

Nokia PC Connectivity SDK 2.0

Component Library Reference

For Nokia Phones

Table of Contents

1. INTRODUCTION.....	5
1.1 Legal Information	5
1.2 Prerequisites	8
1.3 Overview	8
1.4 Document Conventions	9
1.5 Examples	10
1.6 References	11
2. NOKIA PC CONNECTIVITY SDK 2.0 INSTALLATION	12
2.1 Installation	12
2.1.1 General	12
2.1.2 Launching the Installation	12
2.2 Installation Steps	12
2.2.1 License Agreement	12
2.2.2 Connection Method Selection	13
2.2.3 Start Copying Files	15
2.2.4 Setup Complete	15
2.3 Installation Components and Locations	15
2.3.1 DCOM95 Installation	16
3. LANGUAGE SUPPORT	17
3.1 Basic Concepts	17
3.2 Using Libraries with Visual Basic	17
3.2.1 Referencing Object Libraries	17
3.2.2 Referencing Primary Interfaces	18
3.2.3 Referencing Secondary Interfaces	19
3.2.4 Referencing Outgoing Interfaces	19
3.2.5 Starting Sequence	19
3.2.6 Event Handlers	21
3.2.7 Dispatch Interfaces	21
3.2.8 Error Handling	21

3.3	Using Libraries with Visual Basic for Applications	21
3.4	Using Libraries with Visual C++	22
3.4.1	<i>Referencing Incoming Interfaces</i>	23
3.4.2	<i>Referencing Outgoing Interfaces</i>	27
3.5	Using Libraries with Delphi	28
3.5.1	<i>Referencing Primary Interfaces</i>	29
3.5.2	<i>Referencing Secondary Interfaces</i>	30
3.5.3	<i>Referencing Outgoing Interfaces</i>	30
4.	GENERAL SETTINGS LIBRARY (STTINGS3A_SLIB)	32
4.1	Overview	32
4.1.1	<i>Object Model</i>	32
4.2	PhoneInfo_Suite3 Component	33
4.2.1	<i>IPhoneInfo Interface</i>	33
4.2.2	<i>IPhoneInfoNotify Interface</i>	44
4.2.3	<i>IPhoneCapability Interface</i>	47
4.2.4	<i>IPhoneStatus Interface</i>	49
4.3	PhoneControl_Suite3 Component	59
4.3.1	<i>IPhoneControl2 Interface</i>	59
4.3.2	<i>IPhoneUI Interface</i>	64
4.3.3	<i>IClock Interface</i>	70
4.3.4	<i>IMediaPool Interface</i>	75
4.4	TONE Component	78
4.4.1	<i>ITone Interface</i>	78
4.5	Security_Suite3 Component	81
4.5.1	<i>ISecurity Interface</i>	81
4.6	Network_Suite3 Component	87
4.6.1	<i>INetwork Interface</i>	87
4.6.2	<i>INetworkNotify Interface</i>	93
4.7	Enumerated DATA Types	98
4.7.1	<i>CapabilityOpt</i>	98
4.7.2	<i>DevNotifyOpt</i>	101
4.7.3	<i>EnchBoolean</i>	102
4.7.4	<i>MemTypeOpt</i>	102
4.7.5	<i>NETWORK_REG_STATUS</i>	103
4.7.6	<i>NETWORK_SELECTION_MODE</i>	103
4.7.7	<i>NmpAdapterError</i>	104
4.7.8	<i>MediaType</i>	108
5.	SHORT MESSAGE LIBRARY (SMS3ASUITELIB)	109
5.1	Overview	109
5.2	Object Model	109
5.3	SMS_SuiteAdapter Component	110
5.3.1	<i>ISMSSend Interface</i>	110
5.3.2	<i>ISMSReceiveNotify</i>	116
5.3.3	<i>ISMSMemory Interface</i>	120
5.3.4	<i>ISMSSettings Interface</i>	138
5.3.5	<i>IGraphicalMS Interface</i>	149
5.4	ShortMessage Component	154
5.4.1	<i>IShortMessage Interface</i>	154
5.5	GMSPICTURE Component	187

5.5.1	<i>IGMSPicture Interface</i>	188
5.6	Enumerated DATA Types	195
5.6.1	<i>REPORT_QUALIFIER</i>	195
5.6.2	<i>SMS_MEMORY_LOCATION</i>	196
5.6.3	<i>SMS_MESSAGE_TYPE</i>	196
5.6.4	<i>STORAGE_STATUS</i>	197
5.6.5	<i>USER_DATA_FORMAT</i>	197
5.6.6	<i>VALIDITY_PERIOD_FORMAT</i>	198
6.	PHONEBOOK MEMORY LIBRARY (PHONEBOOKADAPTERDS3).....	199
6.1	Overview	199
6.1.1	<i>Object Model</i>	199
6.2	Phonebook Component.....	200
6.2.1	<i>IPhonebook Interface</i>	200
6.2.2	<i>IPhonebook2 Interface</i>	215
6.2.3	<i>IPhonebook3 Interface</i>	218
6.2.4	<i>IPhonebookNotify Interface</i>	236
6.3	Contact Component.....	240
6.3.1	<i>IContact Interface</i>	241
6.3.2	<i>IContact2 Interface</i>	251
6.4	CallerGroup Component.....	265
6.4.1	<i>ICallerGroup Interface</i>	266
6.4.2	<i>ICallerGroup2 Interface</i>	275
6.5	CallerGroupIcon Component.....	284
6.5.1	<i>ICallerGroupIcon Interface</i>	284
6.6	Enumerated DATA Types	293
7.	CALENDAR LIBRARY (CALADAPTERLIB)	302
7.1	Overview	302
7.2	Object Model.....	302
7.3	CALENDARITEMSERVEx Component	303
7.3.1	<i>ICalendarItemServEx Interface</i>	303
7.3.2	<i>CalendarItemServ2 Interface</i>	313
7.3.3	<i>IPhoneCalendarNotify2 Interface</i>	337
7.4	CalendarItemAttrEx Component	339
7.4.1	<i>Type Definitions</i>	339
7.4.2	<i>ICalendarItemAttrEx Interface</i>	340
7.4.3	<i>IBirthdayEx Interface</i>	343
7.4.4	<i>ICallToEx Interface</i>	344
7.4.5	<i>IMeetingEx Interface</i>	345
7.4.6	<i>IMessageEx Interface</i>	346
7.4.7	<i>IBirthday2 Interface</i>	347
7.4.8	<i>ICallTo2 Interface</i>	348
7.4.9	<i>IMeeting2 Interface</i>	349
7.4.10	<i>IMessage2 Interface</i>	350
7.4.11	<i>Event Component Properties</i>	350
8.	WAP ADAPTER LIBRARY (NOKIACLWAP)	361
8.1	Overview	361
8.1.1	<i>Object Model</i>	361
8.2	WAPSettings Component	362

8.2.1	<i>IWAPSettings Interface</i>	362
8.2.2	<i>IWAPBookmark Interface</i>	374
8.3	SettingGroupItem Component	380
8.3.1	<i>ISettingGroup Interface</i>	380
8.4	Beareritem Component	386
8.4.1	<i>ISMS Interface</i>	386
8.4.2	<i>ICSD Interface</i>	389
8.5	Bookmarkitem Component	398
8.5.1	<i>BookmarkItem Interface</i>	398
8.6	Enumerated DATA Types	402
8.6.1	<i>WAPBearerType</i>	403
8.6.2	<i>WAPSessionSecurity</i>	404
8.6.3	<i>WAPSessionType</i>	405
8.6.4	<i>WAPAuthType</i>	406
8.6.5	<i>WAPDataCallType</i>	407
8.6.6	<i>WAPAnalogDatacallSpeed</i>	408
8.6.7	<i>WAPISDNDatacallSpeed</i>	409

1. INTRODUCTION

1.1 Legal Information

Nokia PC Connectivity SDK 2.0

IMPORTANT: READ CAREFULLY BEFORE INSTALLING, DOWNLOADING, OR USING THE SOFTWARE

NOKIA MOBILE PHONES END-USER SOFTWARE AGREEMENT

This Software Agreement ("Agreement") is between You (either an individual or an entity), the Developer, and Nokia Mobile Phones Ltd. ("Nokia"). The Agreement authorizes You to use the Software specified in Clause 1 below, which may be stored on a CD-ROM, sent to You by electronic mail, or downloaded from Nokia's Web pages or Servers or from other sources under the terms and conditions set forth below. This is an agreement on user rights and not an agreement for sale. Nokia continues to own the copy of the Software and the physical media it is provided on and any other copy that You are authorized to make pursuant to this Agreement.

Read this Agreement carefully before installing, downloading, or using the Software. By clicking on the "I Accept" button while installing, downloading, and/or using the Software, You agree to the terms and conditions of this Agreement. If You do not agree to all of the terms and conditions of this Agreement, promptly click the "Decline" or "I Do Not Accept" button, cancel the installation or downloading, or destroy or return the Software and accompanying documentation to Nokia. YOU AGREE THAT YOUR USE OF THE SOFTWARE ACKNOWLEDGES THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS.

1. SOFTWARE. As used in this Agreement, the term "Software" means, collectively: (i) the software product identified above (ii) all the contents of the disk(s), CD-ROM(s), electronic mail and its file attachments, or other media with which this Agreement is provided, including the object code form or the source code form of the software delivered via a CD-ROM, electronic mail, or Web page (iii) digital images, stock photographs, clip art, or other artistic works ("Stock Files") (iv) related explanatory written materials and any other possible documentation related thereto ("Documentation"); (v) fonts, and (vi) upgrades, modified versions, updates, additions, and copies of the Software (collectively "Updates"), if any, licensed to You by Nokia under this Agreement.

2. LICENSED RIGHTS AND USE. Nokia grants to You non-exclusive, non-transferable license to install the Software on the local hard disk(s) or other permanent storage media of one computer and use the Software on a single computer or terminal at a time for the term of this Agreement.

3. LIMITATIONS ON RIGHTS. You may not copy, distribute, or make other derivative works of the Software except as follows:

(a) You may make one copy of the Software on magnetic media as an archival backup copy, provided Your archival backup copy is not installed or used on any computer. Any other copies You make of the Software are in violation of this Agreement.

(b) You may not use, modify, translate, reproduce, or transfer the right to use the Software or copy the Software except as expressly provided in this Agreement.

(c) You may not resell, sublicense, rent, lease, or lend the Software.

(d) You may not reverse engineer, reverse compile, disassemble, or otherwise attempt to discover the source code of the Software (except to the extent that this restriction is expressly prohibited by law)

(e) You may not create other derivative works based on the Software than those using the interfaces listed in the documentation.

(f) Unless stated otherwise in the Documentation, You shall not display, modify, reproduce, or distribute any of the Stock Files included with the Software. In the event that the Documentation allows You to display the Stock Files, You shall not distribute the Stock Files on a stand-alone basis, i.e., in circumstances in which the Stock Files constitute the primary value of the product being distributed. You should review the "Readme" files associated with the Stock Files that You use to ascertain what rights You have with respect to such materials. Stock Files may not be used in the production of libelous, defamatory, fraudulent, infringing, lewd, obscene, or pornographic material or in any otherwise illegal manner. You may not register or claim any rights in the Stock Files or derivative works thereof.

(g) You agree that You shall only use the Software in a manner that complies with all applicable laws in the jurisdiction in which You use the Software, including, but not limited to, applicable restrictions concerning copyright and other intellectual property rights.

4. COPYRIGHT. The Software and all rights, without limitation including proprietary rights therein, are owned by Nokia and/or its licensors and affiliates and are protected by international treaty provisions and all other applicable national laws of the country in which it is being used. The structure, organization, and code of the Software are the valuable trade secrets and confidential information of Nokia and/or its licensors and affiliates. You must not copy the Software, except as set forth in clause 3 (Limitations On End-User Rights). Any copies which You are permitted to make pursuant to this Agreement must contain the same copyright and other proprietary notices that appear on the Software.

5. MULTIPLE ENVIRONMENT SOFTWARE / MULTIPLE LANGUAGE SOFTWARE / DUAL MEDIA SOFTWARE / MULTIPLE COPIES / UPDATES. If the Software supports multiple platforms or languages, if You receive the Software on multiple media, or if You otherwise receive multiple copies of the Software, the number of computers on which all versions of the Software are installed shall be one computer. You may not rent, lease, sublicense, lend, or transfer versions or copies of the Software You do not use. If the Software is an Update to a previous version of the Software, You must possess valid end-user rights to such a previous version in order to use the Update, and You may use the previous version for ninety (90) days after You receive the Update in order to assist You in the transition to the Update. After such time You no longer have a right to use the previous version, except for the sole purpose of enabling You to install the Update.

