S1

7FnAWQQI8ymbLzoB4kUECF38LESRhXN2BAhcGnYwRqQDMgAAAAAAAAAAAAAA=
```

S/MIME2 encryption message format (binary)

This format represents a message as binary, non-printable data, with appropriate MIME headers, as shown in the following example:

```
Content-Type: application/pkcs7-mime; name = "smime.p7m"
Content-Transfer-Encoding:binary
0     *†H†÷  0  1,$0,  0ˆ0,10UUS10\U
California1\0/UMonrovia10
U
STC10UDevelopment1'0%USTC Test Certificate Authority0*†H†÷
 V<±ïíà»,¯Qt£rµ»Ÿ½TûRP[Myß÷ ×ÚÚh-Íá–Ù¾—áô)Ã|bF©[_ˆHESM†2?k …Bmm_t1Gòz
~½ ï/ÈÕ+¶>æ³G¨šXK8yÃ!·Âyá—œB4U0 *†H†÷0*†H†÷b
4˜mDY  jE¯††,ë-]2žI¯e´G®†Ö¤ŸQÜ&ZÈX,¶Ê!4`RK”ÆE«9ýìÂPÝ Q- ní\=(-÷þÚïL
```

3.2 Digital Signature Formats

Although signatures normally are found attached to the message or file that they sign, detached signatures are also supported. A detached signature may be stored and transmitted separately from the message it signs.

Table 2 lists the features of each encrypted message format for attached signatures.

Table 2 Formats for attached signatures

PKCS#7 Format	S/MIME2 Format
▪ Includes original document in plain text, digital signature, and certificates involved, encapsulated, and encoded in Abstract Syntax Notation One (ASN.1) standard format. **Note:** *ASN.1 is an ISO/IEC standard for encoding rules used in ANSI X.509 certificates and PKCS documents.* **Example** 0 *†H†÷ 0 10+ 0 *†H†÷ $: This is only a test message! ,m0,i0,Ò0*†H†÷ 0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0020509184633Z030509184633Z0w10UUS10\U California1\0/UMonrovia10-U SeeBeyond1 0 URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0‰ ®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ– ÍoÑLsuÉA#šk^# ü³ÅÅ§]ñsJAm£8ófsoU¢&mUþ„g,”>©k£ÄXqÜ±Q½êÔú9PºKÍ~’ú“/ 0*†H†÷ _bšFïo7r ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þˆ=›P}°æå·ÌZ§R˜øüÅÌ(àØIãµ ÷Ñj #›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#} …÷üç_"ëµÐÉµ4½¦1,-0,)0ˆ0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0+ 0*†H†÷ ”Ö>»/éR8¶ZaÖ” ¡ÝXS*¿£uõURÑˆ©pCËŸÂÍ,•Ÿ¶I/’{– ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥>Ì°]2ÅpÆ2*Ì |êÏË{lÊ–0%#t‹¥Œåœ ô› VÝ¹k	▪ Includes: ▪ MIME headers ▪ PKCS#7 attached signature object **Example** Content-Type: application/pkcs7-mime; name = "smime.p7m" Content-Transfer-Encoding:binary 0 *†H†÷ 0 10+ 0 *†H†÷ $: This is only a test message! ,m0,i0,Ò0*†H†÷ 0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0020509184633Z030509184633Z0w10UUS10\U California1\0/UMonrovia10-U SeeBeyond1 0 URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0‰ ®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ– ÍoÑLsuÉA#šk^# ü³ÅÅ§]ñsJAm£8ófsoU¢&mUþ„g,”>©k£ÄXqÜ±Q½êÔú9PºKÍ~’ú“/ 0*†H†÷ _bšFïo7r ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þˆ=›P}°æå·ÌZ§R˜øüÅÌ(àØIãµ ÷Ñj #›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#} …÷üç_"ëµÐÉµ4½¦1,-0,)0ˆ0,10UUS10\U California1\0/UMonrovia10 U STC10UDevelopment1'0%USTC Test Certificate Authority0+ 0*†H†÷ ”Ö>»/éR8¶ZaÖ” ¡ÝXS*¿£uõURÑˆ©pCËŸÂÍ,•Ÿ¶I/’{– ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥>Ì°]2ÅpÆ2*Ì |êÏË{lÊ–0%#t‹¥Œåœ ô› VÝ¹k

Table 3 lists the features of each encrypted message format for detached signatures.

Table 3 Formats for detached signatures

PKCS#7 Format	S/MIME2 Format

▪ Includes signature and certificate without the signed data.
 	
Note: *RNIF1.1 uses PKCS#7 and detached format*
 	
Example
 	
`0 *†H†÷  0 10+`
`  0 *†H†÷   ,m0,i0,Ò0*†H†÷ 0,10UUS10\U`
`California1\0/UMonrovia10`
`U`
`STC10UDevelopment1'0%USTC Test Certificate`
`Authority0020509184633Z030509184633Z0w10UUS10\U`
`California1\0/UMonrovia10-U`
`SeeBeyond1`
`0`
`URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0‰`
`®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ–`
`ÍoÑLsuÉA#šk^#`
`ü³ÅÅ§]ñsJAm£8ófsoU¢&mUþ„g,”>©k£ÄXqÜ±Q½êÔú9PºKÍ~’ú“/`
`0*†H†÷  _bšFïo7r`
`ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þˆ=›P}°æå·ÌZ§R˜øüÅÌ(àØIãµ ÷Ñj`
`#›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#}`
`…÷üç_"ëµÐÉµ4½¦1,-0,)0ˆ0,10UUS10\U`
`California1\0/UMonrovia1`
`0`
`U`
`STC10UDevelopment1'0%USTC Test Certificate`
`Authority0+`
` 0*†H†÷  ”Ö>»/éR8¶ZaÖ” ¡ÝXS*¿£uõURÑˆ©pCËŸÂÍ,•Ÿ¶I/’{-`
`ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥>Ì°]2ÅpÆ2*Ì`
`|êÏË{lÊ–0%#t‹¥Œåœ ô› VÝ¹k .`
	▪ Includes a MIME multipart message consisting of the original data in one segment, and the binary format signature in a second segment.
	
	Example
	
	`Content-Type: multipart/signed;`
	`protocol="application/pkcs7-signature";`
	`micalg="SHA1"; boundary="Boundary_12e4421e_NOEWUDYA"`
	
	`--Boundary_12e4421e_NOEWUDYA`
	`Content-Type: text/plain`
	
	`This is only a test message!`
	
	`--Boundary_12e4421e_NOEWUDYA`
	`Content-Type: application/pkcs7-signature;`
	`name="smime.p7s"`
	`Content-Transfer-Encoding: binary`
	`Content-Disposition: attachment; filename=smime.p7s`
	
	`0 *†H†÷  0 10+`
	`  0 *†H†÷   ,m0,i0,Ò0*†H†÷ 0,10UUS10\U`
	`California1\0/UMonrovia1`
	`0`
	`U`
	`STC10UDevelopment1'0%USTC Test Certificate`
	`Authority0020509184633Z030509184633Z0w10UUS10\U`
	`California1\0/UMonrovia10-U`
	`SeeBeyond1`
	`0`
	`URAD10USeeBeyond Test User 10Ÿ0*†H†÷ • 0‰`
	`®ŠGk•Éƒw¯¥S®¢_{!0Õ¢„&KÇéL›Ä,”1Än§lÏ»¶Õï¬©¥$lym´žÏ–`
	`ÍoÑLsuÉA#šk^#`
	`ü³ÅÅ§]ñsJAm£8ófsoU¢&mUþ„g,”>©k£ÄXqÜ±Q½êÔú9PºKÍ~’ú“/`
	`0*†H†÷  _bšFïo7r`
	`ç¦«HêAßl“"zgÛæAÌœXú,‘Õ:Þˆ=›P}°æå·ÌZ§R˜øüÅÌ(àØIãµ ÷Ñj`
	`#›òR1/”Œ80@ìûÍ,-/a†ÛZýý¥·s!ß¿ayS‘”#}`
	`…÷üç_"ëµÐÉµ4½¦1,-0,)0ˆ0,10UUS10\U`
	`California1\0/UMonrovia1`
	`0`
	`U`
	`STC10UDevelopment1'0%USTC Test Certificate`
	`Authority0+`
	` 0*†H†÷  ”Ö>»/éR8¶ZaÖ” ¡ÝXS*¿£uõURÑˆ©pCËŸÂÍ,•Ÿ¶I/’{-`
	`ªÓIÊF62žSð ‡/ñI²e^ü#â„àðf·n(”aE±cÓ,Å¥>Ì°]2ÅpÆ2*Ì`
	`|êÏË{lÊ–0%#t‹¥Œåœ ô› VÝ¹k`
	`--Boundary_12e4421e_NOEWUDYA--`

Table 3 Formats for detached signatures

PKCS#7 Format	S/MIME2 Format
	▪ Includes a MIME multipart message consisting of the original data in one segment, and the base64-encoded signature in a second segment

Example

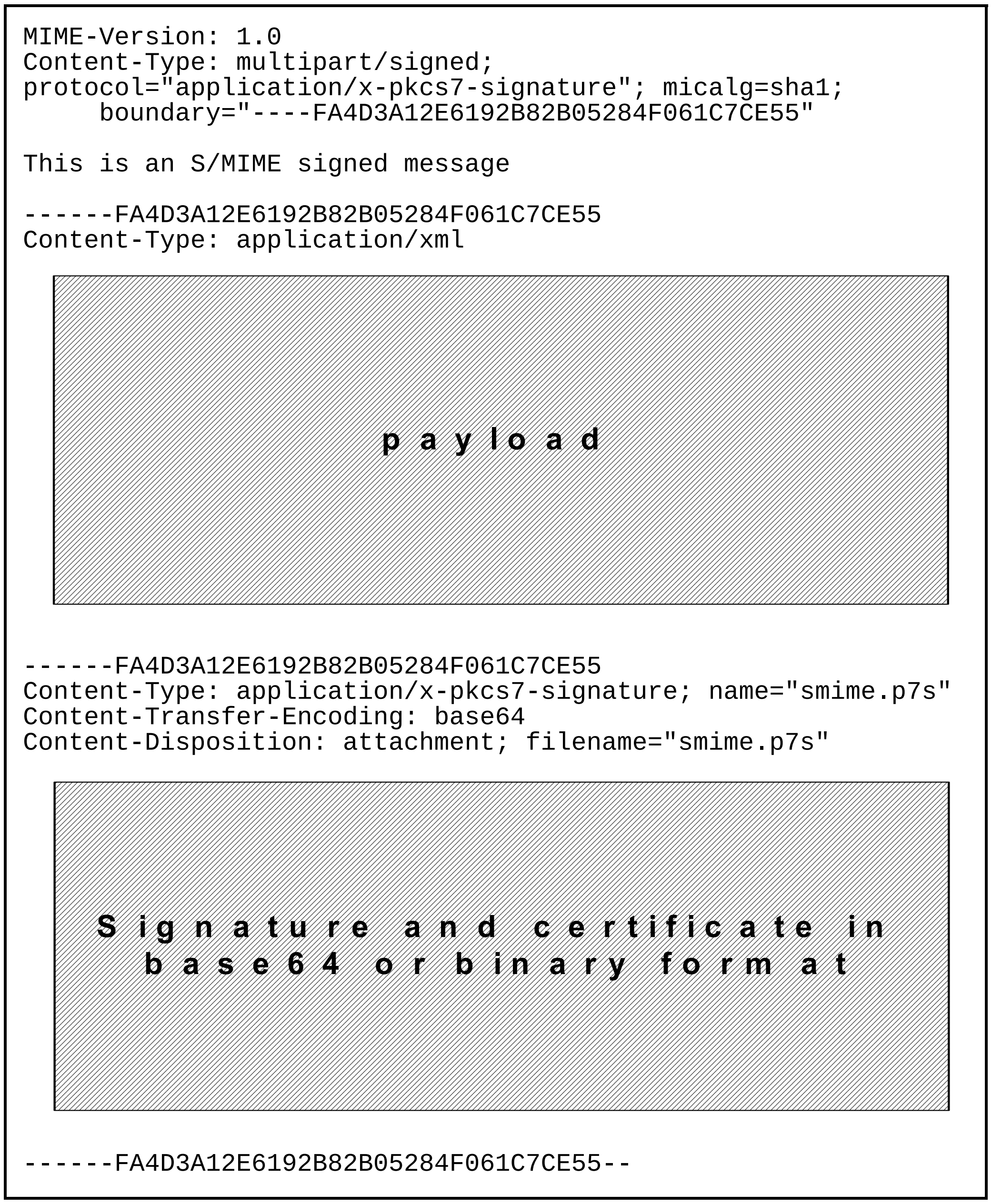
```
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg="SHA1"; boundary="Boundary_12e4421e_FNGBRNRI"


--Boundary_12e4421e_FNGBRNRI
Content-Type: text/plain

This is only a test message!

--Boundary_12e4421e_FNGBRNRI
Content-Type: application/pkcs7-signature;
name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

MIAGCSqGSIb3DQEHAqCAMIACAQExCzAJBgUrDgMCGgUAMIAGCSqG
SIb3DQEHAQAAoIICbTCCAmkw
ggHSAgETMA0GCSqGSIb3DQEBBAUAMIGCMQswCQYDVQQGEwJVUzET
MBEGA1UECBMKQ2FsaWZvcm5p
YTERMA8GA1UEBxMITW9ucm92aWExDDAKBgNVBAoTA1NUQzEUMBIG
A1UECxMLRGV2ZWxvcG1lbnQx
JzAlBgNVBAMTHlNUQyBUZXN0IENlcnRpZmljYXRlIEF1dGhvcml0
eTAeFw0wMjA1MDkxODQ2MzNa
Fw0wMzA1MDkxODQ2MzNaMHcxCzAJBgNVBAYTAlVTMRMwEQYDVQQI
EwpDYWxpZm9ybmlhMREwDwYD
VQQHEwhNb25yb3ZpYTESMBAGA1UEChMJU2VlQmV5b25kMQwwCgYD
VQQLEwNSQUQxHjAcBgNVBAMT
FVNlZUJleW9uZCBUZXN0IFVzZXIgMTCBnzANBgkqhkiG9w0BAQEF
AAOBjQAwgYkCgYEAropHa5XJ
g3evpQFTrqJfeyEw1aKEJksfx+lMm8QsnTHEbqdsj8+7ttXvrKml
JGx5bbSezzkIl81v0Uwfc3XJ
QSOaA2teIxr8swvFDcWnXfFzSkFtkKM482Zzb1WiJhZtVf6EZywD
nT6pAmujxFhx3LFRverU+jlQ
ukvNfpL6ky8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQBfE2IVmo9G
7xRvN3IZC+emq0jqE0HfbJMi
eg1nf9vmQcycWBT6LJHVOt6IPZtQfbDmf+W3zFqnUpj4/
MUGzCjg2EnjtYD30Y9qI5vyF1IxL52M
ODBA7PvNgq0vYYYY239a/f2lt3Mh378dYXlTkZ0jfQmF9/
znXyLrtdDJtTS9pjGCAS0wggEpAgEB
MIGIMIGCMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5p
YTERMA8GA1UEBxMITW9ucm92
aWExDDAKBgNVBAoTA1NUQzEUMBIGA1UECxMLRGV2ZWxvcG1lbnQx
JzAlBgNVBAMTHlNUQyBUZXN0
IENlcnRpZmljYXRlIEF1dGhvcml0eQIBEzAHBgUrDgMCGjANBgkq
hkiG9w0BAQEFAASBgJCd1gU+
uw/pUji2WmHWlCChE91YUyq/
o3X1VQVS0YipcEPLn8LNLI2ftkkvknuWqtNJykY2Mp5T8ICHFy/x
SbJlXvwj4oTg8Ga3bgUdKJ1hRbFj0yzFpT7MsF0yxXDGMirMCnzq
z8t7bMqBlzAlIw10i6WM5ZyA
9JsaH1bdE7lrAAAAAAAA
--Boundary_12e4421e_FNGBRNRI--
```

3.3 Signing and Attaching Signatures

In an S/MIME message with a detached signature, the signature is calculated over on the entire payload data, in addition to its MIME header(s). The default Content-Type for such a MIME part is text/plain.

If signing a Content-Type other than text/plain, the user must generate a Content-Type header line for the payload. All other MIME headers and boundaries, including those of the detached signature part, are produced by SME.

An example XML message, digitally signed with a base64-encoded detached S/MIME signature is shown below.