6. COMMENCEMENT & TERMINATION. This Agreement is effective from the first date You install the Software and shall be valid for a term of six (6) months. You may also terminate this Agreement at any time by permanently deleting, destroying, and returning, at Your own costs, the Software, all backup copies, and all related materials provided by Nokia. Your end-user rights automatically and immediately terminate without notice from Nokia if You fail to comply with any provision of this Agreement. In such an event, You must immediately delete, destroy, or return at Your own cost, the Software, all backup copies, and all related material to Nokia.

7. YOU ACKNOWLEDGE THAT THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW NEITHER NOKIA, ITS LICENSORS OR AFFILIATES, NOR THE COPYRIGHT HOLDERS MAKE ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, RELATED TO THE SOFTWARE OR THE DERIVATIVE WORKS YOU MAY CREATE WITH THE SOFTWARE, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR THAT THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS, OR OTHER RIGHTS. THERE IS NO WARRANTY BY NOKIA OR BY ANY OTHER PARTY THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE OR THE POSSIBLE DERIVATIVE WORKS YOU MAY CREATE WITH THE SOFTWARE WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE. YOU

ASSUME ALL RESPONSIBILITY AND RISK FOR THE SELECTION OF THE SOFTWARE AND DISTRIBUTION OF ANY DERIVATIVE WORKS CREATED WITH THE SOFTWARE AND TO ACHIEVE YOUR INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM IT.

8. NO OTHER OBLIGATIONS. This Agreement creates no obligations on the part of Nokia other than as specifically set forth herein.

9. LIMITATION OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL NOKIA, ITS EMPLOYEES OR LICENSORS OR AFFILIATES BE LIABLE FOR ANY LOST PROFITS, REVENUE, SALES, DATA, OR COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, PROPERTY DAMAGE, PERSONAL INJURY, INTERRUPTION OF BUSINESS, LOSS OF BUSINESS INFORMATION, OR FOR ANY SPECIAL, DIRECT, INDIRECT, INCIDENTAL, ECONOMIC, COVER, PUNITIVE, SPECIAL, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND WHETHER ARISING UNDER CONTRACT, TORT, NEGLIGENCE, OR OTHER THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF NOKIA OR ITS LICENSORS OR AFFILIATES ARE ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME COUNTRIES/STATES/JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF LIABILITY, BUT MAY ALLOW LIABILITY TO BE LIMITED, IN SUCH CASES, NOKIA, ITS EMPLOYEES OR LICENSORS OR AFFILIATES' LIABILITY SHALL BE LIMITED TO U.S. \$50.

Nothing contained in this Agreement limits Nokia's liability to You in the event of death or personal injury resulting from Nokia's negligence. Nokia is acting on behalf of its employees and licensors or affiliates for the purpose of disclaiming, excluding, and/or restricting obligations, warranties, and liability as provided in this clause 9, but in no other respects and for no other purpose.

10. TECHNICAL SUPPORT. Nokia has no obligation to furnish You with technical support unless separately agreed in writing between You and Nokia.

11. EXPORT CONTROL. The Software, including technical data, includes cryptographic software subject to export controls under the U.S. Export Administration Regulations ("EAR") and may be subject to import or export controls in other countries. The EAR prohibits the use of the Software and technical data by a Government End User, as defined hereafter, without a license from the U.S. government. A Government End User is defined in Part 772 of the EAR as "any foreign central, regional, or local government department, agency, or other entity performing governmental functions; including governmental research institutions, governmental corporations, or their separate business units (as defined in part 772 of the EAR) which are engaged in the manufacture or distribution of items or services controlled on the Wassenaar Munitions List, and international governmental organizations. This term does not include: utilities (telecommunications companies and Internet service providers; banks and financial institutions; transportation; broadcast or entertainment; educational organizations; civil health and medical organizations; retail or wholesale firms; and manufacturing or industrial entities not engaged in the manufacture or distribution of items or services controlled on the Wassenaar Munitions List.)" You agree to strictly comply with all applicable import and export regulations and acknowledge that You have the responsibility to obtain licenses to export, re-export, transfer, or import the Software. You further represent that You are not a Government End User as defined above, and You will not transfer the Software to any Government End User without a license.

12. NOTICES. All notices and return of the Software and Documentation should be delivered to:

NOKIA MOBILE PHONES LTD.
P.O. Box 100
FIN-00045 NOKIA GROUP
FINLAND

13. APPLICABLE LAW & GENERAL PROVISIONS

This Agreement is governed by the laws of Finland. All disputes arising from or relating to this Agreement shall be settled by a single arbitrator appointed by the Central Chamber of Commerce of Finland. The arbitration procedure shall take place in Helsinki, Finland in the English language. If any part of this Agreement is found void and unenforceable, it will not affect the validity of the balance of the Agreement, which shall remain valid and enforceable according to its terms. This Agreement may only be modified in writing by an authorized officer of Nokia.

This is the entire agreement between Nokia and You relating to the Software, and it supersedes any prior representations, discussions, undertakings, end-user agreements, communications, or advertising relating to the Software.

PLEASE SUBMIT ANY ACCOMPANYING REGISTRATION FORMS TO RECEIVE REGISTRATION BENEFITS WHERE APPLICABLE

1.2 Prerequisites

To be able to use Nokia PC Connectivity SDK you need:

- Nokia 6110 or 5110 compatible phone operating on a GSM 900, GSM 1800 or GSM 1900 network
- Nokia 6210, 6250, 7110, 7190, 8210, 8810, 8850 or 8890 GSM phone
- Nokia 7160 TDMA phone
- DLR-3P connection cable for 6210, 6250, 7110, 7160 or 7190 phone (available separately)
- DAU-9P connection cable for 6110 or 5110 and compatible phones (supplied with Nokia Data Suite)
- IrDA connection (for 6210, 6250, 7110, 7160, 7190, 8810, 8210, 8850 and 8890)

1.3 Overview

The Nokia PC Connectivity SDK is a sophisticated and easy-to-use programming interface for Nokia GSM and TDMA phones. The library consists of several separate libraries, each performing a specialized set of tasks related to GSM or TDMA phone functionality. These libraries are: the General Settings Library, the SMS Library, the Phonebook Memory Library, the WAP Library and the Calendar Library.

In the following, the libraries contained in Nokia PC Connectivity SDK 2.0 are listed and briefly described.

Library	Description
STTINGS3A_Slib	General Settings Library: adjusting settings on the GSM phone (Stngs3AS.dll).
SMS3AsuiteLib	Short Message Library: sending and receiving of messages and SMS memory management (Sms3aS.dll).
PhonebookAdapterDS3	Phonebook memory, speed dial key, and caller group

	management (SCM3aS.dll).
CALADAPTERLib	Calendar management (Cal3aS.dll).
NOKIACLWAP	WAP Settings Library, handling WAP settings and groups (NclWAP.dll).

All the libraries are implemented as COM (Component Object Model) Libraries. COM is the name of the specification for Microsoft's basic object technology that defines the standard for integration between software components.

A client application exploits these libraries through object libraries, which are also called type libraries in some contexts. An object library can be considered a binary description of the component library. The object libraries corresponding to the libraries of Nokia PC Connectivity SDK are mentioned in parenthesis in the library table above. Many environments support object libraries: Visual Basic, Visual C++, Delphi, Visual J++, and many more. The more detailed environment specific means to reference the object libraries are described in Chapter 3.

Each library in Nokia PC Connectivity SDK contains one or more functional entities called components. These reusable software components present their functionality through a defined set of interfaces. An interface contains a collection of related properties, methods, and functions that are grouped together under one name. A client application creates an instance of a component and a component object, sets a reference to the desired interface, and accesses interface methods through this reference.

Interfaces are divided into two categories according to which party has invoked the interface methods. The two interface categories in this sense are incoming and outgoing interfaces.

The methods of incoming interfaces are implemented on the component object and receive calls from external clients. The object performs the desired service and then returns the results to the client. Most interfaces in this component library are incoming interfaces that are called by your client application.

Meanwhile, methods (events) of outgoing interfaces are implemented on the client's sink and receive calls from the object. The object defines an interface it would like to use, and the client implements it. Thus, outgoing interfaces enable the object to talk back to its client. Outgoing interfaces are usually used to notify the client when something important happens in its sphere or to inform the client when some asynchronous operation has been completed. Outgoing interfaces are also often called connection points, event interfaces, notify interfaces, or source interfaces.

Note: A client thread that has no message pump or a thread that may be blocked in some Win32 wait function is not allowed to declare itself an event sink. Otherwise, the event calls queued by the component object become blocked and the component server is likely to deadlock.

1.4 Document Conventions

The following special fonts and font styles are used throughout this document.

Font & Style	Meaning
Times Bold	Used to indicate both reserved words and the keywords of the component library i.e., property, method, function, and event names.
<i>Times Italics</i>	Used to indicate variable names.
Courier New	Used to indicate both source code and phone key presses.

Below, there is a sample description of a method with short explanations for documentation format. All the properties, methods, functions, and events of Nokia PC Connectivity SDK are described by using the same documentation format.

4.4.10 Terminate Method

This method disconnects from the component server.

Member of STTNGS3A_SLib.IClock, STTNGS3A_SLib.IPhoneStatus, STTNGS3A_SLib.ISecurityPhone, STTNGS3A_SLib.Network_Suite3, STTNGS3A_SLib.PhoneInfo_Suite3 and STTNGS3A_SLib.Security_Suite3.

Syntax

Call *object*.**Terminate**

The **Terminate** method syntax has these parts:

Part	Description
<i>object</i>	An object expression that evaluates to a IClock, INetAOC, IPhoneStatus, IProfile, ISecurityNetwork, ISecurityPhone, Network_Suite3, PhoneControl_Suite3, PhoneInfo_Suite3 or Security_Suite3 object.

Remarks

The **Terminate** method must always be called before releasing the latest occurrence of STTNGS3A_SLib library's object.

Example

This example uses the **Terminate** method to disconnect from the component server before assigning **Nothing** to the PhoneInfo_Suite3 object variable i.e., releasing the interface object variable.

```
Private PhoneInfo As STTNGS3A_SLib.PhoneInfo_Suite3

Private Sub Form_Load()
    'connect to the PhoneInfo_Suite3 object
    Set PhoneInfo = New STTNGS3A_SLib.PhoneInfo_Suite3
End Sub

Private Sub Form_Unload(Cancel As Integer)
    'disconnect from the component server
    PhoneInfo.Terminate
```

1.5 Examples

The sample code blocks in this document are written in Visual Basic. These code blocks have been cut from the example applications, which are collected in the folder named Nokia Phone\Vb. The subfolders under this folder are named according to chapters in this document. There is at least one example Visual Basic application per each library.

Note: Because of the limited row width of this document some lengthy code lines that are meant to be on single line may be wrapped onto several lines.

Note: The examples in this document are not complete programs and cannot be used as-is. At the very least, you'll need to use the startup sequence detailed in 3.2.5 and usually instantiate objects etc. before the example will work.

There are also some miscellaneous Visual Basic, Visual C++ and Delphi applications. These applications are situated in the Miscellaneous subfolder of Nokia Phone\Vb, Nokia Phone\VC++, and Nokia Phone\Delphi folders respectively.

The example applications are collections of code samples implemented to describe the type definitions, properties, methods, functions, and events of the libraries. Therefore, the applications are not the most logical in structure and should not be taken as an example of good programming style.

1.6 References

The following documents are referred to in this document:

[SSM3] Nokia Smart Messaging Specification. The latest version of this document can be found on Forum Nokia's web site (www.forum.nokia.com) under Forum Nokia, Smart Messaging section. For entering the webpage registration is required.

[GSM 03.38] Digital cellular telecommunications system; Alphabets and language-specific information. The latest version of this document can be found on the ETSI web server (www.etsi.org/ipr).

[GSM 03.40] Digital cellular telecommunications system; Technical realization of the Short Message Service (SMS). The latest version of this document can be found on the ETSI web server (www.etsi.org/ipr).

2. NOKIA PC CONNECTIVITY SDK 2.0 INSTALLATION

2.1 Installation

2.1.1 General

- Installation includes Nokia PC Connectivity SDK 2.0 for Nokia Phones components
- Official name of the installation is Nokia PC Connectivity SDK 2.0 for Nokia Phones
- Nokia PC Connectivity SDK 2.0 is the name used in installation dialogs

2.1.2 Launching the Installation

The Nokia PC Connectivity SDK 2.0 for Nokia Phones installation is created with InstallShield 6.2 professional software. The Nclend2.exe can be extracted to any desired location for future use or it can be used as an installation. Note that the installation will start after the file extraction.

You can launch the Nokia PC Connectivity SDK 2.0 for Nokia Phones installation inside an InstallShield 6.2 script using the function DoInstall. In this case, the installation must be created with the same InstallShield 6.2 as the Nokia PC Connectivity SDK 2.0 for Nokia Phones installation.

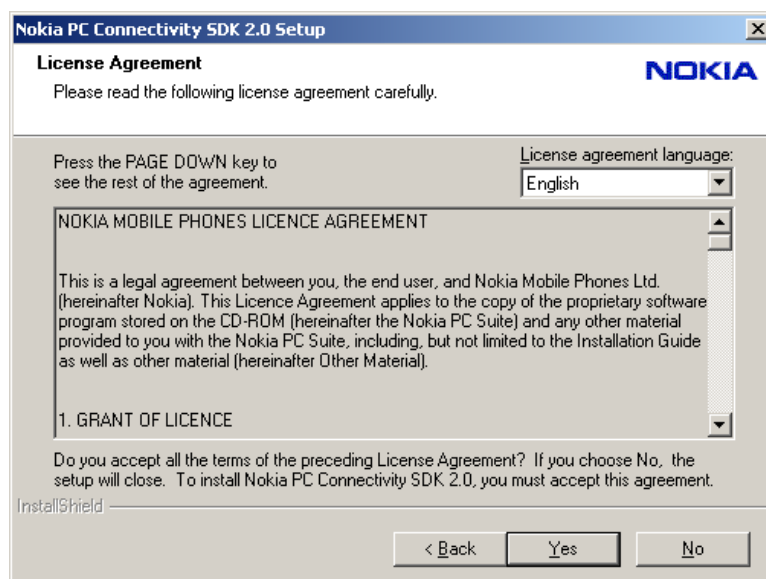
Another way to install Nokia PC Connectivity SDK 2.0 for Nokia Phones using InstallShield are with the functions LaunchApp or LaunchAppAndWait. You can launch either the setup.exe from the extracted nclend2 package or the Nclend2.exe directly.

The Nokia PC Connectivity SDK 2.0 for Nokia Phones installation will detect the connected phone and therefore it cannot be installed silently.

2.2 Installation Steps

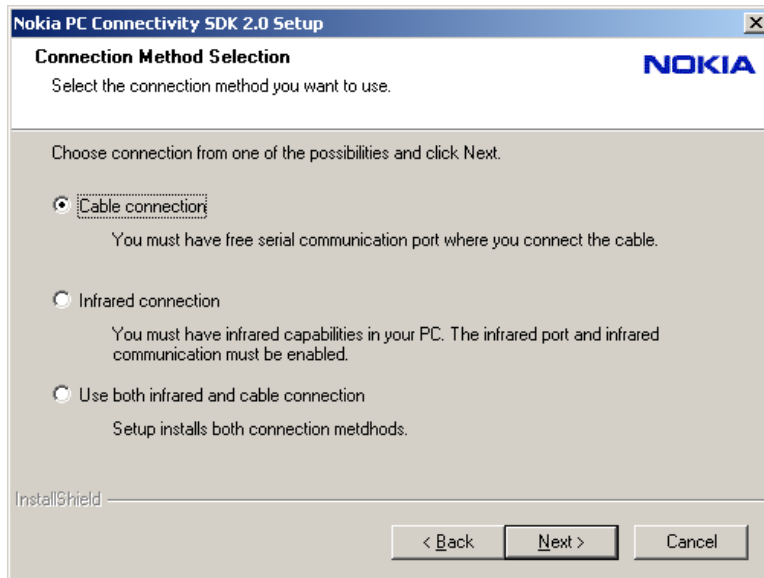
2.2.1 License Agreement

The user is asked to accept the terms of the license agreement otherwise installation will end.

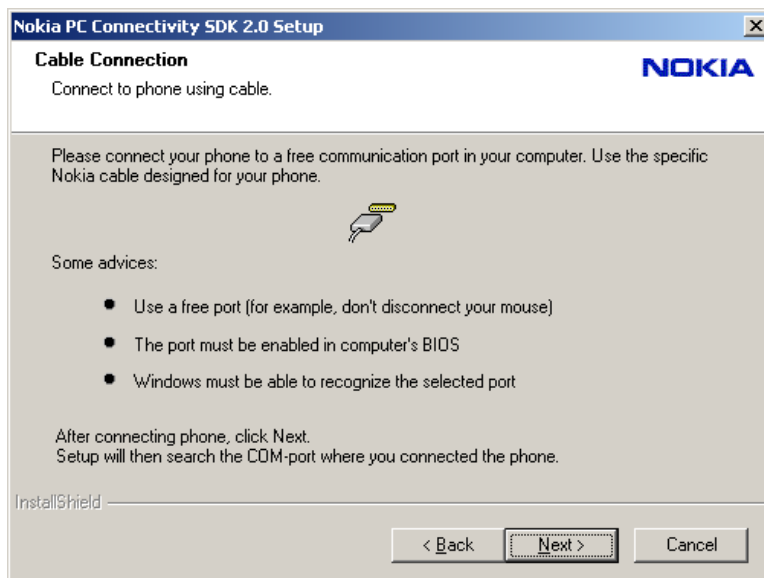


2.2.2 Connection Method Selection

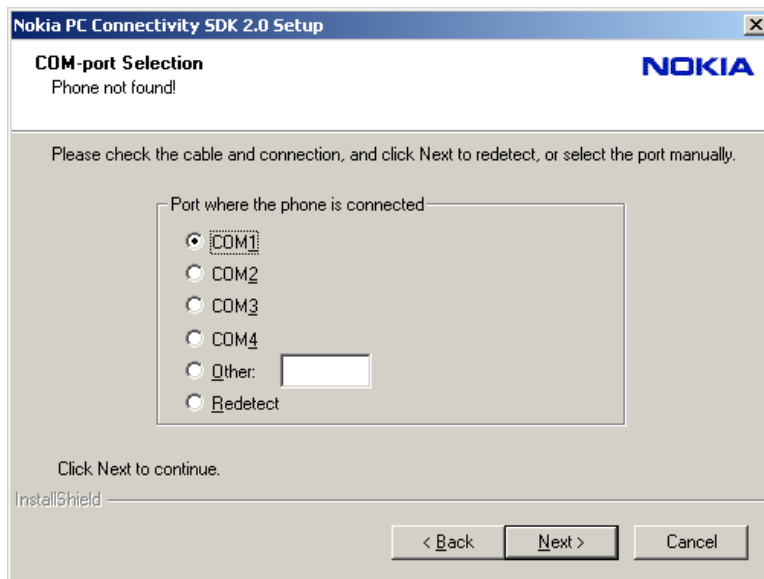
The user is able to choose the connection method from Cable or Infrared. If both the infrared and cable connections are selected, the infrared will be used but the cable ability is verified and the COM port number is stored at the registry.



If the cable connection or both connections are selected, the user is asked to connect the Nokia phone to the specific Nokia serial cable that is connected to the COM port.



The user is able to choose the COM port from the dialog if auto detect was not able to detect it.



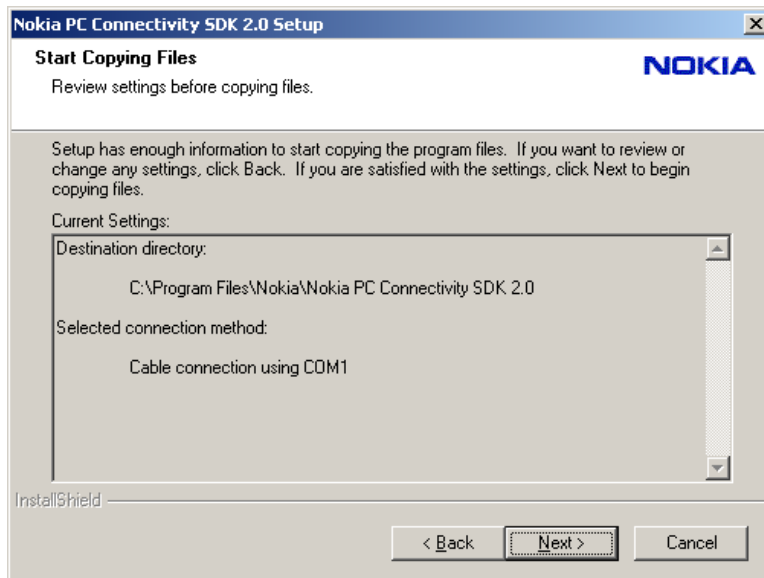
If the infrared connection or both connections are selected, the user is asked to enable the phone IR from the menu and point the infrared window of the phone to the PC's infrared window.



Selected Connection can be found from the registry under key
HKLM\Software\Nokia\MediaWrapper; Media. IrDA = Infrared and RS232 = Cable.

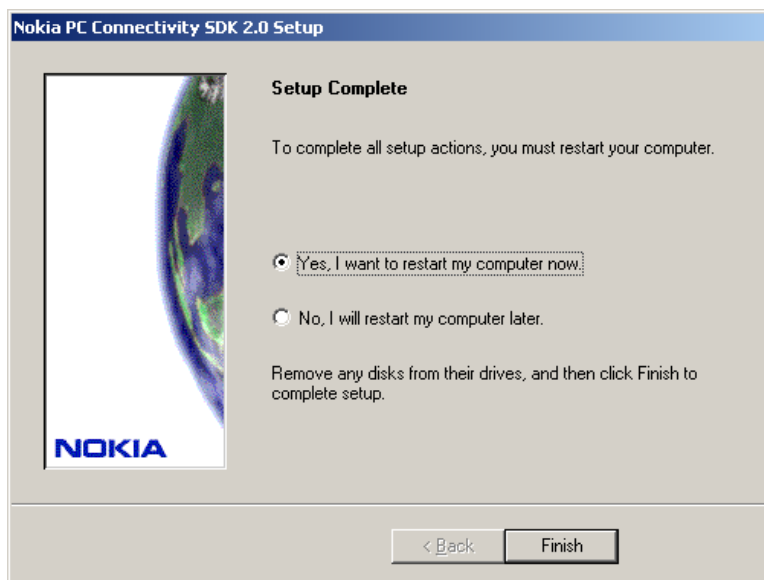
2.2.3 Start Copying Files

Shows the destination folder and selected connection method and port.



2.2.4 Setup Complete

The user is informed that the installation is complete. By default, the user is recommended to restart the computer before using the program.



2.3 Installation Components and Locations

The installation program will copy the following components to the following locations:

Directory: <Common Files>\Nokia\Adapters		
Cal3aS.dll	Self-Reg	Shared
SCM3aS.dll	Self-Reg	Shared
Sms3aS.dll	Self-Reg	Shared

Stngs3aS.dll	Self-Reg	Shared
NclWAP.dll	Self-Reg	Shared
Cgsgsupp.cfg		Shared
Directory: <Common Files>\Nokia\MPAPI		
mpapi3s.exe	Self-Reg	Shared
Mpapi3ps.dll	Self-Reg	Shared
MediaWrapper3s.dll	Self-Reg	Shared
MPAPI3.ppm		Shared
Directory: <WINSYSDIR>		
Msvcirt.dll		Shared
Mfc42.dll	Self-Reg	Shared
Msvcrtdll		Shared
Ddao35.dll		Shared
Inetwh32.dll		Shared
Ws2_32.dll (only Win9x)		Shared
Directory: <WINSYSDIR>\Drivers		
NokiaSuite3.sys		Windows NT&2000
Directory: <WINSYSDIR>		
NCDS3.VXD		Windows 9X

2.3.1 DCOM95 Installation

If the destination Operating System is Windows 95 and the dcom95 is not installed, the installation will run the dcom95.exe installation. The installation will also update the dcom95 if the version number is smaller than 1.2.

3. LANGUAGE SUPPORT

3.1 Basic Concepts

The description of the interfaces exposed by the component library is published in the object library (i.e., in the type library). The object library defines the methods along with their parameters and return values in standard binary format.

Object libraries can be viewed in readable format with the OLE/COM Object Viewer or with Visual Basic's Object Browser. The OLE/COM Object Viewer is distributed with the Microsoft Platform SDK. It can also be downloaded from Microsoft's web site.