```
MIME-Version: 1.0
Content-Type: multipart/signed;
protocol="application/x-pkcs7-signature"; micalg=sha1;
     boundary="----FA4D3A12E6192B82B05284F061C7CE55"

This is an S/MIME signed message

------FA4D3A12E6192B82B05284F061C7CE55
Content-Type: application/xml
```

```
------FA4D3A12E6192B82B05284F061C7CE55
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
```

```
------FA4D3A12E6192B82B05284F061C7CE55--
```

3.4 Certificate Formats

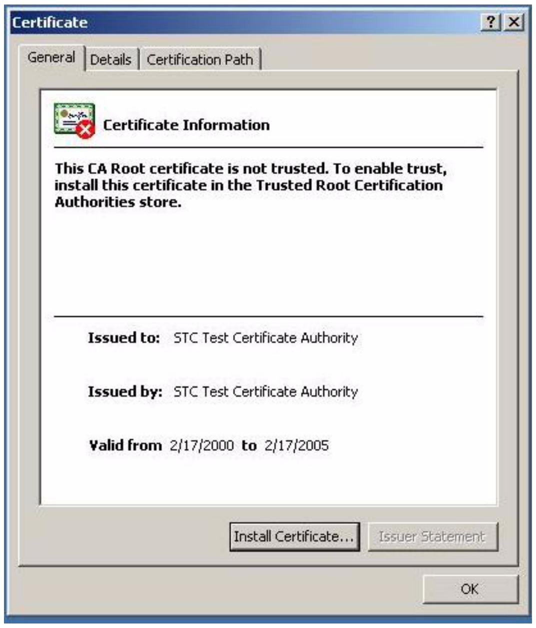
SME accepts certificates in PKCS#7 format. In addition, DER encoded binary X.509 and Base64 encoded X.509 format certificates are accepted.

Windows 2000 and Microsoft™ Internet Explorer provide a Certificate Wizard tool to convert between formats.

To convert from one certificate format to another

1 Double-click the certificate file to open the certificate properties, as shown in Figure 3.

Figure 3 Certificate File



Certificate Information

This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

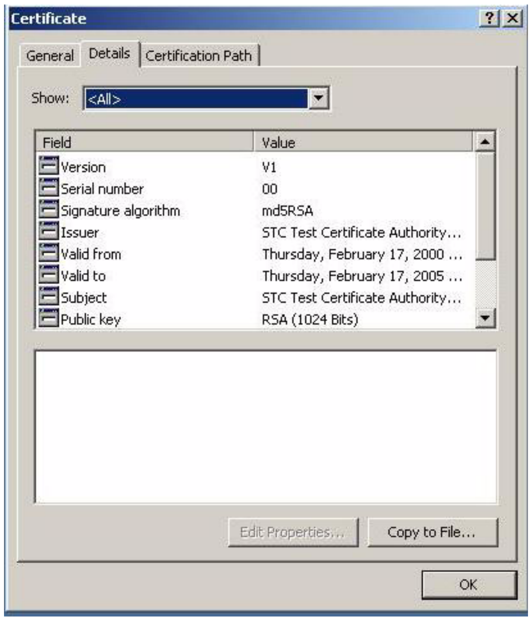
Issued to: STC Test Certificate Authority

Issued by: STC Test Certificate Authority

Valid from 2/17/2000 to 2/17/2005

2 Select the **Details** tab, as shown in Figure 4.

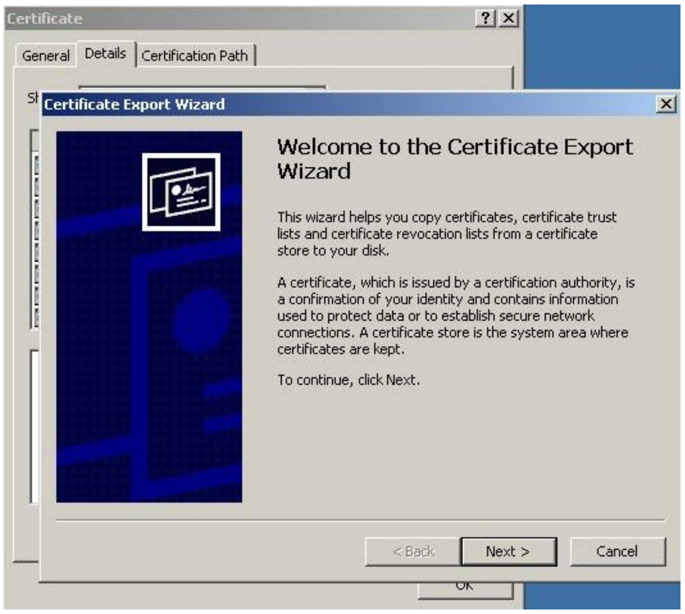
Figure 4 Certificate Detail Tab



Field	Value
Version	V1
Serial number	00
Signature algorithm	md5RSA
Issuer	STC Test Certificate Authority...
Valid from	Thursday, February 17, 2000 ...
Valid to	Thursday, February 17, 2005 ...
Subject	STC Test Certificate Authority...
Public key	RSA (1024 Bits)

3 Click **Copy to File**. The Certificate Export Wizard appears, as shown in Figure 5.

Figure 5 Windows 2000 Certificate Export Wizard



Welcome to the Certificate Export Wizard

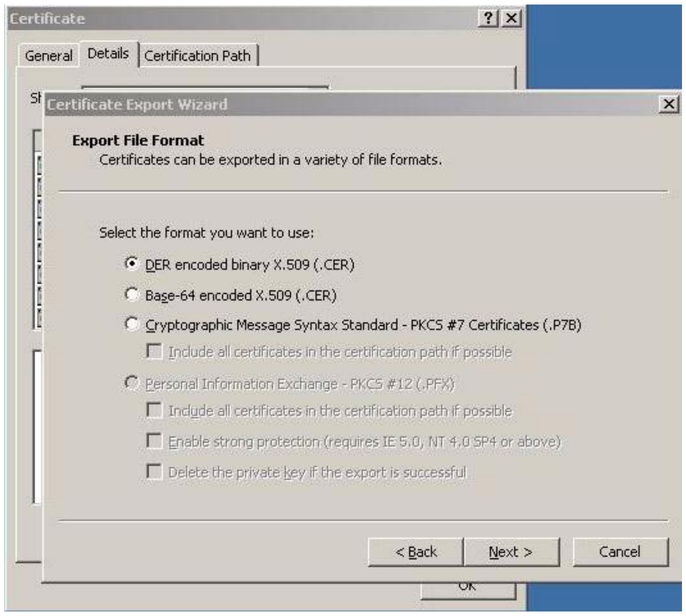
This wizard helps you copy certificates, certificate trust lists and certificate revocation lists from a certificate store to your disk.

A certificate, which is issued by a certification authority, is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

To continue, click Next.

4 Click **Next** to open the certificate export file format, as shown in Figure 6. Select the format.

Figure 6 Certificate Export Wizard file format



Export File Format
Certificates can be exported in a variety of file formats.

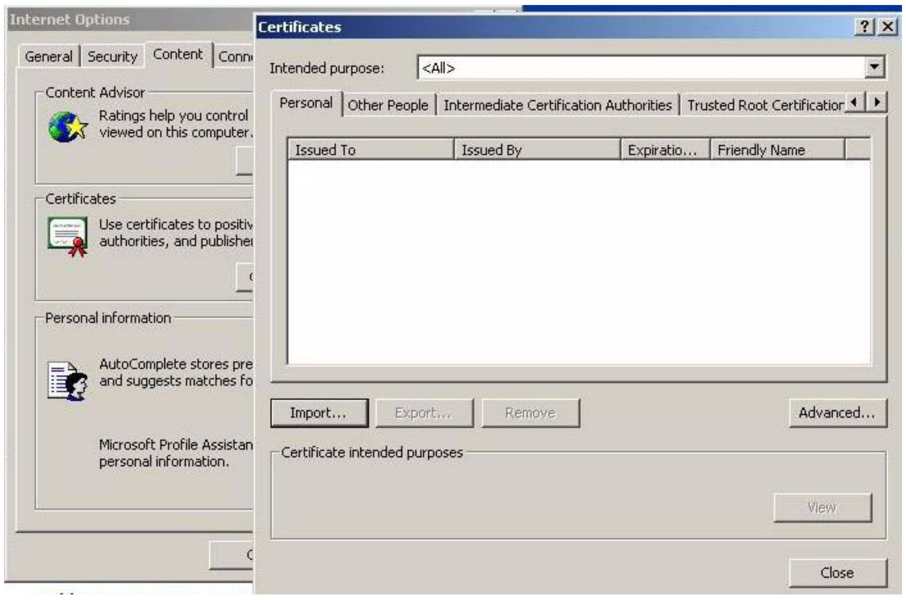
Select the format you want to use:

- [✓ DER encoded binary X.509 (.CER)]
- Base-64 encoded X.509 (.CER)
- Cryptographic Message Syntax Standard - PKCS #7 Certificates (.P7B)
  - Include all certificates in the certification path if possible
- Personal Information Exchange - PKCS #12 (.PFX)
  - Include all certificates in the certification path if possible
  - Enable strong protection (requires IE 5.0, NT 4.0 SP4 or above)
  - Delete the private key if the export is successful

To transfer the certificate formats using Microsoft™ Internet Explorer

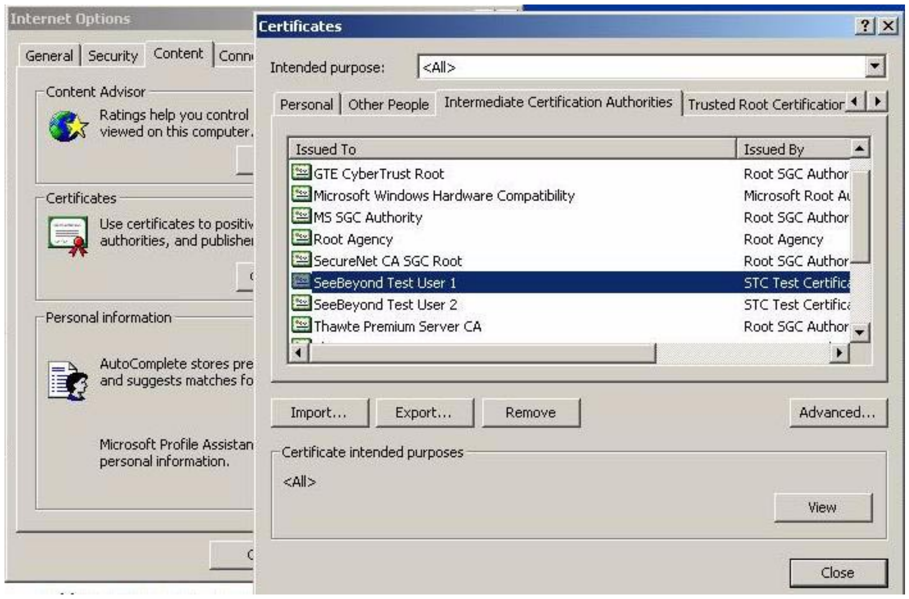
1. From the Tools menu, click **Internet Options**.
2. Click the **Content** tab and then click **Certificates**. The Certificates dialog appears, as shown in Figure 7.

Figure 7 Internet Explorer Certificates



3. Click **Import**, to import your certificate.
4. Click the **Intermediate Certification Authorities** tab. Select the certificate to import, and then click **Export**. The Certificates properties dialog appears, as shown in Figure 8.

Figure 8 Internet Explorer Intermediate Certificate Authorities



Issued To	Issued By
GTE CyberTrust Root	Root SGC Author
Microsoft Windows Hardware Compatibility	Microsoft Root Au
MS SGC Authority	Root SGC Author
Root Agency	Root Agency
SecureNet CA SGC Root	Root SGC Author
SeeBeyond Test User 1	STC Test Certific
SeeBeyond Test User 2	STC Test Certific
Thawte Premium Server CA	Root SGC Author

5 Select the format, and save the file.

3.5 Private Key Format

Private keys, used by SME in the decryption and signing processes, are required to be in PKCS#12 format. If a key has been generated through a browser-based process and appears among your personal certificates in Microsoft Internet Explorer, it may be exported to a PKCS#12 file for use by SME.

For the export procedure, refer to the previous section, choosing your key from the Personal tab in the Internet Explorer Certificate Manager, making sure to select "**Yes, export the private key**" and uncheck the "**Enable strong protection**" option.

Note: *Remember the password you specify to encrypt the exported file; it is needed during the SME configuration process, in order to allow decryption and use the key.*

Chapter 4

e*Way Connection Configuration

This chapter describes how to configure the following components of SME.

- **Configuring the Multi-Mode e*Way** on page 27
- **Configuring e*Way Connections** on page 32

4.1 Configuring the Multi-Mode e*Way

A Multi-Mode e*Way is a multi-threaded component used to route and transform data within e*Gate. Unlike traditional e*Ways, Multi-Mode e*Ways can use multiple simultaneous e*Way Connections to communicate with several external systems, as well as IQs or IQ Managers. The following section describes how to create and configure the Multi-Mode e*Way component for SME.

Note: *The Multi-Mode e*Way properties are set using the Enterprise Manager.*

4.1.1 Creating a Multi-Mode e*Way

1. Select the Navigator's Components tab.
2. Open the host on which you want to create the e*Way.
3. On the Palette, click on **Create a New e*Way** to create a new **e*Way**.
4. Enter the name of the new e*Way, then click **OK**.
5. Select the new e*Way component, right-click, and select **Properties**. The e*Way Properties dialog box opens.
6. The **Executable File** field defaults to **stceway.exe**. (stceway.exe is located in the "bin\" directory).
7. Type any additional command line arguments that the e*Way may require in the **Additional Command Line Arguments** field, taking care to insert them *at the end* of the existing command-line string. Be careful not to change any of the default arguments unless you have specific need to do so.
8. Click **New** under the **Configuration File** field to create a new configuration file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file. The Editor opens to edit settings for the Multi-Mode e*Way. The Multi-Mode e*Way

Configuration Editor opens. The following section provides more information on these parameters.

9 After selecting the desired parameters, **Save** the configuration file and select **Promote to Run Time**. Click **OK** to close the e*Way Properties dialog.

Note: *Although you can make changes to these configuration options while an e*Way is running, the changes will not take effect until you stop and restart the e*Way.*

For more information on Multi-Mode e*Way settings and properties see the *e*Gate Integrator User's Guide*, the *Standard e*Way Intelligent Adapter User's Guide*, or consult the e*Way Editor's online Help.

4.1.2 Multi-Mode e*Way Configuration Parameters

The Multi-Mode e*Way configuration parameters are organized in two sections:

- **JVM Settings** on page 28
- **General Settings** on page 32

4.1.3 JVM Settings

The **JVM Settings** define the basic Java Virtual Machine configuration for the Multi-Mode e*Way. The **JVM Settings** section contains the following parameters:

- **JNI DLL Absolute Pathname** on page 29
- **CLASSPATH Prepend** on page 29
- **CLASSPATH Override** on page 30
- **CLASSPATH Append From Environment Variable** on page 30
- **Initial Heap Size** on page 30
- **Maximum Heap Size** on page 30
- **Maximum Stack Size for Native Threads** on page 31
- **Maximum Stack Size for JVM Threads** on page 31
- **Disable JIT** on page 31
- **Remote debugging port number** on page 31
- **Suspend option for debugging** on page 31
- **Auxiliary JVM Configuration File** on page 32

These settings are listed in the **Goto Parameter** drop-down list box when you select **JVM Settings** in the **Goto Section** list box on the **Edit Settings** window.

The following subsections describe the parameters available in **JVM Settings**.

JNI DLL Absolute Pathname

Description

Specifies the absolute pathname to where the JNI DLL installed by the *Java 2 SDK 1.3* is located on the Participating Host.

Required Values

A valid pathname.

Additional Information

The JNI dll name varies on different O/S platforms, as listed in the following table:

OS	Java 2 JNI DLL Name
Windows 2000/NT 4.0	jvm.dll
Solaris	libjvm.so
HP-UX	libjvm.sl

The value assigned can contain a reference to an environment variable, by enclosing the variable name within a pair of % symbols. For example:

```
%MY_JNIDLL%
```

Such variables can be used when multiple Participating Hosts are used on different platforms.

To ensure that the JNI DLL loads successfully, the Dynamic Load Library search path environment variable must be set appropriately to include all the directories under the Java 2 SDK (or JDK) installation directory that contain shared libraries (UNIX) or DLLs (NT).

CLASSPATH Prepend

Description

Specifies the paths to be prepended to the CLASSPATH environment variable for the JVM.

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

If left unset, no paths will be prepended to the CLASSPATH environment variable.

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_PRECLASSPATH%
```

CLASSPATH Override

Description

Specifies the complete CLASSPATH variable to be used by the JVM. This parameter is optional. If left unset, an appropriate CLASSPATH environment variable (consisting of required e*Gate components concatenated with the system version of CLASSPATH) will be set.