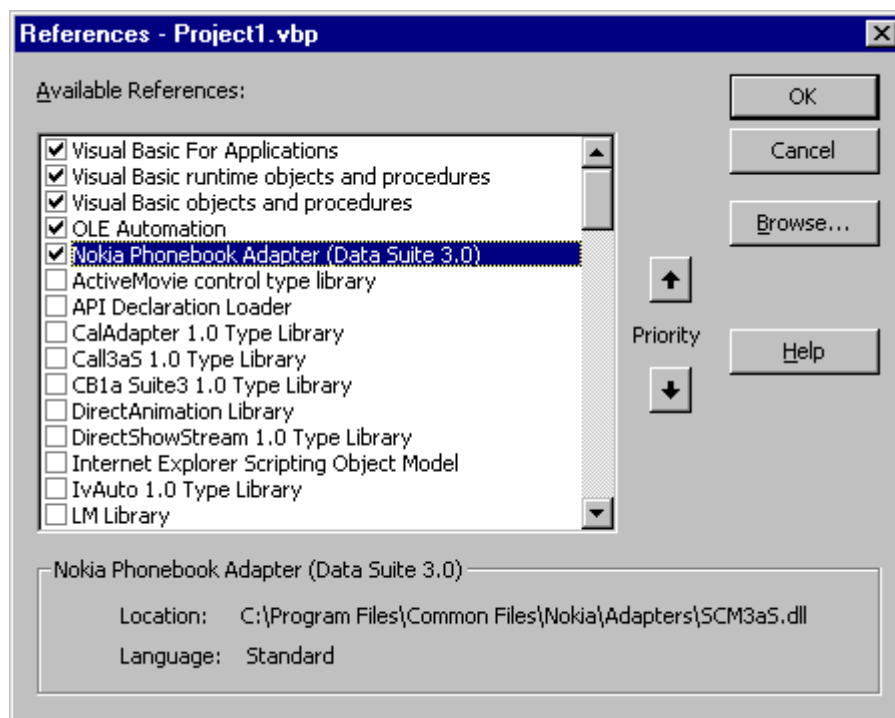
Several environments support object libraries. In the following chapters, the usage of libraries with Visual Basic, Visual C++, and Delphi is discussed in more detail.

3.2 Using Libraries with Visual Basic

3.2.1 Referencing Object Libraries

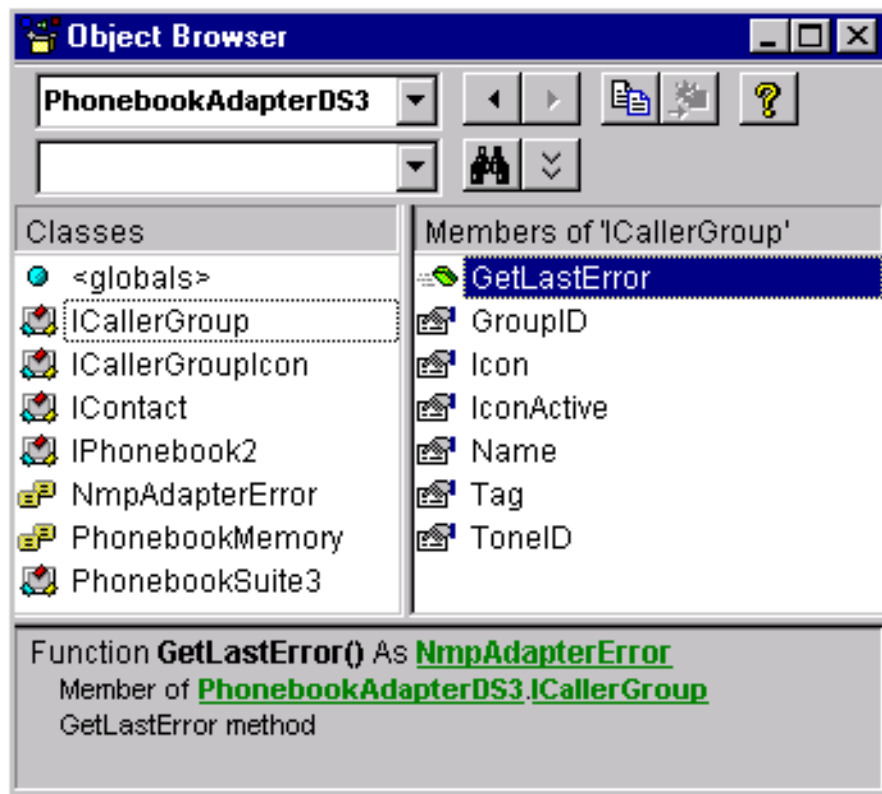
Note: The following applies to Microsoft Visual Basic 5.0.

Before using objects supplied by another application in Visual Basic, you must set a reference to its object library. A reference is set in the References dialog box activated by selecting the **Project | References...** menu item.



Select the needed libraries by selecting the check box next to its name. You should select only the libraries needed to minimize the number of object references Visual Basic must resolve, thus reducing the time it takes your project to compile.

After you set a reference to an object library by selecting the check box next to its name, you can find a specific object and its properties, methods, functions, events and constants in the Object Browser which is activated by selecting the **View | Object Browser** menu item.



In Visual Basic, the default interfaces (also called primary interfaces) are accessed directly through the name of the component class in which they belong. For example, in the Object Browser the methods of the PhonebookSuite3 interface (8.2.1), which is the default interface of the PhonebookSuite3 component class (8.2), are shown under the PhonebookSuite3 class.

Note: Some declarations in Nokia PC Connectivity SDK include unsigned C-language data types. Object Browser will show these as "<unsupported variant type>", and such methods or properties can not be accessed from VB code (at least up to VB5).

3.2.2 Referencing Primary Interfaces

To set a reference to the primary interface of another application's component class, first declare an object variable and then create a new reference with the **New** keyword.

Below, a reference to the PhonebookSuite3 interface of the PhonebookSuite3 component class is set.

```
Dim Phonebook As PhonebookAdapterDS3.PhonebookSuite3
Set Phonebook = New PhonebookAdapterDS3.PhonebookSuite3
```

(This corresponds to the CoCreateInstance and QueryInterface Win32 API calls.)

To remove a reference to the object, set the reference to **Nothing**.

```
Set Phonebook = Nothing
```

(This corresponds to the Release Win32 API call.)

3.2.3 Referencing Secondary Interfaces

References to secondary interfaces in the same component class are set through the primary interface reference.

```
Dim Phonebook2 As PhonebookAdapterDS3.IPhonebook2
Set Phonebook2 = New PhonebookAdapterDS3.PhonebookSuite3
```

(This corresponds to the QueryInterface Win32 API call.)

3.2.4 Referencing Outgoing Interfaces

Note: Outgoing interfaces are also often called connection points, event interfaces, notify interfaces, or source interfaces.

To set a reference to the primary outgoing interface, declare an object variable with the **WithEvents** keyword. This keyword indicates that the object is used to respond to events triggered by the component.

In the following example, reference to the primary outgoing interface is set.

```
Dim WithEvents PhoneInfoEvents As STTNGS3A_SLib.PhoneInfo_Suite3
Set PhoneInfoEvents = New STTNGS3A_SLib.PhoneInfo_Suite3
```

References to secondary outgoing interfaces in the same component class are set through the primary outgoing interface reference as shown in Chapter 3.2.3.

Note: In order for client applications to run correctly on Win9x operating systems, they must declare all the events of the referenced outgoing interface in their code. If the event does not need any special response from the client's side, don't add any code to the event body:

```
PhoneInfoEvents_BatteryNotify(ByVal bLevel As Long, ByVal bCharging As Boolean)
    'response to be added later if needed
End Sub
```

3.2.5 Starting Sequence

After setting the reference for the first component of each library, it takes some time for the lower parts of the library to start up and initialize. This may cause the first requests to the library to fail if the requests are made too soon after startup. Therefore, the startup sequence below is suggested. The example is written with Visual Basic, but the same functionality holds with Visual C++ and Delphi.

1. Create a reference to the PhoneInfo_Suite3 interface.
2. Start listening to the PhoneInfo_Suite3 events.

```
Private bAttached As Boolean
Private WithEvents PhoneInfoEventsSuite As _
    STTNGS3A_SLib.PhoneInfo_Suite3

Private Sub Form_Load()
```

```
Dim devN As STTNGS3A_SLib.DevNotifyOpt

Set PhoneInfoEventsSuite = _
New STTNGS3A_SLib.PhoneInfo_Suite3
PhoneInfoEventsSuite.StartListeningPhoneInfoEvents
PhoneInfoEventsSuite.get_DeviceStatus (devN)

If devN = ATTACHED Then
    bAttached = True
End If

If devN = REMOVED Then
    bAttached = False
End If

End Sub

Private Sub Form_Unload(Cancel As Integer)
    PhoneInfoEventsSuite.Terminate
End Sub

Private Sub PhoneInfoEventsSuite_BatteryNotify( _
    ByVal bLevel As Long, ByVal bCharging As Boolean)
    'Must be implemented
End Sub

Private Sub PhoneInfoEventsSuite_DeviceNotify( _
    ByVal val As STTNGS3A_SLib.DevNotifyOpt)
    'Must be implemented
End Sub
```

3. Ask if the phone is attached to or removed from the PC. This is done by the method `get_DeviceStatus`, which gives a value of type `DevNotifyOpt`. If this value is `ATTACHED`, the phone is attached to the PC. If this value is `REMOVED`, then either the phone is not connected to the PC or the components are not ready yet.

These values of type `DevNotifyOpt` can also be parameters of a private event handler, e.g., `PhoneInfoEventsSuite_DeviceNotify`. If the status of the phone changes, it is noticed by this method.

```
Private Sub PhoneInfoEventsSuite_DeviceNotify( _
    ByVal val As STTNGS3A_SLib.DevNotifyOpt)

    Select Case val
    Case ATTACHED
        bAttached = True
    Case DISCONNECTED
        bAttached = False
    Case REMOVED
        bAttached = False
    Case UNKNOWN
        bAttached = False
    End Select

End Sub
```

You should use both `get_DeviceStatus` and the `DeviceNotify` events to check that the components are ready.

When the phone is attached, the interfaces of the adapter can be used.

3.2.6 Event Handlers

StartListeningEvents method of an adapter object must be called before your program can receive events from the adapter. If StartListeningEvents is used, you must call Terminate method before deleting the adapter object. If StartListeningEvents is called you must also implement all event handlers for the adapter object.

3.2.7 Dispatch Interfaces

Dispatch interfaces are not generally implemented in Nokia PC Connectivity SDK. Some components have dispatch interfaces, but we do not recommend using them. You can *not* declare variables in the following way:

```
Dim Adapter As Object
Set Adapter = New SMS3ASuiteLib.SMS_SuiteAdapter
```

3.2.8 Error Handling

The component library uses various error codes to communicate with the application. (see 4.7.7 for description of the error codes). These can be handled by using the Visual Basic's **On Error Goto** statement, and implementing an error handler that calls the **GetLastError** function of the library, which generated the error.

Error handlers should use **Resume** after handling the error, for example:

```
On Error GoTo errortrap
    Phonebook.DeleteContact MEMORY_ME, 1
    GoTo Success
errortrap:
    Errorcode = Phonebook.GetLastError
    MsgBox "NmpError: " & Errorcode & " VB Error: " & Err.description
    Resume Success
Success:
```

Although the components do not usually raise errors using Visual Basic's **Err** object, this sometimes happens. Therefore **Resume** statement should be used to clear the **Err** object in all errorhandlers. If this is not done, errors will accumulate and cause Visual Basic to show apparently random "Automation Error, Unknown Error" messageboxes. See MSDN Library Visual Studio 6.0 help on "Resume statement" for more information.

NmpAdapterErrors are maintained on a per library basis and **GetLastError** always returns the latest error that occurred. Therefore it should only be called inside an error handler. See 4.7.7 and 4.2.1.3 for more information.

3.3 Using Libraries with Visual Basic for Applications

Although Visual Basic (VB) and Visual Basic for Applications (VBA) are very similar – they share the same core language and form engine – they differ in one important way. By using VB, one can create stand alone executables to be distributed to users but VBA applications always require the product (MS Excel, MS Access, MS Word, MS Project, etc.) with which they were

created. In addition to this, VBA has different object model and automation in each different product.

Using Nokia PC Connectivity SDK from VBA applications is mostly similar to VB, since only slight differences can be found.

Variables of the type defined inside the component library cannot be declared in the VBA application. For example, in the Phonebook Library, the phone memory type is defined as follows:

```
enum {  
    MEMORY_ME  
    MEMORY_SIM  
    ...  
} PhonebookMemory;
```

In VB application, a variable holding the phone memory type can be declared as such.

```
'VB application  
  
Private MemoryType As PhoneBookAdapterDS3.PhonebookMemory  
  
Private Sub ShowMemory()  
    Call PhoneBook.GetMemoryInUse(MemoryType)  
    ...  
End Sub
```

In VBA application, this type is replaced by the closest VB integral data type, which in this case is Long.

```
'VBA application  
  
'Compile error: Automation type not supported in VB ->  
'Private MemoryType As PhoneBookAdapterDS3.PhonebookMemory  
  
OK ->  
Private MemoryType As Long  
  
Private Sub ShowMemory()  
    Call PhoneBook.GetMemoryInUse(MemoryType)  
    ...  
End Sub
```

The same rule applies to other data types and other libraries too.

3.4 Using Libraries with Visual C++

Note: The following applies to Visual C++ 5.0.

Before using objects supplied by another application in Visual C++, you must incorporate information from its object library. In Visual C++, this is performed by using the **#import** directive.

The following line shows how to import information from the settings library.

```
#import "c:\program files\common files\nokia\\  
    adapters\Stngs3aS.dll"
```

The **#import** directive instructs the C++ compiler to process the designated type library, converting the contents to C++ code that describes the interfaces contained within the object library. When the compiler encounters the **#import** directive, it generates two header files that reconstruct the object library's contents in C++ source code.

The primary and the secondary header files have the same name as the object library but with the .tlh (type library header) and the .tli (type library implementation) extensions, respectively.

The primary header file contains the abstract base class definitions for the interfaces. The classes expose the raw interface with pure virtual member functions (prefixed with `raw_` by default) plus the non-virtual inline member functions (no prefix) that wrap them. The client applications should call the wrapper member functions.

The object library contents in the primary header file are defined in a namespace. The namespace name is specified in the library statement of the original IDL file. If you are importing only the single object library and you do not want the namespace to be generated by the compiler, specify the `no_namespace` attribute with the **#import** directive.

```
#import "c:\program files\common files\nokia\\  
    adapters\Stngs3aS.dll" no_namespace
```

It is good programming style to use namespaces, especially when importing several object libraries in the same module, in order to avoid naming and re-definition conflicts between libraries. If you want access to the complete namespace easily without using explicit access qualification (`namespace::`), use the `using` directive, e.g., in the following way:

```
using namespace STNGS3A_SLib;
```

Note: If you want to rename the namespace that contains the contents of the type library, use the `rename_namespace` attribute of the **#import** directive.

The secondary header file contains the implementations of compiler-generated wrapper member functions. The secondary header file is included in the primary header file by using the **#include** directive.

Note: Components in Nokia PC Connectivity SDK are not IDispatch interface providers. Thus, they can not be added to your MFC project by using the ClassWizard utility.

3.4.1 Referencing Incoming Interfaces

In C++ terminology, there are no concepts like primary and secondary interfaces. All the incoming interfaces are referenced by using either one of the methods described below.

3.4.1.1 Referencing Interfaces by Using Win32 API

Referencing interfaces in the component class is carried out by using a rather limited set of Win32 API functions.

After initializing the COM Library, the instance of the component class must be created by using the CoCreateInstance API method. The desired component class is identified by giving the class identifier (CLSID) as a parameter. The method returns a pointer to the requested component.

```
// initialize COM
::CoInitializeEx( NULL, COINIT_MULTITHREADED );

IUnknown* pUnknown;
HRESULT hr( ::CoCreateInstance(
    _uuidof( PhoneInfo_Suite3 ),
    NULL,
    CLSCTX_INPROC_SERVER,
    _uuidof( IUnknown ),
    ( void * * ) &pUnknown );

if ( FAILED( hr ) ) { ...
```

Since the components of the Nokia PC Connectivity SDK are implemented as DLLs, and are thus loaded in the client's process space, the third parameter of the CoCreateInstance method is set to CLSCTX_INPROC_SERVER, indicating that the code that creates and manages objects of this class, runs in the same process as the caller of the function.

The pointer to the desired interface is queried by using the QueryInterface method of the requested component.

```
IPhoneInfo *pIPhoneInfo;

// check the support for IPhoneInfo interface
hr = pUnknown->QueryInterface(
    _uuidof( IPhoneInfo ),
    ( void * * ) &pIPhoneInfo );

if ( FAILED( hr ) ) { ...
```

Now the methods of the interface are ready to be called, e.g., to check device status.

```
try {
    DevNotifyOpt opt;
    hr = pIPhoneInfo->get_DeviceStatus(&opt);
}
catch ( _com_error ) {

    long lErrorCode( pIPhoneInfo->GetLastError() );
    // handle error here...
}
```

Finally, when the interface object is no longer needed, it can be released and the COM Library can be uninitialized.


```
// release the IPhoneInfo interface object  
pIPhoneInfo->Release();
```

```
// uninitialized COM  
::CoUninitialize();
```

3.4.1.2 Referencing Interfaces by Using Smart Pointers

In Visual C++, predefined smart pointer template classes may also be used to instantiate interface objects. Smart pointers encapsulate the interface pointers and wrap the CoCreateInstance, AddRef, QueryInterface, and Release calls. In this chapter, examples of smart pointer usage in instantiation and usage of interface objects are given.

The definition of interface pointer is done by using the `_COM_SMARTPTR_TYPEDEF` macro (defined in `comdef.h`). The interface name and identifier are given as parameters. The compiler then defines an interface pointer type with name `<InterfaceName>Ptr`. Thus, in this example, the `IPhoneInfo` interface pointer type will be `IPhoneInfoPtr`.

To instantiate a `IPhoneInfo` interface object, a variable of type `IPhoneInfoPtr` must be declared and the identifier of the component containing the instantiated interface must be given as a parameter to the constructor. Now, the interface pointer can be used to call methods of `IPhoneInfo` interface.

The following C++ code clip defines an `IPhoneInfo` interface pointer and instantiates an interface object.

```
_COM_SMARTPTR_TYPEDEF(  
    IPhoneInfo,  
    _uuidof( IPhoneInfo ) );  
  
// explicit namespace access qualification needed here to  
// avoid ambiguity  
STTINGS3A_SLib::IPhoneInfoPtr gPhoneInfo(  
    _uuidof( PhoneInfo_Suite3 ) );
```

After initializing the COM Library, the object can be used to check device status.

```
// initialize COM  
::CoInitializeEx( NULL, COINIT_MULTITHREADED );  
  
try {  
    DevNotifyOpt opt;  
    hr = pIPhoneInfo->get_DeviceStatus(&opt);  
}  
catch ( _com_error ) {  
    long lErrorCode( gPhoneInfo->GetLastError() );  
    // handle error here...  
}
```

Finally, the object is released and the COM Library uninitialized.

```
// release the interface object  
gPhoneInfo->Terminate();  
gPhoneInfo = NULL;  
  
// uninitialized COM  
::CoUninitialize();
```

3.4.2 Referencing Outgoing Interfaces

Note: Outgoing interfaces are also often called connection points, event interfaces, notify interfaces, or source interfaces.

In order to use a connectable object, the client must implement a sink class to include an object that will receive and respond to events.

```
class CPhoneInfoNotify : public IPhoneInfoNotify {
public:
    // IUnknown
    ULONG __stdcall AddRef();
    ULONG __stdcall Release();
    HRESULT __stdcall QueryInterface( REFIID, void** );

    // ICallNotify
    HRESULT __stdcall raw_DeviceNotify(
        DevNotifyOpt val);
    ...

    CPhoneInfoNotify() : m_cRef( 0 ) {}

private:
    long m_cRef;
};
```

The sink class implements the IUnknown methods in the standard way. The class also overrides the pure abstract methods (prefixed with `raw_` by default) of IPhoneInfoNotify defined in the generated type library header (.tlh) file. The client is free to implement these methods as they please. Either way, the successful use of these methods should return `S_OK`.

Below, an example implementation of DeviceNotify notification is presented.

```
HRESULT CPhoneInfoNotify::raw_DeviceNotify(DevNotifyOpt val) {
    if (val == ATTACHED) {
        MessageBox( 0, "The phone was attached",
            PROCESS_NAME, MB_OK );
    }
    else if (val == REMOVED) {
        MessageBox( 0, "The phone was removed",
            PROCESS_NAME, MB_OK );
    }
    return S_OK;
}
```

Note: If you do not like the `raw_` prefix of raw interface members, you can change the prefix by using the `raw_method_prefix` attribute of the `#import` directive.

Establishing a connection between a client and a connectable object in VC++ starts by initializing the COM Library and by creating an instance of the component object containing the desired outgoing interface with the `CoCreateInstance` method. See code sample in Chapter 3.4.1.1 for details.

The next step for the client is to query for `IConnectionPointContainer` on the object to determine if the object is connectable.

```
IConnectionPointContainer * pConnectionPointContainer;  
hr = pUnknown->QueryInterface(  
    _uuidof( IConnectionPointContainer ),  
    ( void * * ) &pConnectionPointContainer );  
  
if ( FAILED( hr ) ) { ...
```

Then the client has to obtain a pointer to the `IConnectionPoint` interface on a connection point within the connectable object.

```
IConnectionPoint * pConnectionPoint;  
hr = pConnectionPointContainer->FindConnectionPoint( _uuidof( IPhoneInfoNo-  
    tify ), &pConnectionPoint );  
  
if ( FAILED( hr ) ) { ...
```

The sink object must then be instantiated, the pointer of which is sent to the component. After this, there is an advisory relationship between the client's sink and the connectable object.

```
CPhoneInfoNotify * pPhoneInfoNotify = new CPhoneInfoNotify;  
DWORD dwCookie;  
  
pConnectionPoint->Advise(  
    ( IUnknown * ) pPhoneInfoNotify,  
    &dwCookie );
```

Now the relationship between client's sink and the connectable object has been established.

When the connection is no longer needed, it can be terminated.

```
pConnectionPoint->Unadvise( dwCookie );
```

Then, the client must free its hold on all other interfaces used.

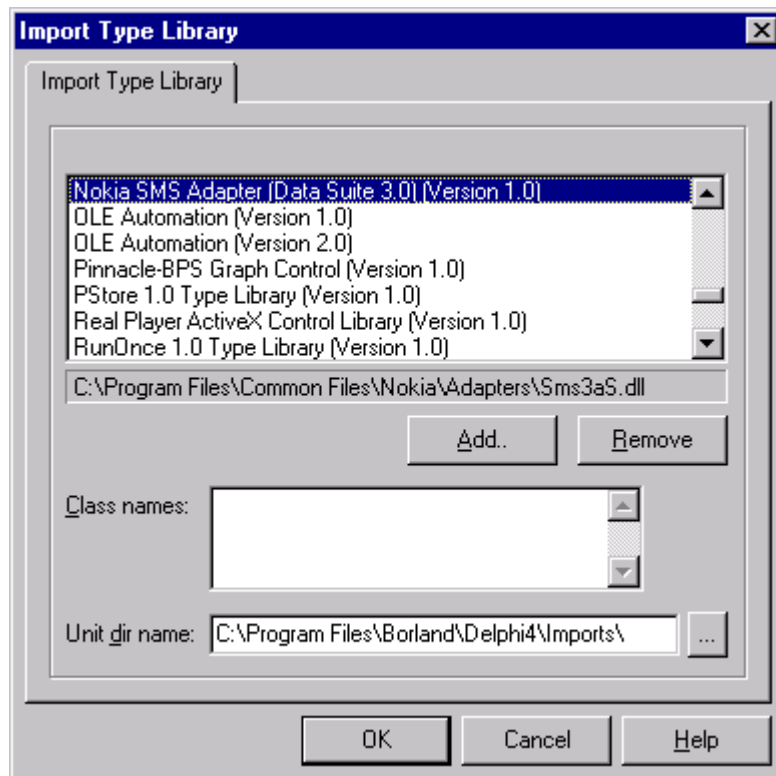
```
pConnectionPoint->Release();  
pConnectionPointContainer->Release();  
pUnknown->Release();
```

Finally, the COM Library needs to be uninitialized. Check Chapter 3.4.1.1 for details.

3.5 Using Libraries with Delphi

Note: The following applies to Delphi 4.0.

Before using objects supplied by another application in Delphi, you must import the corresponding object library, which is also often called type library. To import the library, open the dialog displayed below by selecting the **Project | Import Type Library...** menu item.