Note: *All necessary JAR and ZIP files needed by both e*Gate and the JVM must be included. It is advised that the **CLASSPATH Prepend** parameter should be used.*

Required Values

An absolute path or an environmental variable. This parameter is optional.

Additional Information

Existing environment variables may be referenced in this parameter by enclosing the variable name in a pair of % signs. For example:

```
%MY_CLASSPATH%
```

CLASSPATH Append From Environment Variable

Description

Specifies whether the path is appended for the CLASSPATH environmental variable to jar and zip files needed by the JVM.

Required Values

YES or **NO**. The configured default is YES.

Initial Heap Size

Description

Specifies the value for the initial heap size in bytes. If set to 0 (zero), the preferred value for the initial heap size of the JVM is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Heap Size

Description

Specifies the value of the maximum heap size in bytes. If set to 0 (zero), the preferred value for the maximum heap size of the JVM is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for Native Threads

Description

Specifies the value of the maximum stack size in bytes for native threads. If set to 0 (zero), the default value is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Maximum Stack Size for JVM Threads

Description

Specifies the value of the maximum stack size in bytes for JVM threads. If set to 0 (zero), the preferred value for the maximum heap size of the JVM is used.

Required Values

An integer between 0 and 2147483647. This parameter is optional.

Disable JIT

Description

Specifies whether the Just-In-Time (JIT) compiler is disabled.

Required Values

YES or **NO**.

Note: *This parameter is not supported for Java Release 1.*

Remote debugging port number

Description

Specifies the port number for the remote debugging of the JVM.

Required Values

An integer between 2000 and 65536.

Suspend option for debugging

Description

Specifies whether the option for debugging is enabled or suspended upon JVM startup.

Required Values

YES or **NO**.

Auxiliary JVM Configuration File

Description

Specifies an auxiliary JVM configuration file for additional parameters.

Required Values

The location of the auxiliary JVM configuration file.

4.1.4 General Settings

The **General Settings** section contains the following parameters:

- **Rollback Wait Interval** on page 32
- **Standard IQ FIFO** on page 32

These settings are listed in the **Goto Parameter** drop-down list box when you select **General Settings** in the **Goto Section** list box on the **Edit Settings** window.

The following subsections describe the parameters available in **General Settings**.

Rollback Wait Interval

Description

Specifies the time interval to wait before rolling back a transaction.

Required Values

An integer between 0 and 99999999, representing the time interval in milliseconds.

Standard IQ FIFO

Description

Specifies whether the highest priority messages from all STC_Standard IQs will be delivered in the first-in-first-out (FIFO) order.

Required Values

Select **YES** or **NO**. **YES** indicates that the e*Way will retrieve messages from all STC_Standard IQs in the first-in-first-out (FIFO) order. **NO** indicates that this feature is disabled. **NO** is the configured default.

For more information on the STC_Standard IQ Service and FIFO processing, see the *e*Gate Integrator Intelligent Queue Services Reference Guide*.

4.2 Configuring e*Way Connections

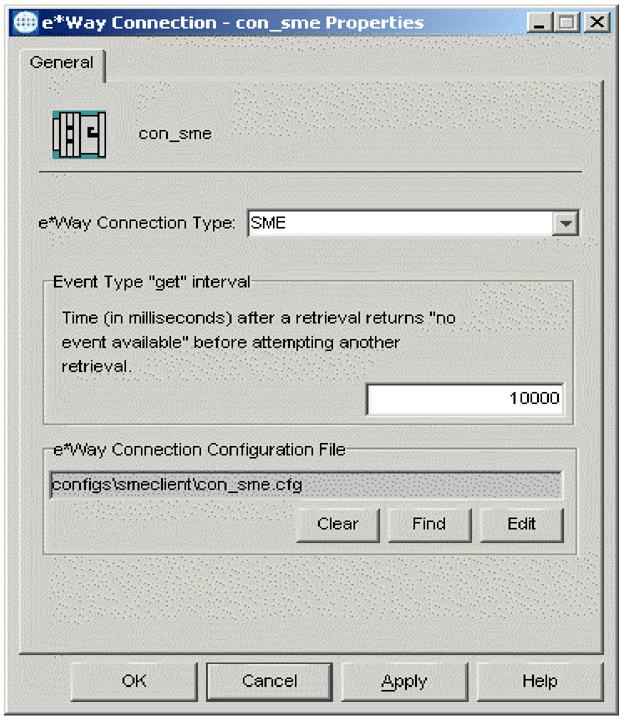
e*Way Connections are the encoding of access information for specific external connections. The e*Way Connection configuration file contains the parameters necessary for connecting with a specific external system.

Note: *The e*Way Connection parameters are set using the Enterprise Manager.*

Creating an e*Way Connection

1. In the Enterprise Manager's Component editor, select the **e*Way Connections** folder.
2. On the palette, click on **Create a New e*Way Connection**.
3. The **New e*Way Connection Component** dialog box opens. Enter a name for the new e*Way Connection and click **OK**.
4. Double-click on the new e*Way Connection. The **e*Way Connection Properties** dialog box opens, as shown in Figure 9.

Figure 9 e*Way Connection Properties



e*Way Connection - con_sme Properties

General

con_sme

e*Way Connection Type: SME

Event Type "get" interval

Time (in milliseconds) after a retrieval returns "no event available" before attempting another retrieval.

10000

e*Way Connection Configuration File

configs\smeclient\con_sme.cfg

5. From the **e*Way Connection Type** drop-down box, select **SME**.
6. Enter the **Event Type "get"** interval in the dialog box provided. The configured default is 10000 milliseconds.
7. Click **New** under the **e*Way Connection Configuration File** field to create a new configuration file, **Find** to select an existing configuration file, or **Edit** to edit the currently selected file. The e*Way Connection Configuration Editor opens. The following section provides more information on these e*Way Connection parameters.
8. After selecting the desired parameters, **Save** the configuration file and select **Promote to Run Time**. Click **OK** to close the e*Way Connection Properties dialog.

Note: *If changes are made to an existing e*Way Connection file, any e*Ways using the revised e*Way Connection must be restarted.*

e*Way Connections are set using the Enterprise Manager.

To create and configure e*Way Connections

1. In the Enterprise Manager's **Component** editor, select the **e*Way Connections** folder.
2. On the palette, click on the icon to create a new **e*Way Connection**.
3. The **New e*Way Connection Component** dialog box opens, enter a name for the **e*Way Connection**. Click **OK**.
4. Double-click on the new **e*Way Connection**. For this example, the connection has been defined as **con_SME**.
5. The **e*Way Connection Properties** dialog box opens.
6. From the **e*Way Connection Type** drop-down box, select **SME**.
7. Enter the **Event Type "get"** interval in the dialog box provided.
8. From the **e*Way Connection Configuration File**, click **New** to create a new Configuration File for this e*Way Connection. (To use an existing file, click **Find**.)

The SME e*Way Connection configuration parameters are organized into the following sections:

- **Connector** on page 34
- **Encrypt** on page 35
- **Decrypt** on page 37
- **Sign** on page 38
- **Verify** on page 39
- **Certificate** on page 40
- **CRL** on page 41

4.2.1 Connector

This section contains the following top level parameters:

- type
- class
- Property.Tag

Type

Description

Specifies the type of connection.

Required Values

SME. The value defaults to SME.

Class

Description

Specifies the class name of the SME Client connector object.

Required Values

A valid package name. The default is com.stc.sme.eway.SMEClientConnector.

Property.Tag

Description

Specifies the data source identity. This parameter is required by the current EBobConnectorFactory.

Required Values

A valid data source package name.

4.2.2 Encrypt

This section contains the following top level parameters:

- Certificate
- format
- algorithm
- MessageFormat
- EncodingFormat

Certificate

Description

Specifies the certificate of the partner to which encrypted messages is sent.

A certificate is the binding between an individual or an organization, and a public key. It contains identity information, the subjects's public key, the identity of the issuing Certificate Authority, and the digital signature performed by the Certificate Authority.

Required Values

A valid certificate.

Format

Description

Specifies the certificate format.

Required Values

The appropriate certificate format. One of three provided: DER, PEM, or PK7. The default is DER.

Note: *The user should be aware of the certificate format being used, since the certificate format specifies the input format. Table 4 defines each of the accepted certificate formats.*

Table 4 Accepted Certificate Formats

Distinguished Encoding Rules (DER) format	Privacy Enhanced Mail (PEM) format	PKCS#7 (PK7) format
DER encoded Certificate Revocation List (CRL) structure.	Base64 encoded version of the DER format with header and footer lines.	A PKCS#7 format message which contains only the user's certificate (no data).

Algorithm

Description

Specifies the algorithm used for encryption.

Required Values

A string. The appropriate encryption algorithm. One of four provided: DES_EDE3_CBC, RC2_CBC_40, RC2_CBC_64, RC2_CBC_128. The default is DES_EDE3_CBC.

MessageFormat

Description

Specifies the format used for the encrypted message.

Required Values

The appropriate message format. One of two provided: PKCS7, or SMIME2. The default is SMIME2.

See **"Encrypted Message Formats" on page 17** for more information on message formats.

EncodingFormat

Description

Specifies the format used to encode the output message. This setting is only applied if MessageFormat is set to SMIME2.

Required Values

The appropriate encoding format. One of two provided: Base64 or Binary.

The default is Base64.

4.2.3 Decrypt

This section contains the following top level parameters:

- Message Format
- Encoding Format
- PKCS12
- PassPhrase

MessageFormat

Description

Specifies the format used for encrypting the message.

Required Values

The appropriate encryption format. One of two provided: PKCS7 or SMIME2.

The default is SMIME2.

EncodingFormat

Description

Specifies the encoding format for the output message. This setting is only applied if MessageFormat is set to SMIME2.

Required Values

The appropriate encoding format. One of two provided: Base64 or Binary.

The default is Base64.

PKCS12

Description

Specifies the PKCS12 file. The PKCS12 format is used for storage of private keys. The PKCS12 file is where the private key is stored.

Required Values

The valid path and PKCS12 file.

Note: *SME supports only PKCS12-formatted private key storage files. Other formats can be converted to PKCS12 by using Microsoft™ Internet Explorer or Netscape™ Navigator.*

PassPhrase

Description

Specifies the PassPhrase (password) used to protect/access the PKCS12 file.

Required Values

A valid PassPhrase.

4.2.4 Sign

This section contains the following top level parameters:

- algorithm
- detached
- MessageFormat
- EncodingFormat
- PKCS12
- PassPhrase

Algorithm

Description

Specifies the signing algorithm, the algorithm used to sign the message.

Required Values

The appropriate algorithm type. One of two provided, RSA_MD5 or RSA_SHA1, as defined in the following table.

RSA_MD5	RSA_SHA1
Rivest-Shamir-Adelman (**RSA**). **MD5** is a message digest algorithm.	Rivest-Shamir-Adelman (**RSA**). Secure Hash Algorithm1 (**SHA1**), an NIST FIPS 180-1 standard message digest algorithm.

The default is RSA_SHA1.

Detached

Description

Specifies whether the signature is separated from the original message.

Required Values

Yes or No. Yes detaches the signature.

MessageFormat

Description

Specifies the format used for encrypting the message.

Required Values

The appropriate encryption format. One of two provided: PKCS7 or SMIME2.

The default is SMIME2.

EncodingFormat

Description

Specifies the encoding format for the output message. This setting is only applied if MessageFormat is set to SMIME2.

Required Values

The appropriate encoding format. One of two provided: Base64 or Binary.

The default is Base64.

PKCS12

Description

Specifies the PKCS12 file. The PKCS12 file is where the private key is stored.

Required Values

The valid path and PKCS12 file.

PassPhrase

Description

Specifies the PassPhrase (password) used to protect/access the PKCS12 file.

Required Values

A valid PassPhrase.

4.2.5 Verify

This section contains the following top level parameters:

- MessageFormat
- EncodingFormat
- Certificate
- format

MessageFormat

Description

Specifies the format used for encrypting the message.

Required Values

The appropriate encryption format. One of two provided: PKCS7 or SMIME2.

The default is SMIME2.

EncodingFormat

Description

Specifies the encoding format for the output message. This setting is only applied if MessageFormat is set to SMIME2.

Required Values

The appropriate encoding format. One of two provided: Base64 or Binary.

The default is Base64.

Certificate

Description

Specifies the certificate used to verify the signed message. If not set, the certificate attached to the signed message is used to verify the signed message.

Required Values

A valid certificate.

Format

Description

Specifies the certificate format.

Required Values

The appropriate certificate format. One of three provided: DER, PEM, or PK7. For more information, see **Table 4 on page 36**.

The default is DER.

4.2.6 Certificate

This section contains the following top level parameters:

- Checking
- TrustedCA
- format

Checking

Description

Specifies the method used to assess trustworthiness of a certificate. Three options are available for certificate trust checking:

- **Direct**: The certificate used in the program has been received through trusted means. Or, the certificate's authenticity has been directly verified with its owner and should be trusted regardless of its issuer. The **Direct** option must be used for self-signed certificates, and may be used for other, CA-issued certificates.

- **CA**: If this option is selected, only those certificates issued by one of the trusted CAs can be used in the program. This option requires that one or more CA certificates be specified under the **TrustedCA** parameter.
- **CA_CRL**: As with the **CA** option, only certificates issued by the trusted CAs are allowed. In addition, certificates are verified against Certificate Revocation Lists (CRLs). If selected, this requires that a CRL be configured (see **Trusted CA** below).

Required Values

The appropriate trust verification method. One of three options provided: DIRECT, CA, or CA_CRL.

The default is CA.

TrustedCA

Description

Specifies the certificates belonging to the trusted CAs. Certificates entered here are used to verify whether or not other certificates are to be trusted for use within the program. Multiple CAs may be selected. This parameter is required if either CA or CA_CRL trust checking method is chosen above.

Note: *This parameter is not required if the Direct option is set for the Checking parameter.*

Required Values

One or more trusted CA certificates.

Format

Description

Specifies the format of the CA's certificate. If PK7 is selected, the end entity is regarded as the CA's Certificate.

Required Values

The appropriate certificate format. One of three provided: DER, PEM, or PK7. For more information, see **Table 4 on page 36**.

The default is DER.

4.2.7 CRL

CRL is the acronym for Certificate Revocation List, a time stamped list identifying revoked certificates. It is signed by a Certification Authority (CA) and made freely available for download by HTTP in a public repository.

This section contains the following top level parameters:

- filename
- CACRLCertifciate
- CACRLformat

- CACertifciate
- format

Filename

Description

Specifies the file name of the stored CRL.

DER is the required format.

Required Values

The path and file name of the stored CRL.

CACRLCertificate

Description

Specifies the certificate of the CA that was used to sign the CRL.

Required Values

A valid certificate name.

CACRLformat

Specifies the format of the CA's certificate. If PK7 is selected, the end entity is regarded as the CA's Certificate.

Required Values

The appropriate certificate format. One of three provided: DER, PEM, or PK7. For more information, see **Table 4 on page 36**.

The default is DER.

CACertificate

Description

Specifies the certificate of the CA that was used to sign user's certificate. This certificate is regarded as the trusted CA certificate. If it is the same as the one used to sign the CRL, leave this parameter blank.

Required Values

A valid certificate name.

Format

Description

Specifies the format of the CA's certificate. If PK7 is selected, the end entity is regarded as the CA's Certificate.

Required Values

The appropriate certificate format. One of three provided: DER, PEM, or PK7. For more information, see **Table 4 on page 36**.

The default is DER.