In this dialog, select the desired type libraries in the topmost panel and press OK to add them to your current Delphi project. The contents of the type library are placed inside a Delphi wrapper and stored in a .pas file, which has the same name as the type library. This file contains Pascal declarations for methods and types contained in the type library, and is added as a new unit in your Delphi project.

For example, importing type library SMS3ASuiteLib causes a SMS3ASuiteLib_TLB.pas file to be generated.

In the main unit of the client application, add the name of the generated unit to the `uses` clause to permit access to the methods and types contained within that unit.

```
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ComObj,
  SMS3ASuiteLib_TLB;
```

3.5.1 Referencing Primary Interfaces

A reference to the primary interface of the component is established by using the helper class found in the generated .pas file.

```
var
  FSMSSend: ISMSSend;

  FSMSSend := CoSMS_SuiteAdapter.Create;
```

The methods of the interface can now be called through this reference.

```
...
var
  SMS: IShortMessage;
  hRes: HRESULT;
begin
  hRes := FSMSSend.CreateShortMsg( SMS );
  ...
```

3.5.2 Referencing Secondary Interfaces

Reference to any secondary interface can be made through the QueryInterface method of the primary interface as follows:

```
var
  FSMSMemory: ISMSMemory;
  ...

hRes := FSMSSend.QueryInterface(ISMSMemory, FSMSMemory);
```

3.5.3 Referencing Outgoing Interfaces

Referencing outgoing interfaces requires the client application to implement an event sink class, which will be called by the server. The sink must implement IDispatch (and therefore IUnknown) interface in addition to the desired notify interface of the library. Since the outgoing interfaces are custom interfaces, only the QueryInterface method needs to be implemented in the sink class, all the other methods can be inherited. The event interface functions must be mapped to the sink's corresponding functions, where the actual event handler code is written.

```
TEventSink = class(TObject, IUnknown, IDispatch, INotify)
private
  ...

  // Map event interface function to sink class function
  function DoSomeNotify(i: integer): HRESULT; stdcall;
  function INotify.SomeNotify = DoSomeNotify;

  ...
end;
```

A connection to the server is obtained by the InterfaceConnect procedure.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  // Create server and event sink instances
  FServer := CoClass_Suite3.Create;
  FEventSink := TEventSink.Create;

  // Connect the sink to server through INotify interface
  InterfaceConnect( FServer,           // Event source
                    IID_INotify,       // Event interface IID
                    FEventSink,        // Event sink
                    FCookie );         // Reference parameter
end;
```

It is also recommended that the connection to the server is closed using the `InterfaceDisconnect` procedure.

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
    // Disconnect from the server
    InterfaceDisconnect( FEventSink, // Event source
                        IID_INotify, // Event interface IID
                        FCookie );    // Reference parameter
    FEventSink.Destroy;
    FServer.Terminate;
end;
```

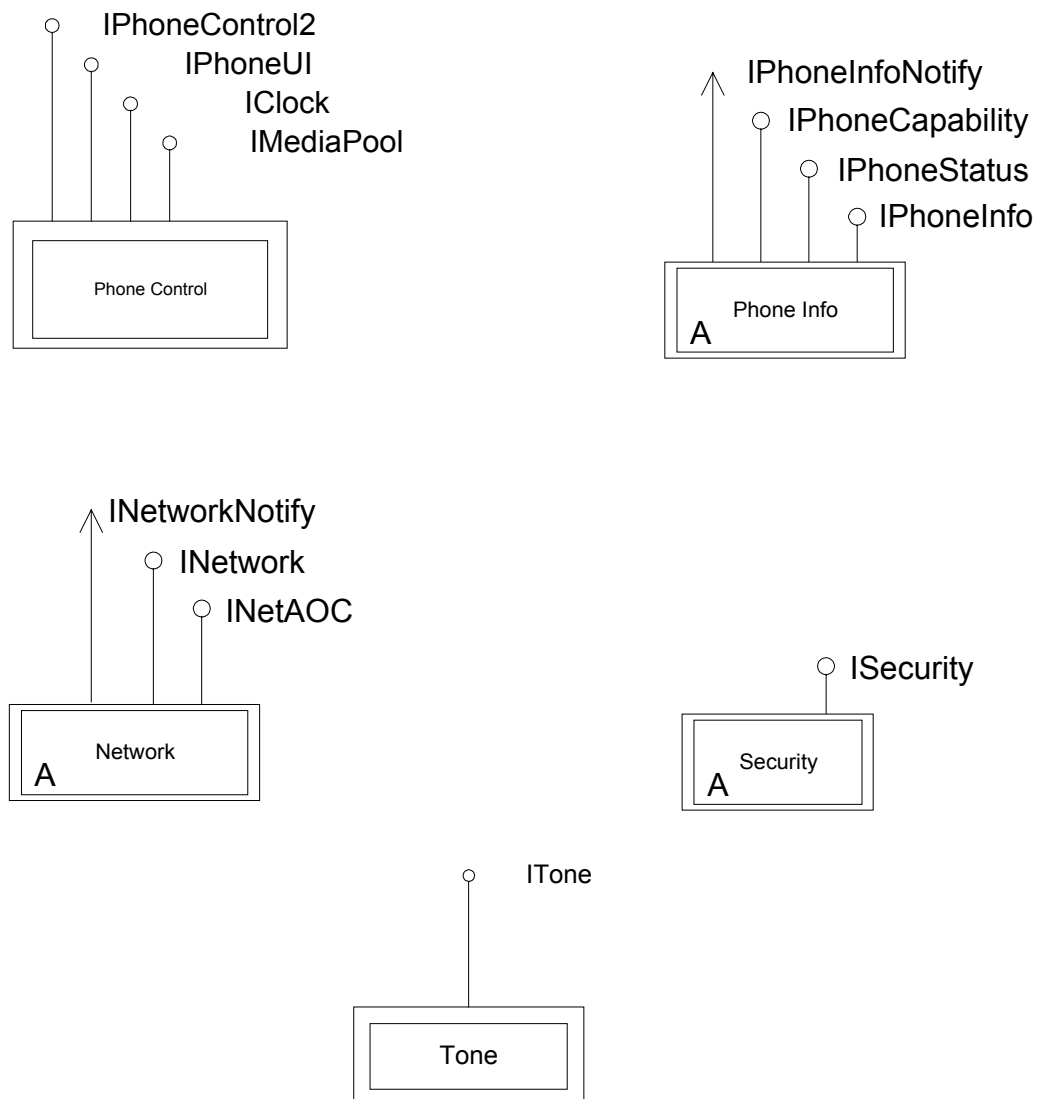
4. GENERAL SETTINGS LIBRARY (STTNGS3A_SLIB)

4.1 Overview

The General Settings Library contains components for accessing and changing the phone settings. All the settings encountered in the phone's menus can be accessed with this library, e.g., profile settings, clock, password changing and operator information.

4.1.1 Object Model

The picture below represents the Stngs3AS.dll module. The module contains implementation of five objects: PhoneControl, PhoneInfo, Network, Security and Tone. The objects are described in more detail in the next chapters.



Picture 1. Components and interfaces of the Settings module.

4.2 PhoneInfo_Suite3 Component

This component contains methods, functions and events for obtaining information about the phone, its status and capabilities. Changes in the phone's state are automatically updated to the component.

4.2.1 IPhoneInfo Interface

The IPhoneInfo interface gives static information about the phone device.

Method	Description
get_DeviceStatus	Queries if the phone is connected to the computer.
StartListeningPhoneInfoEvents	Informs the component object that the client is ready to receive outgoing method calls from the object.
GetLastError	Returns the code of the latest error.
Terminate	Disconnects from the component server.

Property	Description
HwVersion	Phone's hardware version number.
ProductCode	Phone's product code.
ProductType	Phone's product type.
SwVersion	Phone's software version number.

4.2.1.1 get_DeviceStatus

This method returns the status of the phone device.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo

Syntax

```
[helpstring("Device status query")]  
HRESULT get_DeviceStatus([out] DevNotifyOpt *pVal);
```

The **get_DeviceStatus** method syntax has these parts:

Part	Description
<i>pVal</i>	Output. A value of type DevNotifyOpt containing the device status.

Remarks

If the method fails, call the GetLastError function (4.2.1.3) to get extended error information.

Example

This method retrieves various pieces of information from the phone and places them on labels in the form. The following properties and functions are used in the method: SwVersion, ProductType, ProductCode, get_DeviceStatus and IMEICode.

```
Public InfoAdapter As New STTNGS3A_SLib.PhoneInfo_Suite3
```

```
Private Sub ReadInfo()  
    Dim PhoneStatus As STTNGS3A_SLib.IPhoneStatus  
    Dim CommErrorRetries As Integer  
    Dim devStatus As DevNotifyOpt
```

```
    On Error GoTo ErrorTrap
```

```
    Set PhoneStatus = InfoAdapter
```

```
    CommErrorRetries = 0
```

Beginning:

```
    'Read information to the Phone Information frame  
    With InfoAdapter  
        'Hardware version  
        HwVerLabel = .HwVersion
```

```
        'Software version  
        SwVerLabel = .SwVersion
```

```
        'Product Type  
        ProdTypeLabel = .ProductType
```

```
'Product Code
ProdCodeLabel = .ProductCode

'Device present
Call .get_DeviceStatus (devStatus)
Select Case devStatus
Case ATTACHED
    DeviceLabel = "Attached"
Case REMOVED
    DeviceLabel = "Removed"
Case UNKNOWN
    DeviceLabel = "Unknown"
End Select
End With

With PhoneStatus
    'IMEI Code
    IMEILabel = .IMEICode
End With

Exit Sub

ErrorTrap:
    'Inform the user about failure
    Dim err As NmpAdapterError
    err = InfoAdapter.GetLastError
    Select Case err
    Case errCommunicationError
        'If communication error, retry 5 times
        CommErrorRetries = CommErrorRetries + 1
        If CommErrorRetries > 5 Then
            MsgBox ("Communication error.")
        Else
            GoTo Beginning
        End If
    Case Else
        MsgBox ("Error #" & err & " in getting phone status.")
    End Select
End Sub
```

4.2.1.2 StartListeningPhoneInfoEvents

This method informs the component object that the client is ready to receive outgoing method calls (events) from the object concerning the status of the phone.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo,
STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneCapability,
STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Syntax

```
[helpstring("method StartListeningPhoneInfoEvents")]  
HRESULT StartListeningPhoneInfoEvents();
```

Remarks

If the method fails, call the **GetLastError** function (4.2.1.3) to get extended error information.

To be able to receive outgoing method calls, the client must implement and register an outgoing interface sink.

A client thread that has no message pump or a thread that may be blocked in some Win32 wait function is not allowed to declare itself as an event sink by calling **StartListeningPhoneInfoEvents**. Otherwise, the event calls queued by the component object become blocked and the component server is likely to deadlock.

Example

This timer handler is executed at 10-second intervals. When it is run for the first time, the password status is checked, the events and the form are initialized, and constant information is read from the phone. In successive method calls only the phone status and clock information are updated.

```
Public WithEvents InfoAdapter As STTNGS3A_SLib.PhoneInfo_Suite3  
  
Private Sub RefreshTimer_Timer()  
    'This section is run only at startup  
    If PageInited = False Then  
        'Check password status  
        Call CheckDevice  
  
        'We're interested in events  
        Call InfoAdapter.StartListeningPhoneInfoEvents  
  
        'Get the information to the form  
        Call ReadInfo  
        Call ReadCapabilities  
  
        'Release controls  
        Call ChangeControlStatuses(True)  
        MousePointer = vbDefault  
        PageInited = True
```

```
End If

'Status and clock are always updated
Call ReadStatus
Call ReadClock
End Sub
```

4.2.1.3 GetLastError

This method returns the Component Library's latest error code value.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo,
STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneCapability,
STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus **Syntax**

```
[helpstring("Get last error method method")]  
HRESULT GetLastError([out, retval] NmpAdapterError* pErr);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	An error code variable of type NmpAdapterError.

Remarks

The error code is maintained on a per library basis. For detailed information on error codes, see Chapter 4.7.7 (**NmpAdapterError**).

4.2.1.4 Terminate

This method disconnects from the component server.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo,
STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneCapability,
STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Syntax

```
[helpstring("Terminate")]  
    HRESULT Terminate()
```

Remarks

The **Terminate** method must always be called before releasing the latest occurrence of STTNGS3A_SLib library's object.

Example

This example uses the **Terminate** method to disconnect from the component server before assigning **Nothing** to the PhoneInfo_Suite3 object variable i.e., releasing the interface object variable.

```
Private PhoneInfo As STTNGS3A_SLib.PhoneInfo_Suite3  
  
Private Sub Form_Load()  
    'connect to the PhoneInfo_Suite3 object  
    Set PhoneInfo = New STTNGS3A_SLib.PhoneInfo_Suite3  
End Sub  
  
Private Sub Form_Unload(Cancel As Integer)  
    'disconnect from the component server  
    PhoneInfo.Terminate  
  
    'release the object  
    Set PhoneInfo = Nothing  
End Sub
```

4.2.1.5 HwVersion (r)

This property contains the phone's hardware version number.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo

Type: **String**

Syntax

```
[propget, helpstring("Hw-version property")]  
    HRESULT HwVersion([out, retval] BSTR *pVal);
```

Example

See 4.2.1.1

4.2.1.6 ProductCode (r)

This property contains the phone's product code.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo

Type: **String**

Syntax

```
[propget, helpstring("Product code property")]  
    HRESULT ProductCode([out, retval] BSTR *pVal);
```

Example

See 4.2.1.1

4.2.1.7 ProductType (r)

This property contains the phone's product type.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo

Type: **String**

Syntax

```
[propget, helpstring("Product type property")]  
    HRESULT ProductType([out, retval] BSTR *pVal);
```

Example

See 4.2.1.1

4.2.1.8 SwVersion (r)

This property contains the phone's software version number.

If this number is lower than 3.14, the driver gives a warning if Nokia 6110 is used.

Driver may also work with previous versions.

When testing some phone models, the SwVersion Property is accurate to one decimal place but in the phone's UI, it is accurate to two decimal places, for example, 5.1 and 5.17. In some phone models the number is the same with either method.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfo

Type: **String**

Syntax

```
[propget, helpstring("Sw-version property")]  
    HRESULT SwVersion([out, retval] BSTR *pVal);
```

Example

See 4.2.1.1

4.2.2 IPhoneInfoNotify Interface

Event	Description
BatteryNotify	Occurs when the phone's battery level changes or the phone is connected to a battery charger.
DeviceNotify	Occurs when the phone is attached to or removed from the PC.

4.2.2.1 Battery Notify

Occurs when the phone's battery level changes or the phone is connected to a battery charger.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfoNotify

Syntax

```
[helpstring("Battery notify")]  
HRESULT BatteryNotify([in] long bLevel, [in] VARIANT_BOOL bCharging);
```

The **BatteryNotify** event syntax has these parts:

Part	Description
<i>bLevel</i>	A long value containing the new battery level.
<i>bCharging</i>	A boolean value indicating whether the battery is charging.

Example

This event handler checks if the new battery value equals 1 and if that is the case, shows an informative message box.

```
Public WithEvents InfoAdapter As STTNGS3A_SLib.PhoneInfo_Suite3  
  
Private Sub InfoAdapter_BatteryNotify( _  
    ByVal bLevel As Long, _  
    ByVal bCharging As Boolean)  
  
    If bLevel = 1 Then  
        MsgBox "Battery is almost empty. Connect it to a charger."  
    End If  
  
End Sub
```

4.2.2.2 DeviceNotify

Occurs when the phone device is connected to or removed from the system.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneInfoNotify

Syntax

```
[helpstring("Device notify")]  
    HRESULT DeviceNotify([in] DevNotifyOpt val);
```

The **DeviceNotify** event syntax has these parts:

Part	Description
<i>val</i>	A value of type DevNotifyOpt containing the status of the device.

Example

This event handler terminates the application if the phone device is removed from the system.

```
Public WithEvents InfoAdapter As STTNGS3A_SLib.PhoneInfo_Suite3  
  
Private Sub InfoAdapter_DeviceNotify( _  
    ByVal val As STTNGS3A_SLib.DevNotifyOpt)  
  
    'If device was removed, exit from the application  
    If val = REMOVED Then  
        MsgBox "Phone device is not connected " & _  
            "to the system. Exiting..."  
        End  
    End If  
  
End Sub
```

4.2.3 IPhoneCapability Interface

IPhoneCapability interface contains information about the capabilities of the phone.

Method	Description
StartListeningPhoneInfoEvents	Informs the component object that the client is ready to receive outgoing method calls from the object.
Terminate	Disconnects from the component server.
GetLastError	Returns the code of the latest error.

Property	Description
IsCapability	Tells whether the phone implements a certain capability.

4.2.3.1 IsCapability (r)

This property tells whether the phone has a specified capability.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneCapability

Type: **Boolean**

Syntax

```
HRESULT IsCapability(CapabilityOpt property,  
                    [out, retval] VARIANT_BOOL* pVal);
```

The **IsCapability** property has these parts:

Part	Description
<i>property</i>	The queried capability. See CapabilityOpt.

Remarks

If the phone holds the specified capability, return value is True. Otherwise it is False.

Example

See 4.3.3.2

4.2.4 iPhoneStatus Interface

iPhoneStatus interface contains information about the current status of the phone.

Method	Description
StartListeningPhoneInfoEvents	Notifies the component object that the client is ready to receive outgoing method calls from the object.
GetLastError	Returns the code of the latest error.
Terminate	Disconnects from the component server.

Property	Description
BatteryCharging	A flag that indicates if the phone is currently charging.
BatteryLevel	Current battery level.
IMEIcode	The IMEI code of the phone.
IMSIcode	The IMSI code of the SIM.
PhoneStatus	Current status of the phone (off, idle, call active, not in service).
RFLevel	Current RF level.
SelectedMemory	Type of memory (phone, SIM, FDN) selected in the phone.
ServState	A flag that indicates if the phone is in service.

4.2.4.1 BatteryCharging (r)

This property tells whether the phone's battery is currently charging.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus Type: **Boolean**

Syntax

```
[propget, helpstring("Battery charging property")]  
    HRESULT BatteryCharging([out, retval] VARIANT_BOOL* pVal);
```

Example

See 4.2.4.6

4.2.4.2 BatteryLevel (r)

This property contains the current battery level.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Type: **Long**

Syntax

```
[propget, helpstring("Battery level property")]  
    HRESULT BatteryLevel([out, retval] long *pVal);
```

Example

See 4.2.4.6

4.2.4.3 IMEICode (r)

This property contains the IMEI code of the phone.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Type: **String**

Syntax

```
[propget, helpstring("IMEI-code property")]  
    HRESULT IMEICode([out, retval] BSTR *pVal);
```

Example

See 4.2.1.1

4.2.4.4 IMSICode (r)

This property contains the IMSI code of the currently inserted SIM.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Note: This property is not supported by Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones.

Type: **String**

Syntax

```
[propget, helpstring("IMSI-code property")]  
    HRESULT IMSICode([out, retval] BSTR *pVal);
```

4.2.4.5 PhoneStatus (r)

This property contains the phone's current status.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Type: PHONE_STATUS (See enumerated data types)

Syntax

```
[propget, helpstring("Phone status property")]  
    HRESULT PhoneStatus([out, retval] PHONE_STATUS *pVal);
```

Example

See 4.2.4.6

4.2.4.6 RFLevel (r)

This property contains the phone's current RF level.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Type: **Long**

Syntax

```
[propget, helpstring("RF level property")]  
    HRESULT RFLevel([out, retval] long *pVal);
```

Example

This method retrieves various pieces of information from the phone and places them onto labels on the form. The following properties and functions are used in the method: **BatteryCharging**, **BatteryLevel**, **SelectedMemory**, **PhoneStatus** and **ServState**.

```
Public InfoAdapter As New STTNGS3A_SLib.PhoneInfo_Suite3
```

```
Private Sub ReadStatus()  
    Dim PhoneStatus As STTNGS3A_SLib.IPhoneStatus  
  
    On Error GoTo ErrorTrap  
  
    Set PhoneStatus = InfoAdapter
```

Beginning:

```
'Read information to the Phone Status frame  
With PhoneStatus  
    'RF Level  
    RFLabel = Str(.RFLevel)  
  
    'Battery Level  
    If .BatteryCharging Then  
        BatteryLabel = "Charging"  
    Else:  
        BatteryLabel = .BatteryLevel  
    End If  
  
    'Selected Memory  
    Select Case .SelectedMemory  
        Case FDN  
            SelMemLabel = "FDN"  
        Case PHONE  
            SelMemLabel = "Phone"  
        Case SIM  
            SelMemLabel = "SIM"  
    End Select  
  
    'Status  
    Select Case .PhoneStatus  
        Case PS_IDLE  
            StatusLabel = "Idle"  
        Case PS_CALL_ACTIVE  
            StatusLabel = "Call active"
```

```
        Case PS_NO_SERVE
            StatusLabel = "No service"
        Case PS_OFF
            StatusLabel = "Off"
    End Select

    'Service State
    If .ServState Then
        ServStateLabel = "In service"
    Else
        ServStateLabel = "Not in service"
    End If
End With

Exit Sub

ErrorTrap:
    'Inform the user about failure
    MsgBox ("Error #" & PhoneStatus.GetLastError & " in getting phone
status.")
End Sub
```


4.2.4.7 SelectedMemory (rw)

This property contains the currently selected memory type in the phone.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Type: MemTypeOpt

Syntax

```
[propget, helpstring("Selected memory property")]  
    HRESULT SelectedMemory([out, retval] MemTypeOpt *pVal);
```

```
[propput, helpstring("Selected memory property")]  
    HRESULT SelectedMemory([in] MemTypeOpt newVal);
```

Example

See 4.2.4.6

4.2.4.8 ServState (r)

This property contains the phone's service status.

Member of STTNGS3A_SLib.PhoneInfo_Suite3.IPhoneStatus

Type: **Boolean**

Syntax

```
[propget, helpstring("Service state property")]  
    HRESULT ServState([out, retval] VARIANT_BOOL* pVal);
```

Remarks

The value of this property is True when the phone is connected to a service, otherwise it is False.

Example

See 4.2.4.6

4.3 PhoneControl_Suite3 Component

The Phone Control Component offer is for controlling the user interface, profile settings and clock.

4.3.1 IPhoneControl2 Interface

The IPhoneControl2 interface contains information about the media, which can be used for the connection between the phone and the computer.

Method	Description
GetSupportedMedia	Check if specified media is supported.
GetCurrentMedia	Get the media currently in use.
ChangeMedia	Change the media.
ChangePort	Change the COM port used by the current media.

4.3.1.1 GetSupportedMedia

This method checks if specified media is supported or not.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneControl2

Type: MediaType

Syntax

```
[helpstring("GetSupportedMedia method")]  
HRESULT GetSupportedMedia([in, out] MediaType *mMedia);
```

The **GetSupportedMedia** method syntax has these parts:

Part	Description
<i>mMedia</i>	In/Output. Media to search for one of the MediaType enumeration values.

Remarks

Specifies whether media is supported; if not, mMedia is set to MEDIA_UNKNOWN

4.3.1.2 GetCurrentMedia

This method checks which type of media is currently in use.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneControl2

Syntax

```
[helpstring("GetCurrentMedia method")]  
HRESULT GetCurrentMedia([in, out] MediaType *mMedia, [in, out] long  
*plPortNumber);
```

The **GetCurrentMedia** method syntax has these parts:

Part	Description
<i>mMedia</i>	In/Output. Media to search for one of the MediaType enumeration values.
<i>long</i>	COM port number used by the media.

4.3.1.3 ChangeMedia

This method changes the currently used media.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneControl2

Syntax

```
[helpstring("ChangeMedia method")]  
HRESULT ChangeMedia([in] MediaType mMedia);
```

The **ChangeMedia** method syntax has these parts:

Part	Description
<i>mMedia</i>	In. Media to change to one of the MediaType enumeration values.

4.3.1.4 ChangePort

This method changes the communication port used by media.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneControl2

Syntax

```
[helpstring("Change COM port used by the media")]  
HRESULT ChangePort([in] long lPortNumber);
```

The **ChangePort** method syntax has these parts:

Part	Description
<i>long</i>	In. The number of COM port.

4.3.2 iPhoneUI Interface

The iPhoneUI interface contains properties, methods and functions for setting up the phone device's user interface.

Method	Description
AllowReplyCost	A flag that indicates whether it is allowed for the recipients of user's short messages to reply at the cost of the user.
DeliveryReports	A flag that indicates whether the phone should request a delivery report for sent short messages.
OwnNumberSending	A flag that indicates whether the user's phone number should be shown to the recipient of a call.
SpeedDialing	A flag that selects speed dialing.
WelcomeText	The text that appears when phone is turned on.

4.3.2.1 AllowReplyCost (rw)

This property indicates whether it is allowed for the recipients of user's short messages to reply at the cost of the user.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneUI

Type: **Boolean**

Syntax