Chapter 5

ETD Structure

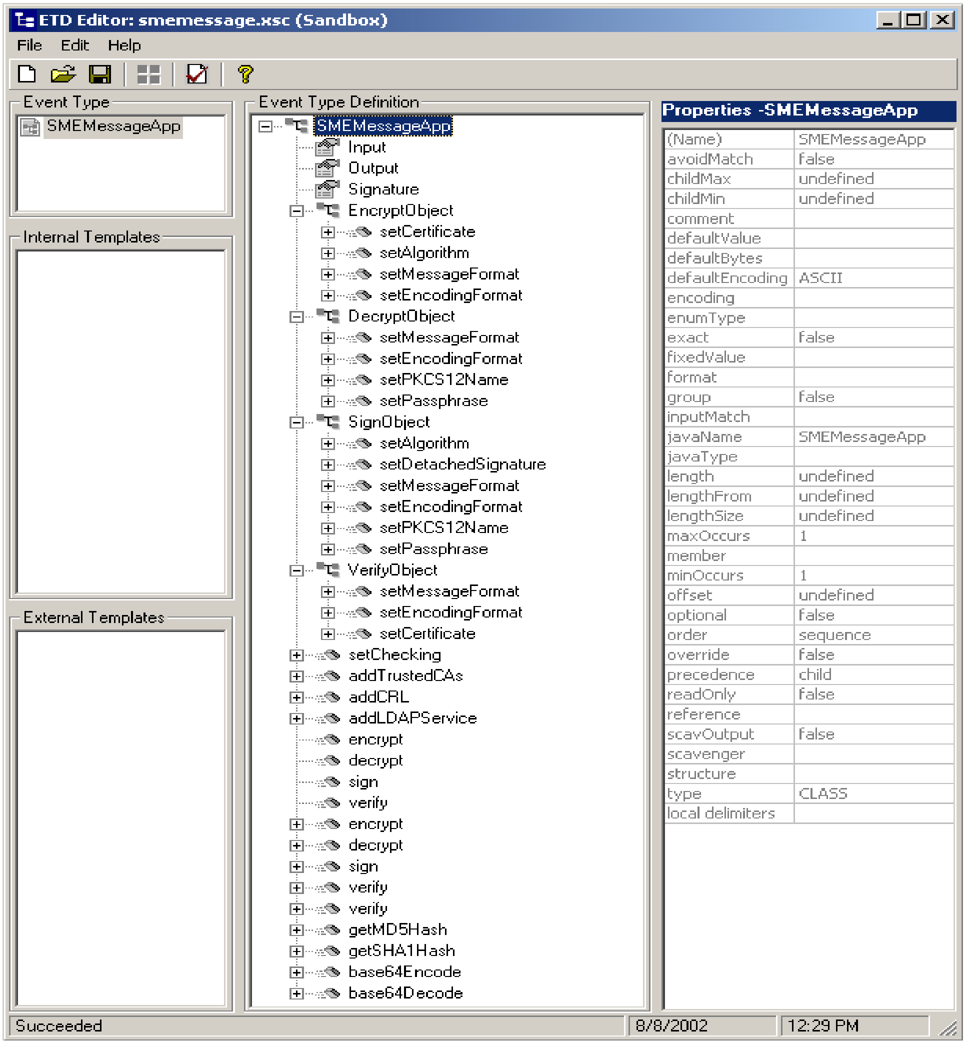
This chapter describes the structure of the SME ETD.

5.1 Understanding the Structure of the SME ETD

The SME Event Type Definition (ETD) exposes the APIs used in the e*Gate Java Collaboration environment. There are two components to the SME ETD: the **smemessage.xsc** file, which exposes the structures and methods, and the Java classes, which implement those structures and methods.

The SME installation includes the file **smemessage.xsc**. This file represents an ETD to use for encryption, decryption, signing, and verify. The following section describes the SMEMessageApp ETD, as shown in **Figure 10 on page 45**.

Figure 10 SMEMessageApp ETD in the ETD Editor



```
SMEMessageApp
    Input
    Output
    Signature
    EncryptObject
        setCertificate
        setAlgorithm
        setMessageFormat
        setEncodingFormat
    DecryptObject
        setMessageFormat
        setEncodingFormat
        setPKCS12Name
        setPassphrase
    SignObject
        setAlgorithm
        setDetachedSignature
        setMessageFormat
        setEncodingFormat
        setPKCS12Name
        setPassphrase
    VerifyObject
        setMessageFormat
        setEncodingFormat
        setCertificate
    setChecking
    addTrustedCAs
    addCRL
    addLDAPService
    encrypt
    decrypt
    sign
    verify
    encrypt
    decrypt
    sign
    verify
    verify
    getMD5Hash
    getSHA1Hash
    base64Encode
    base64Decode
```

(Name)	SMEMessageApp
avoidMatch	false
childMax	undefined
childMin	undefined
comment	
defaultValue	
defaultBytes	
defaultEncoding	ASCII
encoding	
enumType	
exact	false
fixedValue	
format	
group	false
inputMatch	
javaName	SMEMessageApp
javaType	
length	undefined
lengthFrom	undefined
lengthSize	undefined
maxOccurs	1
member	
minOccurs	1
offset	undefined
optional	false
order	sequence
override	false
precedence	child
readOnly	false
reference	
scavOutput	false
scavenger	
structure	
type	CLASS
local delimiters	

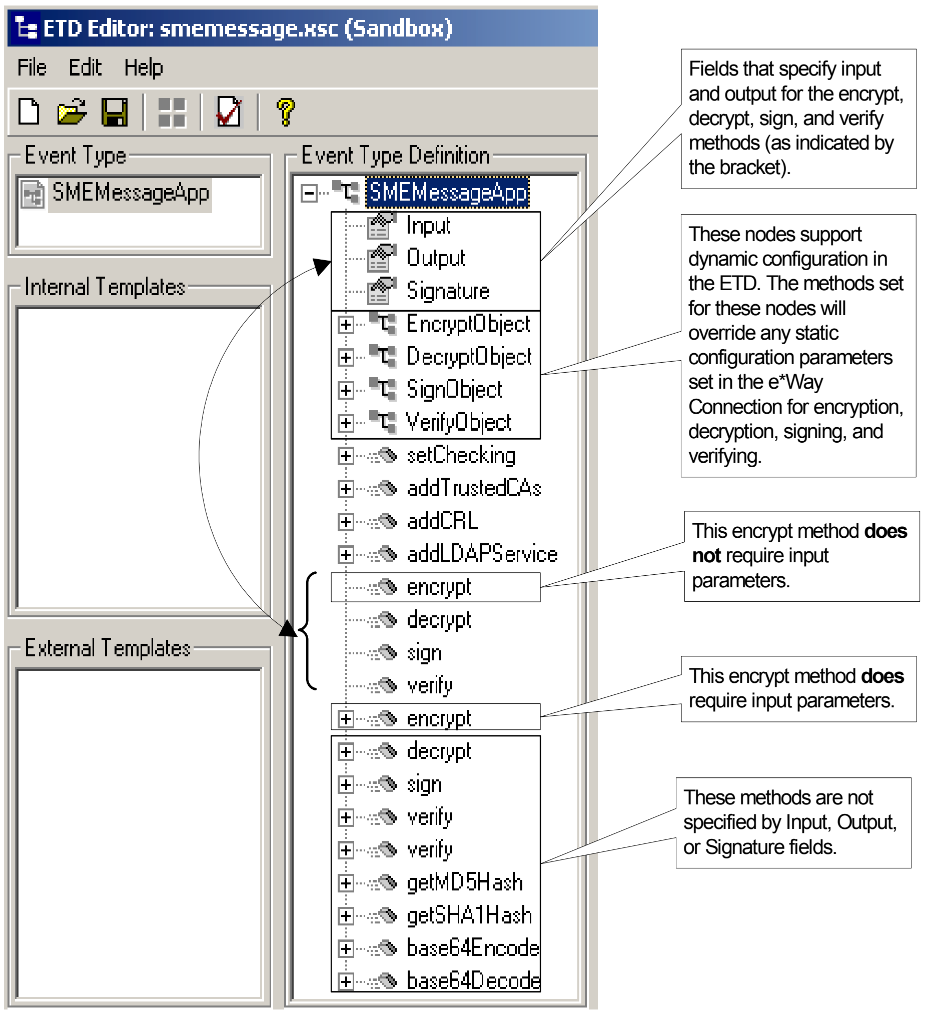
5.1.1 SMEMessageApp Root Node

SMEMessageApp is the root node and provides a graphical representation of the interface. Expanding the node reveals all the methods and attributes on the interface, which are themselves represented as nodes. A node representing a method is normally expandable and reveals all the parameters for the method, as well as the return value (if present).

Note: *The SMEMessageApp ETD methods are described in detail in* **Chapter 7**.

Figure 11 shows a detailed close-up of the nodes and methods of the SMEMessageApp ETD in the e*Gate Enterprise Manager ETD Editor Main dialog.

Figure 11 SMEMessagAppETD Nodes and Methods



Fields that specify input and output for the encrypt, decrypt, sign, and verify methods (as indicated by the bracket).

These nodes support dynamic configuration in the ETD. The methods set for these nodes will override any static configuration parameters set in the e*Way Connection for encryption, decryption, signing, and verifying.

This encrypt method **does not** require input parameters.

This encrypt method **does** require input parameters.

These methods are not specified by Input, Output, or Signature fields.

Important: *The Input field of the SMEMessageApp ETD must be set to the filename which contains the message.*

Chapter 6

Implementation

This chapter summarizes the procedures required for implementing a working system incorporating SME. It also provides instructions on creating sample schemas. The sample schemas will enhance your understanding of how to implement SME in a production environment.

The following assumptions apply to this implementation: 1) SME has been successfully installed. 2) The executable and the configuration files have been appropriately assigned. 3) All necessary .jar files are accessible.

See the *e*Gate Integrator User's Guide* for additional information.

6.1 Overview

This chapter takes you through each step required to implement the SME sample schemas. The sample schemas contain Collaborations, which link different data or Event types and Intelligent Queues. Other e*Way types are also used as components of the schemas.

The sample schemas can be used to test your system following installation and, if appropriate, as a template that you can modify to produce your own schema.

6.1.1 Pre-Implementation Tasks

Installing SeeBeyond Software

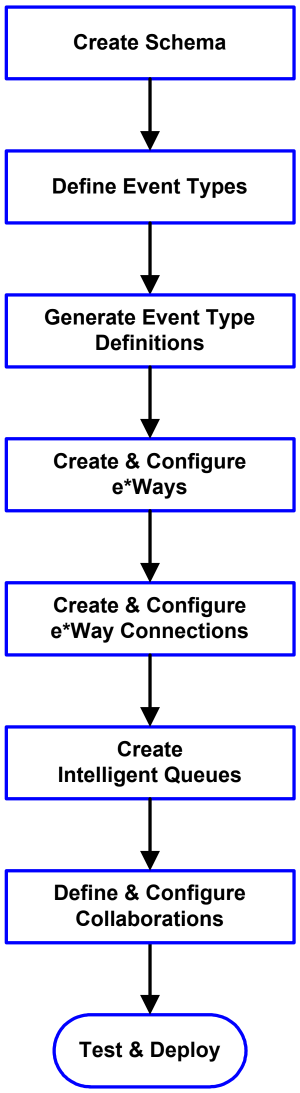
The first task is to install the SeeBeyond software as described in **Chapter 2**.

Importing the Sample Schemas

To use the sample schemas supplied with the SME Add-on, the schema files must be imported from the installation CD-ROM (see **“Sample Schemas” on page 50**).

Note: *It is highly recommended that you make use of the sample schemas to familiarize yourself with the operation of SME, test your system, and use as templates for your working schemas.*

6.1.2 Implementation Sequence


Create Schema
Define Event Types
Generate Event Type Definitions
Create & Configure e*Ways
Create & Configure e*Way Connections
Create Intelligent Queues
Define & Configure Collaborations
Test & Deploy

1 The first step is to create a new schema—the subsequent steps apply only to this schema (see **Creating Schemas** on page 50).

2 The second step is to define the Event Types you are transporting and processing within the schema (see **Creating Event Types** on page 51).

3 The third step is to create and configure the required e*Ways (see **Creating and Configuring the e*Ways** on page 52).

4 The fourth step is to configure the e*Way Connections (see **Creating the e*Way Connection** on page 57).

5 The fifth step is to create Intelligent Queues to hold published Events (see **Creating and Modifying Intelligent Queues** on page 59).

6 Next you need to define and configure the Collaborations between Event Types (see **Creating Collaboration Rules** on page 59).

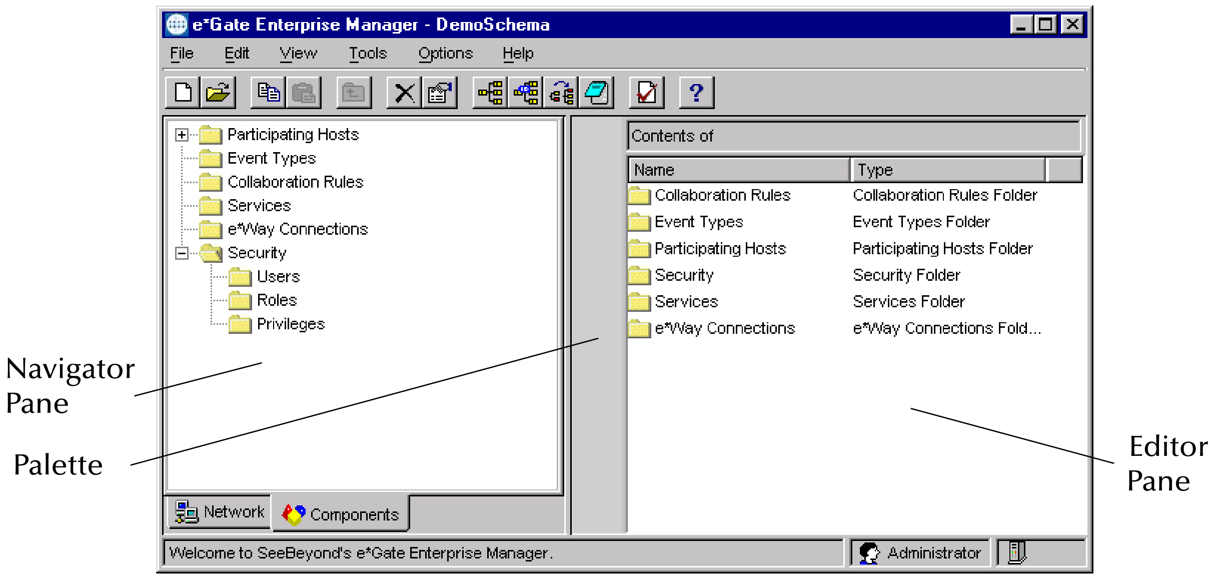
7 Finally, test your schema. Once you have verified that it is working correctly, you may deploy it to your production environment.

6.1.3 Using the e*Gate Enterprise Manager

Note: *The e*Gate Enterprise Manager GUI runs only on the Windows operating system.*

This section provides an overview of the e*Gate Enterprise Manager. The general features of the e*Gate Enterprise Manager dialog are shown in Figure 12. See the *e*Gate Integrator User's Guide* for a detailed description of the features and use of the Enterprise Manager.

Figure 12 e*Gate Enterprise Manager dialog (Components View)



Name	Type
Collaboration Rules	Collaboration Rules Folder
Event Types	Event Types Folder
Participating Hosts	Participating Hosts Folder
Security	Security Folder
Services	Services Folder
e*Way Connections	e*Way Connections Fold...

Use the Navigator and Editor panes to view the e*Gate components. You may only view components of a single schema at one time, and all operations apply only to the current schema. All procedures in this chapter should be performed while displaying the Components Navigator pane.

6.2 SME Sample Implementations

During installation, the host and Control Broker are automatically created and configured. The default name of each is the name of the host on which you are installing the e*Gate Enterprise Manager GUI. To complete the implementation of SME, do the following:

- Make sure that the Control Broker is activated.
- In the e*Gate Enterprise Manager, define and configure the following as necessary:
  - Inbound e*Way using **stcewfile.exe**
  - Outbound e*Way using **stcewfile.exe**
  - The Multi-Mode e*Way component as described in “Multi-Mode e*Way Configuration” on page 52
  - Event Type Definitions used to package the data to be exchanged with the external system.
  - Collaboration Rules to process Events.
  - The e*Way Connection as described in **Chapter 4**.
  - Collaborations, to be associated with each e*Way component, to apply the required Collaboration Rules.

- The destination to which data will be published prior to being sent to the external system.

Note: *If you planning to customize the smeinputmsg.dtd file, you will need to use the XML Tool Kit to convert the file to an .xml format. If you chose to use Customer ETDs, do not use the included .xml formatted data as your input data.*

The following sections describe how to define and associate each of the above components.

6.2.1 Creating Schemas

A schema is the structure that defines e*Gate system parameters and the relationships between components within the e*Gate system. Schemas can span multiple hosts.

All setup and configuration operations take place within an e*Gate schema. A new schema must be created, or an existing one must be started before using the system. Schemas store all their configuration parameters in the e*Gate Registry.

Sample Schemas

The sample schemas .zip files are provided for SME in the ..\samples\ewsme\.. directory on the Installation CD-ROM. Table 5 lists the filenames and purpose of each sample.

Table 5 SME Sample Schemas

File name	Purpose
SMEstatic.zip	Demonstrates static configuration parameters, as set in the e*Way Connection.
SMEdynamic.zip	Demonstrates dynamic configuration parameters, as set in the Collaboration ETD Editor.

For each of the sample schemas, the components are created when each schema is imported. The only changes required are changes to the configuration parameters of the e*Ways and e*Way Connections for your specific system. However, to help you learn how to implement SME, the rest of this chapter describes how the sample components are created manually.

The ***SMEstatic*** sample schema (SMEstatic.zip) is used as an example in all of the following sections. This sample schema demonstrates static configuration, where the parameters for encryption, decryption, signing, and verifying are set in the e*Way Connection.

The ***SMEdynamic*** sample schema (SMEdynamic.zip) is similar to the ***SMEstatic*** sample schema and contains all the same components. However, the ***SMEdynamic*** schema demonstrates dynamic configuration, where the parameters for encryption, decryption, signing, and verifying are set in the Collaboration Rules Editor.

To implement the ***SMEdynamic*** sample schema, follow the same steps described in the rest of this chapter and see **"Dynamic Configuration in the Collaboration Rules Editor" on page 71** for information specific to using dynamic configuration.

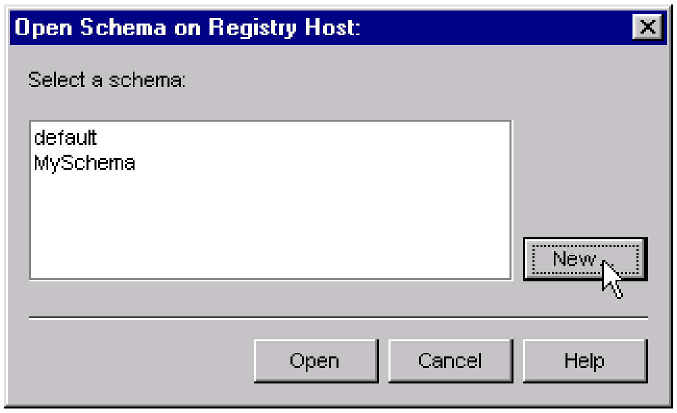
The following sections explain how the components for both of the SME sample schemas are created.

To create a new schema

The first task in deploying the sample implementation is to create a new schema name. While it is possible to use the default schema for the sample implementation, it is recommended that you create a separate schema for testing purposes. After you install SME, do the following:

1. Start the e*Gate Enterprise Manager GUI.
2. When the Enterprise Manager prompts you to log in, select the host that you specified during installation, and enter your password.
3. You will then be prompted to select a schema, as shown in Figure 13. Click **New**.

Figure 13 Open Schema Dialog



4. Enter a name for the new schema. In this case, enter **SMEdynamic**, or any name as desired.
5. To import the sample schema select **Create from Export**, and use **Find** to locate and select the sample .zip file on the CD-ROM.
6. Click **Open**.

The e*Gate Enterprise Manager opens under your new schema. You are now ready to begin creating the necessary components for this sample schema.

6.2.2 Creating Event Types

Within e*Gate, messages and/or packages of data are defined as Events. Each Event must be categorized into a specific Event Type within the schema.

The SME installation includes the file **smemessage.xsc** which represents a standard SME Event Type Definition template.

As the name implies, an Event Type Definition (ETD) defines the structure of the Event Types employed in your schema. Any one ETD can be associated with more than one Event Type within the schema.

Creating an Event Type and Associating an Existing .xsc

For the purpose of this example, the following procedure shows how to associate an **Event Type** with an existing .xsc file using **smemessage.xsc**. The **smemessage.xsc** comes with SME and is used when creating all schemas.

1. Select the **Event Types** folder on the **Components** tab of the e*Gate Navigator.
2. On the palette, click **Create a New Event Type**.
3. Enter the name of the **Event Type** in the **New Event Type Component** dialog, then click **OK**. (For this sample, the Event Type is defined as "**smemessage**").
4. Double-click the new **Event Type** to edit its properties.
5. When the **Properties** dialog opens, click **Find**.
6. Browse to and select **smemessage.xsc** (provided as the default destination **.xsc** file).
7. Click **Apply** and **OK** to close the Event Type Properties dialog box.

6.2.3 Creating and Configuring the e*Ways

The first components to be created are the following e*Ways.

- **"Creating the Inbound e*Way" on page 52**
- **"Creating the Outbound e*Way" on page 54**
- **"Creating the Multi-Mode e*Way (ew_java)" on page 55**

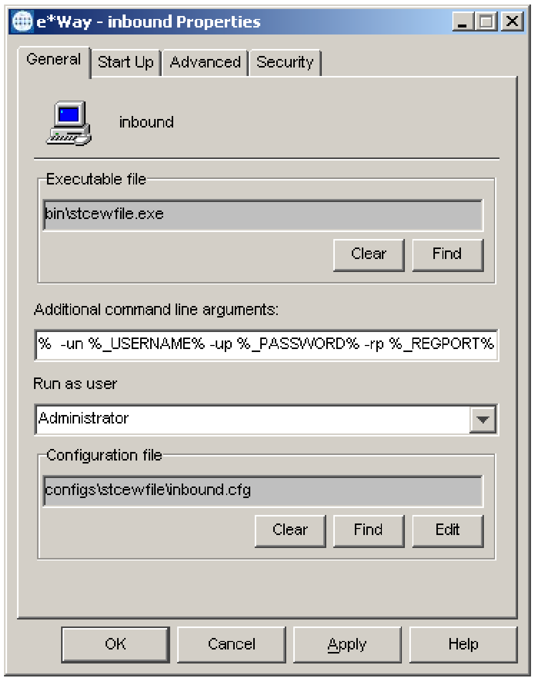
The ***SMEstatic*** sample contains three e*Ways, two of which are pass-through (inbound and outbound) and one Multi-Mode (ew_java).

The following sections provide instructions for creating each e*Way.

Creating the Inbound e*Way

1. Select the Navigator's **Components** tab.
2. Open the host on which you want to create the e*Ways.
3. Select the **Control Broker** that will manage the new e*Ways.
4. On the palette, click **Create a New e*Way**.
5. Enter the name of the new e*Way (in this case "**inbound**"**)**, then click **OK**.
6. Right-click the new e*Way and select **Properties** to edit its properties.
7. The e*Way Properties dialog opens, as shown in Figure 14.

Figure 14 e*Way Properties - inbound



inbound

Executable file: bin\stcewfile.exe

Additional command line arguments: % -un %_USERNAME% -up %_PASSWORD% -rp %_REGPORT%

Run as user: Administrator

Configuration file: configs\stcewfile\inbound.cfg

8 Click **Find** beneath the **Executable File** field, and select **stcewfile.exe** as the executable file.

9 Under the **Configuration File** field, click **New**. The dialog opens. Select the settings shown in Table 6 for this configuration file.

Table 6 Configuration Parameters for the Inbound e*Way

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
AllowIncoming	YES
AllowOutgoing	NO
Outbound Settings	Default
Poller Inbound Settings	
PollDirectory	C:\egate\client\data\input
InputFileExtension	*.fin (input file extension)
PollMilliseconds	1000
Remove EOL	YES
MultipleRecordsPerFile	NO
MaxBytesPerLine	4096
BytesPerLineIsFixed	NO
File Records Per eGate Event	1
Performance Testing	Default

10 After selecting the desired parameters, save the **configuration** file (as "**inbound.cfg**").

11 From the **File** menu, click **Promote to Run Time**. This closes the .cfg file.

12 In the e*Way - Properties dialog, use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each e*Way you configure.

  A Use the **Startup** tab to specify whether the e*Way starts automatically, or restarts after abnormal termination or due to scheduling, and so forth.

  B Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.

  C Use **Security** to view or set privilege assignments.

13 Select **OK** to close the e*Way Properties dialog.

Creating the Outbound e*Way

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the e*Ways.

3 Select the **Control Broker** that will manage the new e*Ways.

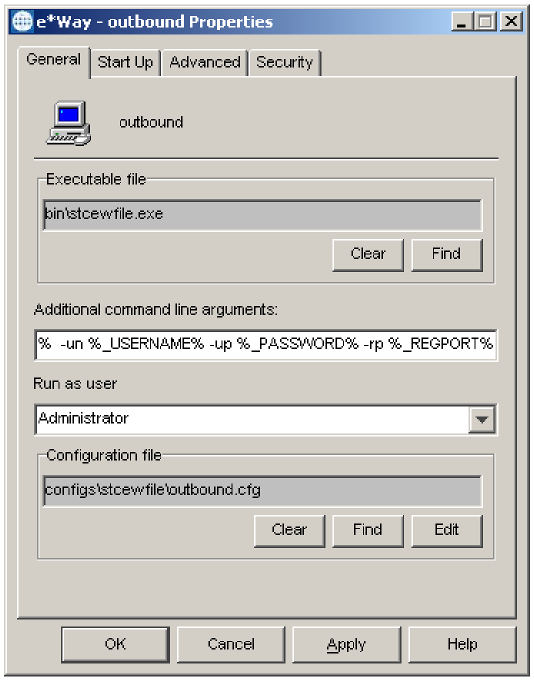
4 On the palette, click **Create a New e*Way**.

5 Enter the name of the new e*Way (in this case "**outbound**"), then click **OK**.

6 Select the new e*Way, right-click and select **Properties** to edit its properties.

7 The e*Way Properties dialog opens, as shown in Figure 15.

Figure 15 e*Way Properties - outbound



outbound

Executable file: bin\stcewfile.exe

Additional command line arguments: % -un %_USERNAME% -up %_PASSWORD% -rp %_REGPORT%

Run as user: Administrator

Configuration file: configs\stcewfile\outbound.cfg

8 Click **Find** beneath the **Executable File** field. Select **stcewfile.exe** as the executable file.

9 Under the **Configuration File** field, click **New**. The **Edit Settings** dialog opens. Select the settings shown in Table 7 for this configuration file.

:

Table 7 Configuration Parameters for the Outbound e*Way

Parameter	Value
General Settings (unless otherwise stated, leave settings as default)	
AllowIncoming	NO
AllowOutgoing	YES
Outbound Settings	
OutputDirectory	C:\egate\client\data\output
OutputFileName	output%d.dat
MultipleRecordsPerFile	NO
MaxRecordsPerFile	10000
AddEOL	YES
Poller Inbound Settings	Default
Performance Testing	Default

10 Save the .cfg file (**outbound.cfg**), and from the **File** menu, click **Promote to Run Time** to close the Edit Settings dialog.

11 In the e*Way - Properties dialog, use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for the e*Way.

12 Use **Security** to view or set privilege assignments.

13 Click **OK** to close the e*Way Properties dialog.

Creating the Multi-Mode e*Way (ew_java)

1 Select the Navigator's **Components** tab.

2 Open the host on which you want to create the e*Way.

3 Select the **Control Broker** that will manage the new e*Way.

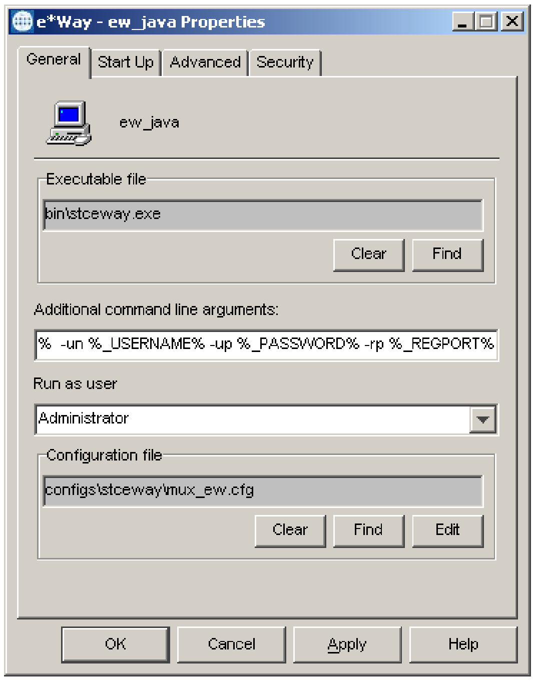
4 On the palette, click **Create a New e*Way**.

5 Enter the name of the new e*Way (in this case, "**ew_java**"), then click **OK**.

6 Right-click the new e*Way and select **Properties** to edit its properties.

7 The e*Way Properties dialog opens, as shown in Figure 16.

Figure 16 e*Way Properties - ew_java



Executable file: bin\stceway.exe

Additional command line arguments: % -un %_USERNAME% -up %_PASSWORD% -rp %_REGPORT%

Run as user: Administrator

Configuration file: configs\stceway\mux_ew.cfg

8 Click **Find** beneath the **Executable File** field, and select **stceway.exe** as the executable file.

9 To edit the JVM Settings, select **New** (or **Edit** if you are editing the existing .cfg file) under Configuration file.

Table 8 lists the parameters and values for the Multi-Mode e*Way. See **“Multi-Mode e*Way Configuration” on page 15** for details on the parameters associated with the Multi-Mode e*Way.

:

Table 8 Configuration Parameters for the Multi-Mode e*Way

Parameter	Value
JVM Settings (unless otherwise stated, leave settings as default)	
JNI DLL absolute pathname	C:\eGate\client\bin\Jre\jvm.dll (or absolute path to proper JNI DLL)
CLASSPATH Prepend	C:\eGate\client\classes\stcsme.jar C:\eGate\client\classes\ThirdParty\baltimore\classes\smime.jar C:\eGate\client\classesThirdParty\baltimore\classes\KeyToolsPro_All_1.2.jar (or absolute path to stcsme.jar, smime.jar, and KeyToolsPro_All1.2.jar)

10 Save the .cfg file (**mux_ew.cfg**), and from the **File** menu, click **Promote to Run Time**.

11 In the e*Way Properties dialog, use the **Startup**, **Advanced**, and **Security** tabs to modify the default settings for each.

A. Use the **Startup** tab to specify whether the e*Way starts automatically, restarts after abnormal termination or due to scheduling, etc.

B. Use the **Advanced** tab to specify or view the activity and error logging levels, as well as the Event threshold information.

C. Use **Security** to view or set privilege assignments.

12. Click **OK** to close e*Way Properties dialog.

For more information on the Multi-Mode e*Way configuration settings see the *e*Gate Integrator User's Guide*.

6.2.4. Creating the e*Way Connection

The e*Way Connection configuration file contains the connection information along with the information needed to communicate using SME.

To create and configure a new e*Way Connection

1. Select the **e*Way Connection** folder on the **Components** tab of the e*Gate Navigator.
2. On the palette, click **Create a New e*Way Connection**.
3. Enter the name of the e*Way Connection (for this sample, "**con_sme**"), then click **OK.**
4. Double-click the new e*Way Connection to edit its properties.
5. The e*Way Connection Properties dialog opens. Select **SME** from the **e*Way Connection Type** drop-down menu.
6. Enter the **Event Type "get" interval** in the dialog box provided. 10000 milliseconds is the configured default. The "get interval is the intervening period at which, when subscribed to, the e*Way Connection is polled.
7. Under e*Way Connection Configuration File, click **New**.
8. The e*Way Connection Editor opens. Select the following parameters listed in Table 9. For more information on SME Connection parameters, see **"e*Way Connection Configuration" on page 27**.

Table 9 e*Way Connection Configuration Parameters

Parameter	Value
connector (unless otherwise stated, leave settings as default)	
type	SME
class	com.stc.eway.SMEClientConnector
encrypt	
Certificate	C:\egate\client\certs\seebeyond-test-user-2-cert.der
format	DER
algorithm	DES_EDE3_CBC

Table 9 e*Way Connection Configuration Parameters

Parameter	Value
MessageFormat	SMIME2
EncodingFormat	BASE64
decrypt	
MessageFormat	SMIME2
EncodingFormat	BASE64
PKCS12	C:\egate\client\keys\seebeyond-test-user-2-3des.p12
PassPhrase	****
sign	
algorithm	RSA_SHA1
detached	YES
MessageFormat	SMIME2
EncodingFormat	BASE64
PKCS12	C:\egate\client\keys\seebeyond-test-user-1-3des.p12
PassPhrase	****
verify	
MessageFormat	SMIM2
EncodingFormat	BASE64
Certificate	C:\egate\client\certs\seebeyond-test-user-1-cert.der
format	DER
Certificate	
Checking	CA
TrustedCA	C:\egate\client\certs\CA\stc-test-ca.cer
format	PEM
CRL	
filename	
CACRLCertificate	
CACRLformat	DER
CACertificate	
format	DER

9 Save the .cfg file (**con_sme.cfg**)and click **File**, **Promote to Run Time**.

6.2.5 Creating and Modifying Intelligent Queues

The next step is to create and modify **Intelligent Queues** (**IQs**). IQs manage the exchange of information between components within the e*Gate system, providing non-volatile storage for data as it passes from one component to another.

IQs use IQ Services to transport data. IQ Services provide the mechanism for moving Events between IQs, handling the low-level implementation of data exchange (such as system calls to initialize or reorganize a database).

For more information on how to add and configure IQs and IQ Managers, see the *e*Gate Integrator System Administration and Operations Guide*.

To create and modify an Intelligent Queue for SME

1. Select the Navigator's **Components** tab.
2. Open the host on which you want to create the IQ.
3. Open a **Control Broker**.
4. Select an **IQ Manager**.
5. On the palette, click **Create a New IQ**.
6. Enter the name of the new **IQ** (in this case **iq1**), then click **OK.**
7. Double-click the new **IQ** to edit its properties.
8. On the **General** tab, specify the **Service** and the **Event Type Get Interval**.

   The SeeBeyond Standard IQ Service provides sufficient functionality for most applications. If specialized services are required, custom IQ Service DLLs may be created.

   The default **Event Type Get Interval** of 100 Milliseconds is satisfactory for the purposes of this initial implementation.
9. On the **Advanced** tab, make sure that **Simple publish/subscribe** is checked under the **IQ behavior** section.
10. Click OK to close the **IQ Properties** dialog
11. For this schema, repeat steps 1 through 10 to create an additional IQ (**iq2**).

6.2.6 Creating Collaboration Rules

The next step is to create the Collaboration Rules that will extract and process selected information from the source Event Type defined above, according to its associated Collaboration Service.

To create Collaboration Rules files

1. From the **Enterprise Manager Task Bar,** select **Options** and click **Default Editor**.
2. Though the **Default Editor** can be set to either **Monk** or **Java**, the default should be set to **Java**.
3. The sample schema requires creating three Collaboration Rules files:
   - Passin (Pass Through)

- Passout (Pass Through)
- cor_java (Java)

Pass Through (inbound)

1 Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

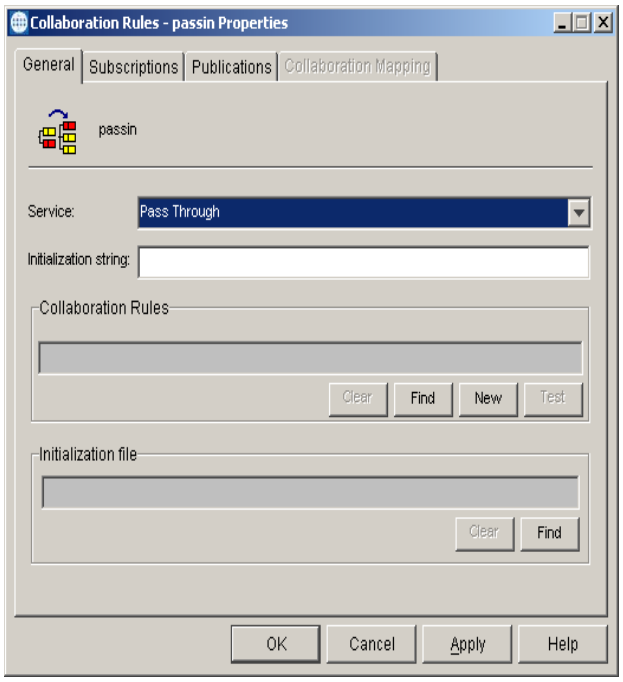
2 In the **Navigator**, select the **Collaboration Rules** folder.

3 On the palette, click **Create New Collaboration Rules**.

4 Enter the name of the new Collaboration Rule Component, then click **OK** (for this case, use **Passin**).

5 Double-click the new Collaboration Rules Component. The **Collaboration Rules Properties** dialog appears, as shown in Figure 17.

Figure 17 Collaboration Properties



passin

Service: Pass Through

Initialization string:

6 The **Service** field defaults to **Pass Through**.

7 Go to the **Subscriptions** tab. Select **input** under **Available Input Event Types**, and click the right arrow to move it to **Selected Input Event Types**. The box under **Triggering Event** should be selected.

8 Go to the **Publications** tab. Select **input** under **Available Output Event Types**, and click the right arrow to move it to **Selected Output Event Types**. The Option button under **Default** is enabled.

9 Click **OK** to close the **Collaboration Rules - passin Properties** dialog.

Pass Through (outbound)

1 Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2 In the **Navigator**, select the **Collaboration Rules** folder.

3 On the palette, click **Create New Collaboration Rules**.

4 Enter the name of the new Collaboration Rule Component, then click **OK** (for this case, use **Passout**).

5 Double-click the new Collaboration Rules Component. The **Collaboration Rules Properties** dialog opens.

6 The **Service** field defaults to **Pass Through**.

7 Go to the **Subscriptions** tab. Select **outevent** under **Available Input Event Types**, and click the right arrow to move it to **Selected Input Event Types**. The box under **Triggering Event** should be selected.

8 Go to the **Publications** tab. Select **outevent** under **Available Output Event Types**, and click the right arrow to move it to **Selected Output Event Types**. The Option button under **Default** is enabled.

9 Click **OK** to close the **Collaboration Rules - passin Properties** dialog.

10 Click **OK** to close the **Collaboration Properties** dialog.

Java (cor_java)

1 Select the Navigator's **Components** tab in the e*Gate Enterprise Manager.

2 In the **Navigator**, select the **Collaboration Rules** folder.

3 On the palette, click **Create New Collaboration Rules**.

4 Enter the name of the new Collaboration Rule, then click **OK** (for this case, use **cor_java**).

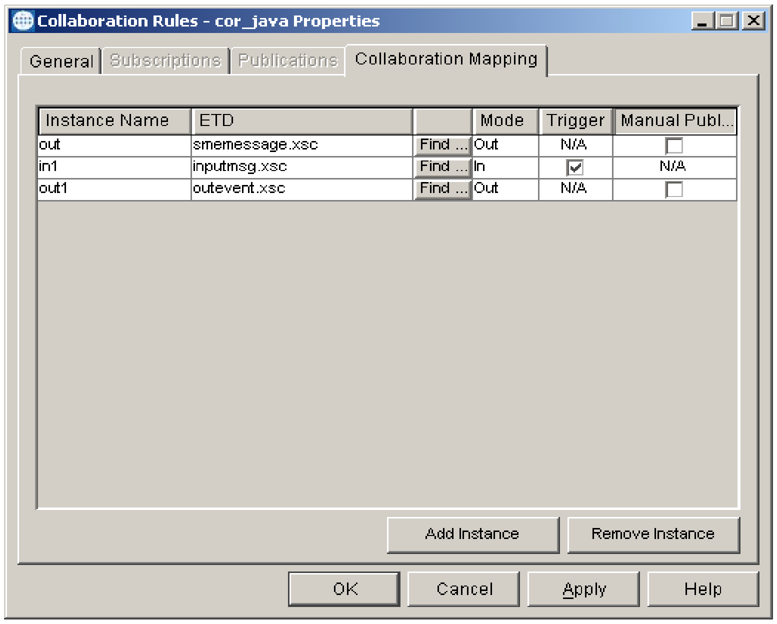
5 Double-click the new Collaboration Rules Component to edit its properties. The **Collaboration Rules Properties** dialog opens.

6 From the **Service** field drop-down box, select **Java**. The **Collaboration Mapping** tab is now enabled, and the **Subscriptions** and **Publications** tabs are disabled.

7 In the **Initialization string** box, enter any required initialization string that the Collaboration Service may require. This field can be left blank.

8 Select the **Collaboration Mapping** tab, as shown in Figure 18.

Figure 18 Collaboration Rules - Collaboration Mapping Properties



Instance Name	ETD		Mode	Trigger	Manual Publ...
out	smemessage.xsc	Find ...	Out	N/A	☐
in1	inputmsg.xsc	Find ...	In	☑	N/A
out1	outevent.xsc	Find ...	Out	N/A	☐

9 Using the **Add Instance** button, create instances to coincide with the Event Types.

For this sample, do the following:

10 In the Instance Name column, enter **in1** for the instance name.

11 Click **Find**, navigate to **etd\inputmsg.xsc**, double-click to select.

inputmsg.xsc is added to the ETD column of the instance row.

12 In the Mode column, select **In** from the drop–down menu available.

13 In the Trigger column, select the box to enable trigger mechanism.

14 Repeat steps 9–13 using the following values:

- Instance Name — **out1**
- ETD — **outevent.xsc**
- Mode — **Out**
- Trigger — do not select

15 Repeat steps 9–13 again using the following values:

- Instance Name — **out**
- ETD — **smemessage.xsc**
- Mode — **Out**
- Trigger — do not select

Select the **General** tab, under the Collaboration Rule box, select **New**. The **Collaboration Rules Editor** opens.

16 Expand to full size for optimum viewing, expanding the Source and Destination Events as well.

The following section describes the setting up the collaboration rules for **SME** using the Java Collaboration Rules Editor.

Creating the Collaboration Rules Class

Java Collaborations are defined using the e*Gate Java Collaboration Rules Editor. The file extension for Java Collaboration Rules is **.xpr**. See the *e*Gate Integrator User's Guide* for descriptions of the Java Collaboration Rules Editor and its use.

Note: *The Java Collaboration environment supports multiple source and destination ETDs.*

This section provides an example of how to create the Collaboration Rules Class using the Java Collaboration Rules Editor. The completed Collaboration Rules .xpr file is included with the sample schema on the CD. The following section gives a number of examples that demonstrate how these rules were setup. Refer to the completed class, **SME.class** when completing the Collaboration Rules Properties.

Creating Business Rules for the Sample Schema using the Collaboration Rules Editor

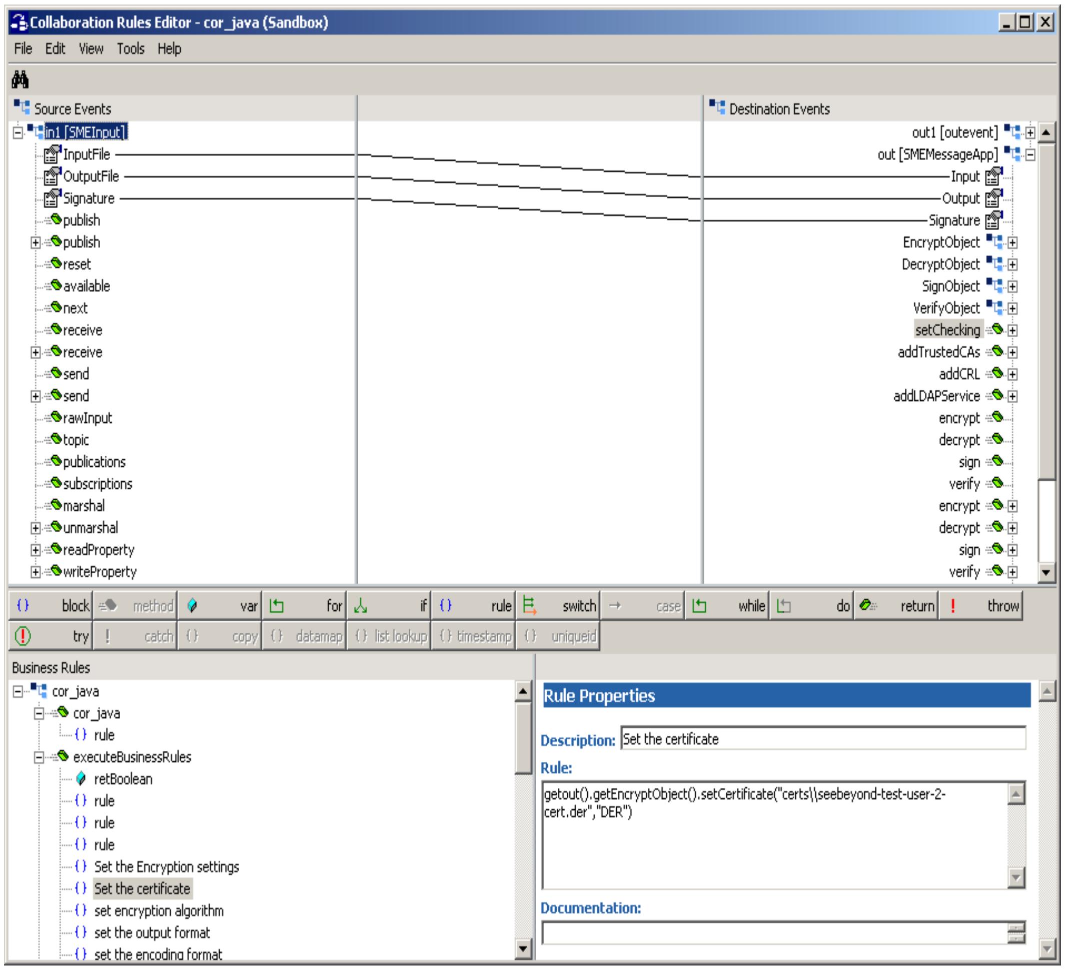
cor_java

Each rule is created by clicking the rule button on the Business Rules toolbar or by dragging and dropping an object from the Source Events pane to an object in the Destination Events pane. Descriptions are added by typing the desired description in the Description field of the Properties dialog.

The cor_java Collaboration Rules, shown in **Figure 19 on page 64**, are created as follows:

1. When the Collaboration Rules Editor opens maximize the dialog and expand the Source Events and Destination Events command nodes to display available nodes and methods.
2. Select **retBoolean** in the Business Rules pane of the Collaboration Rules Editor. All of the user–defined business rules are added as part of this method.
3. Select **InputFile** from the **Source Events** pane. Drag and drop onto **Input** in the **Destination Events** pane. A connecting line appears between the properties objects.
4. In the **Business Rules** pane, a rule expression appears, with the properties of that rule displayed in the **Rule Properties** pane.
5. Select **OutputFile** from the **Source Events** pane. Drag and drop onto **Output** in the **Destination Events** pane.
6. Select **Signature** from the **Source Events** pane. Drag and drop onto **Signature** in the **Destination Events** pane.

Figure 19 Collaboration Rules Editor— cor_java



Description: Set the certificate

Rule:

```
getout().getEncryptObject().setCertificate("certs\\seebeyond-test-user-2-cert.der","DER")
```

Documentation:

7 From the Collaboration Rules Editor toolbar, click on **rule**. This places a **rule** “space” in the Business Rules pane, to which the user can add the Java expression. A **rule** space is now available under **retBoolean** in the Business Rules dialog. Select the new **rule**.

8 Choose the appropriate method (encrypt/sign/verify/decrypt) by dragging it to the Rule panel. This enters the Java code in the Rule Properties, Rules dialog.

For example, to add an encryption rule, from the Destination Event:

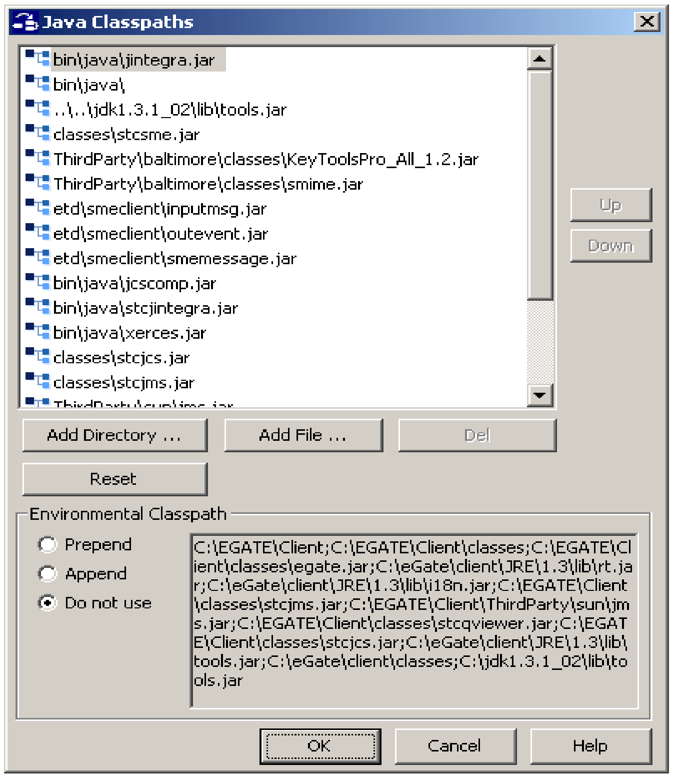
```
getout1().setData(new String(getout().encrypt()))
```

9 When the Collaboration Rules are completed, from the **File** menu, click **Compile** to compile the new collaboration.

10 Before compiling the code, from the **Tools** menu, click **Options**.

11 Verify that all necessary **.jar** files are included. Add **stcsme.jar** (see Figure 20).

Figure 20 Business Rules



```
bin\java\jintegra.jar
bin\java\
..\..\jdk1.3.1_02\lib\tools.jar
classes\stcsme.jar
ThirdParty\baltimore\classes\KeyToolsPro_All_1.2.jar
ThirdParty\baltimore\classes\smime.jar
etd\smeclient\inputmsg.jar
etd\smeclient\outevent.jar
etd\smeclient\smemessage.jar
bin\java\jcscomp.jar
bin\java\stcjintegra.jar
bin\java\xerces.jar
classes\stcjcs.jar
classes\stcjms.jar
```

Environmental Classpath

( ) Prepend
( ) Append
[✓ Do not use]

```
C:\EGATE\Client;C:\EGATE\Client\classes;C:\EGATE\Cl
ient\classes\egate.jar;C:\eGate\client\JRE\1.3\lib\rt.ja
r;C:\eGate\client\JRE\1.3\lib\i18n.jar;C:\EGATE\Client
\classes\stcjms.jar;C:\EGATE\Client\ThirdParty\sun\jm
s.jar;C:\EGATE\Client\classes\stcqviewer.jar;C:\EGAT
E\Client\classes\stcjcs.jar;C:\eGate\client\JRE\1.3\lib\
tools.jar;C:\eGate\client\classes;C:\jdk1.3.1_02\lib\to
ols.jar
```

12 When all the business logic has been defined, the code can be compiled by selecting **Compile** from the **File** menu. The **Save** dialog appears, provide a name for the **.xpr** file. For the sample, use **cor_java.xpr**.

13 Click **Promote** to promote to runtime.

Important: *This is not a complete Collaboration, but an example of how the various components of the collaboration are setup. For the sample schema, select cor_java.class in the Collaboration Rules - cor_java Client Properties dialog box to use the completed cor_java xpr file. For further information on using the Collaboration Rules Editor see the* ***e*Gate Integrator User's Guide****.*

6.2.7 Creating the Sample Schema Collaborations

Collaborations are the components that receive and process Event Types and forward the output to other e*Gate components or to an external system. Collaborations consist of the Subscriber, which "listens" for Events of a known type (sometimes from a given source) and the Publisher, which distributes the transformed Event to a specified recipient. The same Collaboration cannot be assigned to more than one e*Gate component.

To create the SME_Multi_Mode Collaboration

1 In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select a **Control Broker.**

4 Select the **ew_java** e*Way to assign the Collaboration.

5 On the palette, click **Create a New Collaboration**.

6 Enter the name of the new Collaboration, then click **OK.** (For the sample, “**cl_java**”.)

7 Double click the new **Collaboration** to edit its properties.

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. (For the sample, “**cor_java**”.)

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

   A From the **Instance Name** list, select the Instance Name that you previously defined **in1.**

   B From the **Event Type** list, select the **Event Type** that you previously defined (**input**).

   C Select the **Source** from the **Source** list. In this case, it should be **cl_inbound**.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Instance Name** list, select the **Instance Name** that you previously defined **out1.**

   B From the **Event Types** list, select the **Event Type** that you previously defined (**outevent**).

   C Select the publication destination from the **Destination** list. In this case, it should be **iq2**.

11 In the **Publications** area, click **Add** again to define the output **Event Types** that this Collaboration will publish.

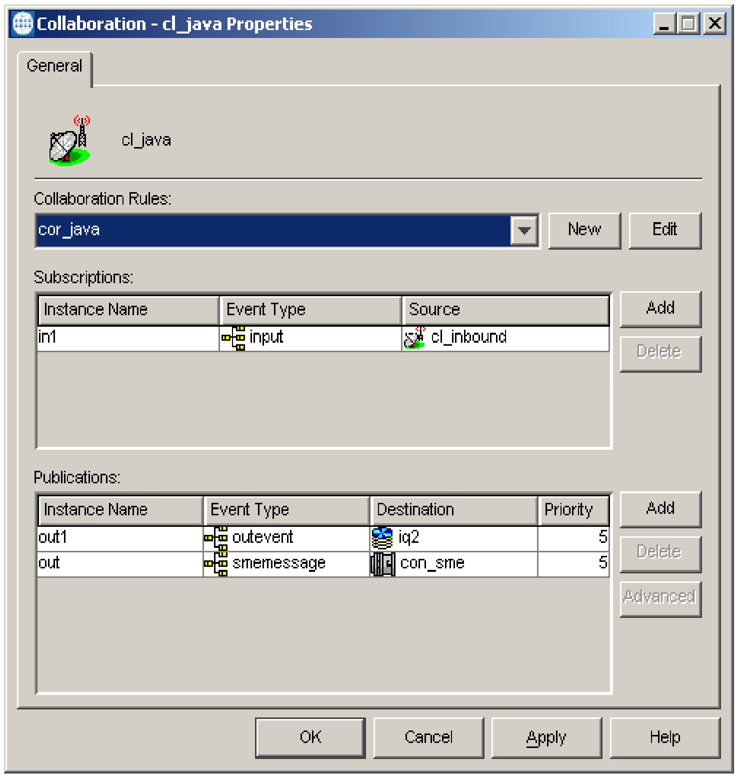
   A From the **Instance Name** list, select the **Instance Name** that you previously defined **out.**

   B From the **Event Types** list, select the **Event Type** that you previously defined (**smemessage**).

   C Select the publication destination from the **Destination** list. In this case, it should be **con_sme**.

The Collaboration for the ew_java e*Way appears as follows when complete (see **Figure 21 on page 67**).

Figure 21 Collaboration Properties



cl_java

Collaboration Rules: cor_java [New] [Edit]

Subscriptions: [Add] [Delete]

Instance Name	Event Type	Source
in1	input	cl_inbound

Publications: [Add] [Delete] [Advanced]

Instance Name	Event Type	Destination	Priority
out1	outevent	iq2	5
out	smemessage	con_sme	5

12 Click **OK** to exit.

To create the Inbound_eWay Collaboration

1 In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select a **Control Broker.**

4 Select the **inbound** e*Way to assign the Collaboration.

5 On the palette, click **Create a New Collaboration**.

6 Enter the name of the new Collaboration, then click **OK.** (For the sample, “Passin”.)

7 Double-click the new **Collaboration** to edit its properties.

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. (For the sample, “Passin”.)

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

A From the **Event Type** list, select the **Event Type** that you previously defined (**input**).

B Select the **Source** from the **Source** list. In this case, it should be <External>.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Event Types** list, select the **Event Type** that you previously defined (**input**).

   B Select the publication destination from the **Destination** list. In this case, it should be **iq1**.

11 Click **Apply** and click **OK** to close the Collaboration Properties dialog box.

To create the Outbound_eWay Collaboration

1 In the e*Gate Enterprise Manager, select the Navigator's **Components** tab.

2 Open the host on which you want to create the Collaboration.

3 Select a **Control Broker.**

4 Select the **Outbound_eWay** to assign the Collaboration.

5 On the palette, click **Create a New Collaboration**.

6 Enter the name of the new Collaboration, then click **OK.** (For the sample, “**Passout**”.)

7 Double-click the new **Collaboration** to edit its properties.

8 From the **Collaboration Rules** list, select the **Collaboration Rules** file that you created previously. (For the sample, “**Passout**”.)

9 In the **Subscriptions** area, click **Add** to define the input Event Types to which this Collaboration will subscribe.

   A From the **Event Type** list, select the **Event Type** that you previously defined (**outevent**).

   B Select the **Source** from the **Source** list. In this case, it should be **cl_java**.

10 In the **Publications** area, click **Add** to define the output **Event Types** that this Collaboration will publish.

   A From the **Event Types** list, select the **Event Type** that you previously defined (**outevent**).

   B Select the publication destination from the **Destination** list. In this case, it should be **<External>**.

11 Click **Apply** and click **OK** to close the Collaboration Properties dialog box.

6.3 Creating and Executing the Sample Schemas

6.3.1 SMEstatic Sample Schema

The previous sections provided the basics for implementing SME. This section describes how to use SME within the ***SMEstatic*** sample schema using static configuration. For a description of how the sample schema’s components are created in dynamic configuration, see **“Dynamic Configuration in the Collaboration Rules Editor” on page 71**.

Note: *It is assumed that SME has been installed properly, and that all of the necessary files and scripts are located in the default location.*

The ***SMEstatic*** sample uses the **smemessage.xsc** Event Type Definition (ETD), created using the Custom ETD Wizard. For more information on creating Event Types and ETDs see **"Creating Event Types" on page 51**.

The structure of the ETD used in this sample is shown as it appears in the ETD editor in **"Understanding the Structure of the SME ETD" on page 44**.

- The **inputmsg.xsc** is an input message that is published to an inbound Intelligent Queue.
- The **smemessage.xsc** assigns the functionality to encrypt, decrypt, sign, and verify the input message.
- The **outevent.xsc** outputs a message that is published by the outbound e*Way.

This implementation consists of two file-based e*Ways, one Multi-Mode e*Way, three Event Types, three Collaboration Rules, two Intelligent Queues and three Collaborations, as follows:

- **inbound** - This e*Way receives input from an external source, apply pass through Collaboration Rules, and publish the information to an Intelligent Queue.
- **ew_java** - This Multi-Mode e*Way applies extended Java Collaboration Rules to an inbound Event to perform the desired business logic, in this case encryption and decryption.
- **outbound -** This e*Way receives information from the Multi-Mode e*Way and publish to the external system.
- **smemessage** - This Event Type contains the methods to be used to perform the necessary transformation.
- **input** - This Event Type describes an Event that is input to the extended Java Collaboration Service.
- **outevent** - This Event Type describes an Event that contains the transformed data.
- **passin** - This Collaboration Rule is associated with the *inbound* e*Way, and is used for receiving the input Event.
- **cor_java** - The Collaboration Rule is associated with the *ew_java* Multi-Mode e*Way, and is used to perform the transformation process.
- **passout** - This Collaboration Rule is associated with the *outbound* e*Way, and is used to send the Event to an external file.
- **iq1** - This Intelligent Queue is a STC_Standard IQ, and forwards data to the *ew_java* Multi-Mode e*Way.
- **iq2** - This Intelligent Queue is a STC_Standard IQ, and forwards data to the *outbound* e*Way.

Executing the SMEstatic Sample Schema

To execute the ***SME*** sample schema, do the following:

1. After configuring all of the e*Way components associated with the sample, start the schema from the command line prompt, and enter the following:

   ```
   stccb -ln localhost_cb -rh localhost -rs <schema name>-un
       Administrator -up <password>
   ```

   Substitute ***hostname***, ***username***, ***schema name***, and ***user password*** as appropriate.
2. Exit from the command line prompt, and start the e*Gate Monitor GUI.
3. Start the component e*Ways (ew_java, inbound and outbound).
4. When prompted, specify the ***hostname*** which contains the Control Broker you started in Step 1 above.
5. Select the SME schema.
6. After you verify that the Control Broker is connected (the message in the Control tab of the console will indicate command *succeeded* and status as *up*), highlight the IQ Manager, ***hostname***_igmgr, then right-click, and select **Start**.
7. Highlight each of the e*Ways, right-click the mouse, and select **Start**.
8. To view the output, copy the output file (specified in the Outbound_eWay configuration file). Save to a convenient location, open.

Note: *While the schema is running, opening the destination file causes errors.*

6.3.2 SMEdynamic Sample Schema

The previous sections provided the basics for implementing SME. This section describes how to use SME within the ***SMEdynamic*** schema using dynamic configuration.

To implement the ***SMEdynamic*** schema, follow all the same steps in this chapter, substituting the name of the schema when appropriate (***SMEdynamic*** should replace ***SME*)**. Refer to **"Dynamic Configuration in the Collaboration Rules Editor" on page 71** for a description of how the ***SMEdynamic*** schema's components are created using dynamic configuration.

Note: *It is assumed that SME has been installed properly, and that all of the necessary files and scripts are located in the default location.*

The ***SMEdynamic*** uses the **smemessage.xsc** Event Type Definition (ETD), created using the Custom ETD Wizard. For more information on creating Event Types and ETDs see **"Creating Event Types" on page 51**.

The structure of the ETD used in this sample is shown as it appears in the ETD editor in **"Understanding the Structure of the SME ETD" on page 44**.

- The **inputmsg.xsc** is an input message.

- The **smemessage.xsc** assigns the functionality to encrypt, decrypt, sign, and verify the input message.
- The **outevent.xsc** outputs a message that is published by the outbound e*Way.

Note: *It is assumed that SME has been installed properly, and that all of the necessary files and scripts are located in the default location.*

Like the ***SME*** sample, the implementation of the ***SMEdynamic*** schema consists of two file-based e*Ways, one Multi-Mode e*Way, three Event Types, three Collaboration Rules, two Intelligent Queues and three Collaborations, as follows:

- **inbound** - This e*Way receives input from an external source, apply pass through Collaboration Rules, and publish the information to an Intelligent Queue.
- **ew_java** - This Multi-Mode e*Way applies extended Java Collaboration Rules to an inbound Event to perform the desired business logic, in this case encryption and decryption.
- **outbound -** This e*Way receives information from the Multi-Mode e*Way and publish to the external system.
- **smemessage** - This Event Type contains the methods to be used to perform the necessary transformation.
- **input** - This Event Type describes an Event that is input to the extended Java Collaboration Service.
- **outevent** - This Event Type describes an Event that contains the transformed data.
- **passin** - This Collaboration Rule is associated with the *inbound* e*Way, and is used for receiving the input Event.
- **cor_java** - The Collaboration Rule is associated with the *ew_java* Multi-Mode e*Way, and is used to perform the transformation process.
- **passout** - This Collaboration Rule is associated with the *outbound* e*Way, and is used to send the Event to an external file.
- **iq1** - This Intelligent Queue is a STC_Standard IQ, and forwards data to the *ew_java* Multi-Mode e*Way.
- **iq2** - This Intelligent Queue is a STC_Standard IQ, and forwards data to the *outbound* e*Way.

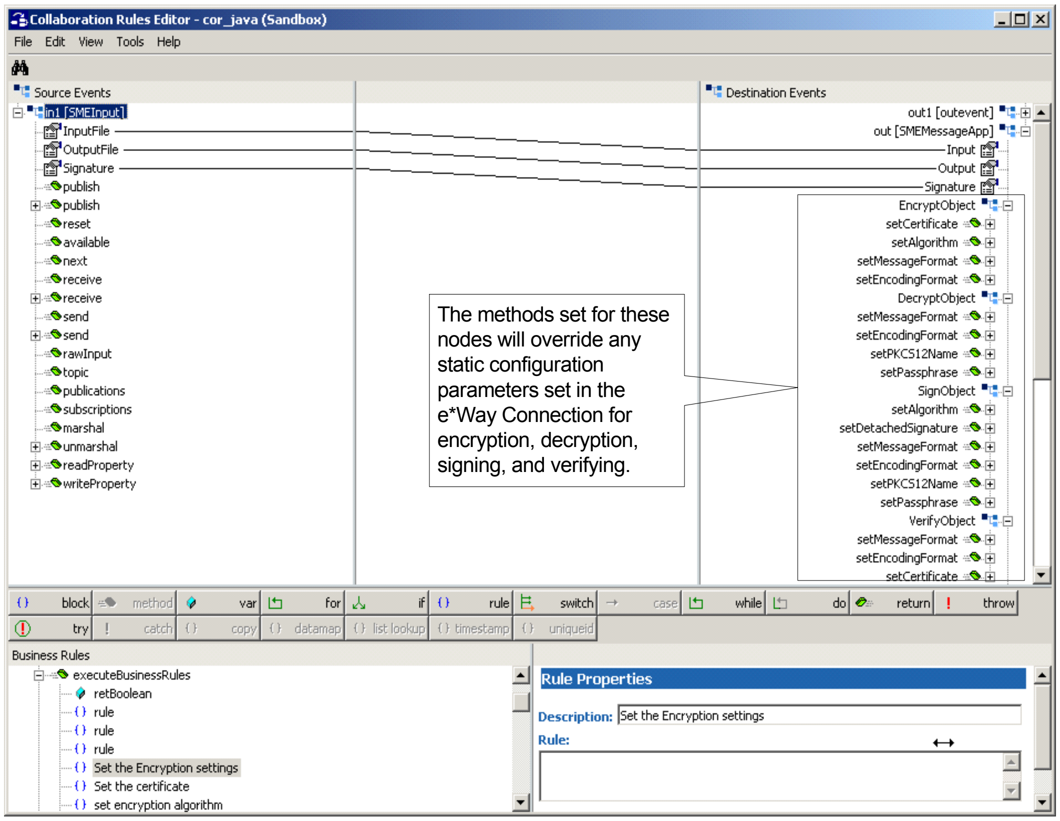
Dynamic Configuration in the Collaboration Rules Editor

The ***SMEdynamic*** schema configuration parameters were created using the Collaboration Rules Editor. As discussed in **"SMEMessageApp Root Node" on page 45**, the following nodes support dynamic configuration:

- EncryptObject
- DecryptObject
- SignObject
- VerifyObject

When creating business rules in the Collaboration Rules Editor, be aware that the methods set for these nodes will override any static configuration parameters set in the e*Way Connection for encryption, decryption, signing, and verifying. Figure 22 shows the nodes that support dynamic configuration in the Collaboration Rules Editor.

Figure 22 Collaboration Rules Editor— Dynamic Configuration Nodes



The methods set for these nodes will override any static configuration parameters set in the e*Way Connection for encryption, decryption, signing, and verifying.

Description: Set the Encryption settings

Executing the SMEdynamic Sample Schema

Once you have completed all the steps to manually create the ***SMEdynamic*** sample schema, you will then execute the schema.

To execute the SMEdynamic schema

1 After configuring all of the e*Way components associated with the sample, start the schema from the command line prompt, and enter the following:

```
stccb -ln localhost_cb -rh localhost -rs <schema name>-un
    Administrator -up <password>
```

Substitute ***hostname***, ***username***, ***schema name***, and ***user password*** as appropriate.

2 Exit from the command line prompt, and start the e*Gate Monitor GUI.

3 Start the component e*Ways (ew_java, inbound and outbound).

4 When prompted, specify the ***hostname*** which contains the Control Broker you started in Step 1 above.

5. Select the SME schema.
6. After you verify that the Control Broker is connected (the message in the Control tab of the console will indicate command *succeeded* and status as *up*), highlight the IQ Manager, ***hostname***_igmgr, then right-click, and select **Start**.
7. Highlight each of the e*Ways, right-click the mouse, and select **Start**.
8. To view the output, copy the output file (specified in the Outbound_eWay configuration file). Save to a convenient location, open.

Note: *While the schema is running, opening the destination file causes errors.*

Chapter 7

Secure Messaging Extension Methods

This chapter explains the Java class and methods contained in Secure Messaging Extension.

7.1 SME Methods: Overview

Using Java Methods

SME contains Java methods that are used to extend functionality. Java methods make it easy to set information in the SME Event Type Definition (ETD), as well as to get information from it. The nature of this data transfer corresponds to either setting the connection parameters in the e*Gate Enterprise Manager's e*Way Editor (known as static configuration), or setting the configuration parameters with methods in the SME Collaboration Editor (known as dynamic configuration).

The Enterprise Manager's Collaboration Rules Editor allows you to call Java methods by dragging and dropping an ETD node into the **Rules** dialog box.

After you drag and drop, the actual conversion takes place in the **smemessage.xsc** file. To view the **smemessage.xsc** file, use the Enterprise Manager's ETD Editor and Collaboration Rules Editor. See **Chapter 5** for more information.

Note: *Node names can be different from Java method names.*

7.2 SMEMessage Methods Used with Static Configuration

Some of the methods in this section are used only when the configuration parameters are set in the e*Way Editor (static configuration). For more information, see **"Configuring e*Way Connections" on page 32**. The following methods of the **SMEMessage class** are found in the **com.stc.sme.eways** package:

Methods of the SMEMessage Class

These methods are described in detail on the following pages:

base64Decode on page 75	**decrypt** on page 77

base64Decode

Description

base64Decode is used obtain the base64 decoded result of the input message.

Syntax

```
public byte[] base64Decode(byte[] input)
```

Parameters

Parameter name	Type	Description
input	byte array	byte array of the message.

Return Values

byte array
Returns the signed message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the signing process.

base64Encode

Description

base64Encode is used obtain the base64 encoded result of the input message.

Syntax

```
public byte[] base64Encode(byte[] input)
```

Parameters

Parameter name	Type	Description
input	byte array	byte array of the message.

Return Values

byte array
Returns the signed message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the signing process.

encrypt

Description

encrypt is used to encrypt the message for the specified recipients. The input field of SMEMessageAPP ETD needs to be set to the filename which contains the message.

Syntax

```
public byte[] encrypt()
```

Parameters

None.

Return Values

byte array
Returns an encrypted message, if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the encryption process.

encrypt

Description

encrypt is used to encrypt the message for the specified recipients.

Syntax

```
public byte[] encrypt(byte[] input)
```

Parameters

Parameter name	Type	Description
input	byte array	The original message for encryption.

Return Values

byte array
Returns an encrypted message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the encryption process.

decrypt

Description

decrypt is used to decrypt the encrypted input message. The input field of SMEMessageAPP ETD needs to be set to the filename which contains the message.

Syntax

```
public byte[] decrypt()
```

Parameters

None.

Return Values

byte array
Returns the original message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the decryption process.

decrypt

Description

decrypt is used to decrypt the encrypted input message.

Syntax

```
public byte[] decrypt(byte[] input)
```

Parameters

Parameter name	Type	Description
input	byte array	The encrypted message.

Return Values

byte array
Returns the original message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the decryption process.

getMD5Hash

Description

getMD5Hash is used obtain the hash code of the input message using MD5.

Syntax

```
public byte[] getMD5Hash(byte[] input)
```

Parameters

Parameter name	Type	Description
input	byte array	byte array of the message.

Return Values

byte array
Returns the signed message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the signing process.

getSHA1Hash

Description

getSHA1Hash is used obtain the hash code of the input message using SH1.

Syntax

```
public byte[] getSHA1Hash(byte[] input)
```

Parameters

Parameter name	Type	Description
input	byte array	byte array of the message.

Return Values

byte array
Returns the signed message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the signing process.

sign

Description

sign is used to sign a message with a detached or inline signature. The input field of SMEMessageAPP ETD needs to be set to the filename which contains the message.

Syntax

```
public byte[] sign()
```

Parameters

None.

Return Values

byte array
Returns the signed message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the signing process.

sign

Description

sign is used to sign a message with a detached or inline signature.

Syntax

```
public byte[] sign(byte[] input clearSign)
```

Parameters

Parameter name	Type	Description
input	byte array	The message used for signing.
clearSign	boolean	If true, signature is a detached signature. If false, signature is an inline signature.

Return Values

byte array
Returns the signed message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the signing process.

verify

Description

verify is used to verify the signed message and return the original message, provided the signed message is verified. The content of the specified input field is used as the byte array message that needs to be authenticated, if the message is in SMIME2 format or in PKCS7 format with inline signature.

If the message to be verified is in PKCS7 format, and has a detached signature, the input field is used to indicate the file that contains the original message, and the Signature field is used to indicate that the file contains the detached signature.

Syntax

```
public byte[] verify()
```

Parameters

None.

Return Values

byte array
Returns the original message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the verification process.

verify

Description

verify is used for verification of the attached PKCS7 signature, in which case, the signature portion and the original content are kept together.

Syntax

```
public byte[] verify(byte[] input1)
```

Parameters

Parameter name	Type	Description
input1	byte[]	Byte array for the attached message.

Return Values

byte array
Returns the original message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the verification process.

verify

Description

verify is used for verification of the detached PKCS7 signature, in which case, the signature portion and the original content is separated.

Syntax

```
public byte[] verify(byte[] input1 byte[] input2)
```

Parameters

Parameter name	Type	Description
input1	byte[]	Byte array for the original message.
input2	byte[]	Byte array of the detached signature.

Return Values

byte array
Returns the original message if successful.

Throws

com.stc.sme.exception.SMEException, indicating an error occurred during the verification process.

7.3 SMEMessage Methods Used with Dynamic Configuration

Some of these methods support setting the connection parameters by using the Collaboration Editor (dynamic configuration). The following methods of the **SMEMessage class** are found in the **com.stc.sme.eways** package:

Methods of the SMEMessage Class

These methods are described in detail on the following pages:

setCertificate

Description

Sets the certificate that is used to encrypt the message.

Syntax

```
public void setCertificate(java.lang.java.lang.String certFileName,
java.lang.java.lang.String certFormat)
```

Parameters

Parameter name	Type	Description
certFileName	java.lang.String	File that contains the certificate.
certFormat	java.lang.String	Certificate format. Valid values are DER, PEM, and PK7.

Return Values

None.

Throws

None.

setAlgorithm

Description

Specifies the symmetric algorithm used for encryption.

Syntax

```
public void setAlgorithm(java.lang.java.lang.String algorithm)
```

Parameters

Parameter name	Type	Description
algorithm	java.lang.String	Valid values are DES_ED3_CBC, RC2_CBC_40, RC2_CBC_64, and RC2_CBC_128.

Return Values

None.

Throws

None.

setMessageFormat

Description

Specifies the message format for encrypted output.

Syntax

```
public void setMessageFormat(java.lang.String messageFormat)
```

Parameters

Parameter name	Type	Description
messageFormat	java.lang.String	Valid values are SMIME2 or PKCS7.

Return Values

None.

Throws

None.

setEncodingFormat

Description

Specifies the encoding format (used only for SMIME2 message format). If PKCS7 message format is used, the ouput is in binary format.

Syntax

```
public void setEncodingFormat(java.lang.String encodingFormat)
```

Parameters

Parameter name	Type	Description
encodingFormat	java.lang.String	Valid values are BASE64 or BINARY.

Return Values

None.

Throws

None.

setMessageFormat

Description

Specifies the message format for encrypted output.

Syntax

```
public void setMessageFormat(java.lang.String messageFormat)
```

Parameters

Parameter name	Type	Description
messageFormat	java.lang.String	Valid values are SMIME2 or PKCS7.

Return Values

None.

Throws

None.

setEncodingFormat

Description

Specifies the encoding format (used only for SMIME2 message format). If PKCS7 message format is used, the ouput is in binary format.

Syntax

```
public void setEncodingFormat(java.lang.String encodingFormat)
```

Parameters

Parameter name	Type	Description
encodingFormat	java.lang.String	Valid values are BASE64 or BINARY.

Return Values

None.

Throws

None.

setPKCS12Name

Description

Sets the file name which contains the user's private key.

Syntax

```
public void setPKCS12Name(java.lang.String pkcs12FileName)
```

Parameters

Parameter name	Type	Description
pkcs12FileName	java.lang.String	Name of the file which contains the private key.

Return Values

None.

Throws

None.

setPassphrase

Description

Sets the password for the pkcs12 file.

Syntax

```
public void setPassphrase(char[] passphrase)
```

Parameters

Parameter name	Type	Description
passphrase	char	Unencrypted password for the pkcs12 file.

Return Values

None.

Throws

None.

Note: *SeeBeyond does not recommend using this method. For password protection, set the PKCS12 file and PassPhrase through the e*Way Connection. For more information, see* **"PKCS12" on page 37***, and* **"PassPhrase" on page 39***.*

setAlgorithm

Description

Specifies the signing algorithm used for encryption.

Syntax

```
public void setAlgorithm(java.lang.String algorithm)
```

Parameters

Parameter name	Type	Description
algorithm	java.lang.String	Valid values are RSA_SHA1 and RSA_MD5

Return Values

None.

Throws

None.

setDetachedSignature

Description

Specifies whether the signature should be detached or not.

Syntax

```
public void setDetachedSignature(boolean isDetached)
```

Parameters

Parameter name	Type	Description
isDetached	boolean	Set to True if the signature should be detached; false if otherwise.

Return Values

None.

Throws

None.

setMessageFormat

Description

Specifies the message format used for encrypted output.

Syntax

```
public void setMessageFormat(java.lang.String messageFormat)
```

Parameters

Parameter name	Type	Description
messageFormat	java.lang.String	Valid values are SMIME2 and PKCS7.

Return Values

None.

Throws

None.

setEncodingFormat

Description

Specifies the encoding format (used only for SMIME2 message format). If PKCS7 message format is used, the ouput is in binary format.

Syntax

```
public void setEncodingFormat(java.lang.String encodingFormat)
```

Parameters

Parameter name	Type	Description
encodingFormat	java.lang.String	Valid values are BASE64 and BINARY.

Return Values

None.

Throws

None.

setPKCS12Name

Description

Sets the file name which contains the user's private key.

Syntax

```
public void setPKCS12Name(java.lang.String pkcs12FileName)
```

Parameters

Parameter name	Type	Description
pkcs12FileName	java.lang.String	Name of the file which contains the private key.

Return Values

None.

Throws

None.

setPassphrase

Description

Sets the password for the pkcs12 file.

Syntax

```
public void setPassphrase(char[] passphrase)
```

Parameters

Parameter name	Type	Description
passphrase	char	Unencrypted password for the pkcs12 file.

Return Values

None.

Throws

None.

Note: *SeeBeyond does not recommend using this method. For password protection, set the PKCS12 file and PassPhrase through the e*Way Connection. For more information, see* **"PKCS12" on page 37***, and* **"PassPhrase" on page 39***.*

setCertificate

Description

Sets the certificate that is used to encrypt the message.

Syntax

```
public void setCertificate(java.lang.String certFileName,
java.lang.String certFormat)
```

Parameters

Parameter name	Type	Description
certFileName	java.lang.String	File that contains the certificate.
certFormat	java.lang.String	Certificate format. Valid values are DER, PEM, and PK7.

Return Values

None.

Throws

None.

setMessageFormat

Description

Specifies the message format for encrypted output.

Syntax

```
public void setMessageFormat(java.lang.String messageFormat)
```

Parameters

Parameter name	Type	Description
messageFormat	java.lang.String	Valid values are SMIME2 or PKCS7.

Return Values

None.

Throws

None.

setEncodingFormat

Description

Specifies the encoding format (used only for SMIME2 message format). If PKCS7 message format is used, the ouput is in binary format.

Syntax

```
public void setEncodingFormat(java.lang.String encodingFormat)
```

Parameters

Parameter name	Type	Description
encodingFormat	java.lang.String	Valid values are BASE64 or BINARY.

Return Values

None.

Throws

None.

setChecking

Description

Specifies the checking method for the certificate.

Syntax

```
public void setChecking(java.lang.String checkingMethod)
```

Parameters

Parameter name	Type	Description
checkingMethod	java.lang.String	Valid values are DIRECT, CA, and CA_CRL.

Return Values

None.

Throws

SMEException

addTrustedCAs

Description

Adds the trusted CA's certificate. This method is used to insert the trusted CA's certificate if CA is selected as the checking method. Do not use this method if DIRECT is selected.

Syntax

```
public void addTrustedCAs(java.lang.String CACertFile,
java.lang.String CACertFormat)
```

Parameters

Parameter name	Type	Description
CACertFile	java.lang.String	File that contains the CA's certificate.
CACertFormat	java.lang.String	Certificate format. Valid values are DER, PEM, and PK7.

Return Values

None.

Throws

SMEException

addCRL

Description

Locates the CRLs. Use only if CACRL is selected for checking.

Syntax

```
public void addCRL(java.lang.String CRLFileName, java.lang.String
CACRLFileName, java.lang.String CACRLFormat, java.lang.String
CAFileName, java.lang.String CAFormat)
```

Parameters

Parameter name	Type	Description
CRLFileName	java.lang.String	File that contains the CRL. File must be DER encoded.
CACRLFileName	java.lang.String	CA's certificate. The corresponding private key is used to sign the CRL.
CACRLFormat	java.lang.String	Valid values are DER, PK7, and PEM.
CAFileName	java.lang.String	CA's certificate used to verify the user's certificate if it is the same as the certificate used to verify the CRL. If it is not the same, it is null.
CAFormat	java.lang.String	Valid values are DER, PK7, and PEM.

Return Values

None.

Throws

SMEException

addLDAPService

Description

Locates the CRLs. Use only if CACRL is selected for checking.

Syntax

```
public void addLDAPService(java.lang.String hostname, int portNumber,
java.lang.String username, java.lang.String password,
java.lang.String CACRLCertFile, java.lang.String CACertFile,
java.lang.String hierarchy, java.lang.String localCRLFile)
```

Parameters

Parameter name	Type	Description
hostname	java.lang.String	Hostname for the LDAP server.
portNumber	int	Port number for LDAP server.

Parameter name	Type	Description
username	java.lang.String	Login name for the LDAP server. Set to null if not required.
password	java.lang.String	Password for the login name of the LDAP server. Set to null if not required.
CACRLCertFile	java.lang.String	File name for the CA's certificate used to verify the CRL. File content should be DER encoded.
CACertFile	java.lang.String	File name for the CA's certificate used to verify the user's certificate. File content should be DER encoded. If it is the same as the verified CACRL CertFile, then set this parameter to null.
hierarchy	java.lang.String	LDAP hierarchy used to retrieve the CRL.
localCRLFile	java.lang.String	Local file name used to store retrieved CRL. If set to null, then retrieved CRL is not stored.

Return Values

None.

Throws

SMEException

setAllowSelfSignedCertificate

Description

Specifies whether the self signed certificate is allowed for use.

Syntax

```
public void setAllowSelfSignedCertificate(boolean allowable)
```

Parameters

Parameter name	Type	Description
allowable	boolean	Set to true if self signed certificate is allowed; false if otherwise.

Return Values

None.

Throws

None.

Index