```
[propget, helpstring("Allow reply cost property")]  
    HRESULT AllowReplyCost([out, retval] VARIANT_BOOL *pVal);  
[propput, helpstring("Allow reply cost property")]  
    HRESULT AllowReplyCost([in] VARIANT_BOOL newVal);
```

4.3.2.2 DeliveryReports (rw)

This property tells whether the phone should request a delivery report for sent short messages.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneUI

Type: **Boolean**

Syntax

```
[propget, helpstring("DeliveryReports property")]  
    HRESULT DeliveryReports([out, retval] VARIANT_BOOL *pVal);  
[propput, helpstring("DeliveryReports property")]  
    HRESULT DeliveryReports([in] VARIANT_BOOL newVal);
```

4.3.2.3 OwnNumberSending (rw)

This property tells whether the user's phone number should be shown to the recipients of user's calls.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneUI

Type: **EnchBoolean**

Syntax

```
[propget, helpstring("Own number sending property")]  
    HRESULT OwnNumberSending([out, retval] EnchBoolean* pVal);  
[propput, helpstring("Own number sending property")]  
    HRESULT OwnNumberSending([in] EnchBoolean newVal);
```

4.3.2.4 SpeedDialing (rw)

This property tells whether the speed dialing is activated in the phone.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneUI Type: **Boolean**

Syntax

```
[propget, helpstring("Speed dialling property")]  
    HRESULT SpeedDialing([out, retval] VARIANT_BOOL* pVal);  
[propput, helpstring("Speed dialling property")]  
    HRESULT SpeedDialing([in] VARIANT_BOOL newVal);
```

4.3.2.5 WelcomeText (rw)

This property contains the text that will appear on the screen when the phone has been turned on.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IPhoneUI

Type: **String**

Syntax

```
[propget, helpstring("Welcome text property")]  
    HRESULT WelcomeText([out, retval] BSTR *pVal);  
[propput, helpstring("Welcome text property")]  
    HRESULT WelcomeText([in] BSTR newVal);
```

4.3.3 IClock Interface

The IClock interface encapsulates the phone's clock. The interface handles reading and setting both the time and the alarm.

Method	Description
GetAlarm	Returns the status and the time of alarm.
SetAlarm	Sets the status and the time of alarm.

Property	Description
Time	Current time.

4.3.3.1 GetAlarm

This method returns information about the alarm.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IClock

Syntax

```
[helpstring("Get alarm time method")]  
HRESULT GetAlarm([out] VARIANT_BOOL* activate, [out] DATE* pVal);
```

The **GetAlarm** method syntax has these parts:

Part	Description
<i>activate</i>	Output. A reference to a boolean value representing the status of the alarm (on or off).
<i>pVal</i>	Output. A reference to a date value containing the activation time of the alarm.

Remarks

The alarm can be set with the **SetAlarm** method. If the method fails, call the **GetLastError** function to get extended error information.

Example

See 4.3.3.3

4.3.3.2 SetAlarm

This method sets the alarm.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IClock

Syntax

```
[helpstring("Set alarm time method")]  
HRESULT SetAlarm([in] VARIANT_BOOL activate, [in] DATE newVal);
```

The **SetAlarm** method syntax has these parts:

Part	Description
<i>activate</i>	Input. A boolean value representing the status of the alarm (on or off).
<i>newVal</i>	Input. A date value containing the activation time of alarm. Only the time information is used, not the date information.

Remarks

The alarm cannot be deactivated using this method with Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones.

The *newVal* parameter has effect only if the alarm is to be activated. The phone's alarm settings can be obtained with the **GetAlarm** method (4.3.3.1). If the method fails, call the **GetLastError** function (4.2.1.3) to get extended error information.

Example

This method sets an alarm on the phone according to the controls on the form. First, it checks the **IsCapability** property (4.2.3.1) to determine if the phone is capable of alarming. Then, the alarm time is validated. If the time is valid, the alarm is either activated or deactivated, depending on the status of the check box.

```
Private Clock As STTNGS3A_SLib.IClock  
Private Capability As STTNGS3A_SLib.IPhoneCapability  
  
Private Sub SetAlarmButton_Click()  
    On Error GoTo ErrorTrap  
  
    Dim s As String  
  
    'Check that the phone can handle alarms  
    If Capability.IsCapability(CAPAB_ALARM) = 0 Then  
        MsgBox ("The phone does not support alarm.")  
        Exit Sub  
    End If
```



```
'If alarm is about to be set, check the time
If AlarmCheck = 1 Then
    If (HourEdit > 23) Or (HourEdit < 0) Or _
        (MinuteEdit > 59) Or (MinuteEdit < 0) Then
        MsgBox ("Invalid time of alarm")
        Exit Sub
    End If
End If

'Set the alarm according to the time edit boxes
s = HourEdit & ":" & MinuteEdit
If AlarmCheck = 1 Then
    Call Clock.SetAlarm(True, s)
Else
    Call Clock.SetAlarm(False, s)
End If

Exit Sub

ErrorTrap:
    'Inform the user about failure
    MsgBox "Error #" & Clock.GetLastError & " in setting alarm."
End Sub
```

4.3.3.3 Time (rw)

This property contains current time according to the phone's clock.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IClock

Type: **Date**

Syntax

```
[propget, helpstring("Time property")]
    HRESULT Time([out, retval] DATE *pVal);
[propput, helpstring("Time property")]
    HRESULT Time([in] DATE newVal);
```

Example

This method reads the time and the alarm status from the phone to the controls in the window. The alarm is read using the GetAlarm method.

```
Public PhoneControl As New STTNGS3A_SLib.PhoneControl_Suite3

Private Sub ReadClock()
    Dim Clock As STTNGS3A_SLib.IClock
    Dim d As Date, AlarmOn As Boolean, AlarmTime As Date

    On Error GoTo ErrorTrap

    Set Clock = PhoneControl

    'Read current time
    d = Clock.Time
    TimeLabel = Format(d, "hh:mm")
    DateLabel = Format(d, "dd.mm.yyyy")

    'Read alarm status
    Call Clock.GetAlarm(AlarmOn, AlarmTime)

    'Alarm is on, show alarm time on frame
    If AlarmOn = 1 Then
        AlarmCheck = 1
        HourEdit = Hour(AlarmTime)
        MinuteEdit = Minute(AlarmTime)
    End If

    Exit Sub

ErrorTrap:
    'Inform the user about failure
    MsgBox "Error #" & Clock.GetLastError & " in reading clock."
End Sub
```

4.3.4 IMediaPool Interface

This interface contains methods, which can be used when saving ringing tones.

Method	Description
CreateTone	Create empty Tone component (ITone).
SaveTone	Save ringing tone to phone.

4.3.4.1 CreateTone

This method creates an empty tone object.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IMediaPooool

Syntax

```
[helpstring("CreateTone method")]  
HRESULT CreateTone([out, retval] ITone ** ppITone);
```

The **CreateTone** method syntax has these parts:

Part	Description
ppITone	Output. A reference to new empty tone object (ITone)

Remarks

The empty tone object must be filled by using properties found in the ITone interface. After that, the object is saved by **SaveTone**.

4.3.4.2 SaveTone

This method saves the tone object in the phone.

Member of STTNGS3A_SLib.PhoneControl_Suite3.IMediaPooool

Syntax

```
[helpstring("SaveTone method")]  
    HRESULT SaveTone([in] ITone * pITone);
```

The **SaveTone** method syntax has these parts:

Part	Description
ppITone	Input. A reference to filled tone object (ITone).

4.4 TONE Component

This component contains the data of one ringing tone. Before it can be used, it must be filled by properties found from ITone.

Phone UI will inform the user about the received tone. The user will need to take further action using the phone UI (listen, save, delete the “received” tone). The tone name can and should be embedded in OTA data.

4.4.1 ITone Interface

Method	Description
SetByte	Set ringing tone byte in the specified location of tone.

Property	Description
Length	Sets internal buffer size. Must be set before SetByte is called for the first time. The length tells how many OTA-bytes are included in the new tone. See www.forum.nokia.com > Smart Messaging Specification for more details about ringing tones.

4.4.1.1 SetByte

This method sets the ringing tone byte in the specified location of tone.

Member of STTNGS3A_SLib.Tone.ITone

Syntax

```
[helpstring("method SetByte")]  
HRESULT SetByte([in] short nIndex, [in] short nByte);
```

The **SetByte** method syntax has these parts:

Part	Description
<i>nIndex</i>	Input. byte location in tone (starting at zero).
<i>nByte</i>	Input. OTA-byte value to set.

4.4.1.2 Length (rw)

This property contains amount of bytes in ringing tone.

Member of STTNGS3A_SLib.Tone.ITone Type: **Short**

Syntax

```
[propget, helpstring("property Length")]  
    HRESULT Length([out, retval] short *pVal);  
[propput, helpstring("property Length")]  
    HRESULT Length([in] short newVal);
```

Remarks

Before calling SetByte for the first time, set internal buffer size by calling this method.

4.5 Security_Suite3 Component

The phone's security settings can be accessed with this component. The component contains properties, methods and functions for changing the passwords, security level and password request statuses.

4.5.1 ISecurity Interface

Interface contains methods and functions for entering and changing passwords.

Method	Description
ChangePw	Changes the specified password.
GetPwStatus	Returns the password type the phone is requesting.
EnterPw	Enters the specified password in the phone.

4.5.1.1 ChangePw

This method allows the phone's passwords (PIN, PIN2 Security code) to be changed.

Member of STTNGS3A_SLib.Security_Suite3.ISecurity

Syntax

```
[helpstring("Change password method")]  
HRESULT ChangePw(  
    PASSWORD_TYPE type,  
    BSTR oldPw,  
    BSTR newPw  
);
```

The **ChangePw** method syntax has these parts:

Part	Description
<i>type</i>	Input. A value or constant of type PASSWORD_TYPE specifying the type of password to be changed.
<i>oldPw</i>	Input. An ASCII string containing the old password of the selected type.
<i>newPw</i>	Input. An ASCII string containing the new password of the selected type.

Remarks

The PIN code can be changed only if the PIN code request is activated. Otherwise, ChangePw method will return *errNotSupported* error. If the method fails, call the **GetLastError** function to get extended error information.

The maximum length of PIN codes varies on SIM cards. The length of the security code is 5 digits.

Example

This method changes the selected password (PIN, PIN2 or security code).

```
Private SecurityAdapter As STTNGS3A_SLib.Security_Suite3  
  
Private Sub ChangePassButton_Click()  
    On Error GoTo ErrorTrap  
  
    Dim pwtype As PASSWORD_TYPE  
  
    'Check that something is written in the password texts  
    If (Len(OldPassEdit) < 4) Or (Len(NewPassEdit) < 4) Then  
        MsgBox ("Invalid password(s)")  
        Exit Sub  
    End If
```

```
With SecurityAdapter
  Select Case PassTypeCombo.ListIndex
    Case -1 'Nothing selected
      MsgBox ("Password type not selected")
      Exit Sub
    Case 0 'PIN
      pwtype = SEC_PW_PIN
      Call .ChangePw(pwtype, OldPassEdit, NewPassEdit)
    Case 1 'PIN2
      pwtype = SEC_PW_PIN2
      Call .ChangePw(pwtype, OldPassEdit, NewPassEdit)
    Case 2 'Security code
      pwtype = SEC_PW_SECURITY_CODE
      Call .ChangePw(pwtype, OldPassEdit, NewPassEdit)
  End Select
End With

MsgBox ("Password changed.")

Exit Sub

ErrorTrap:
  'Inform user about the failure
  Dim err As NmpAdapterError
  err = SecurityAdapter.GetLastError
  Select Case err
    Case errNotSupported
      MsgBox ("Turn PIN code request on, then try again.")
    Case errWrongPassword
      MsgBox ("Invalid password.")
    Case Else
      MsgBox ("Error #" & err & " in changing password.")
  End Select
End Sub
```

4.5.1.2 GetPwStatus

This function returns the password type the phone is requesting.

Member of STTNGS3A_SLib.Security_Suite3.ISecurity

Syntax

```
[helpstring("Get password status method")]  
HRESULT GetPwStatus(  
    [out, retval] PASSWORD_TYPE* type  
);
```

The **GetPwStatus** method syntax has these parts:

Part	Description
<i>type</i>	A value of type PASSWORD_TYPE containing the password type.

Remarks

This function returns a password type other than SEC_PW_NONE only during the phone's startup procedure. For example, the PIN2 queries concerning the Advice of Charge cannot be detected with this function.

If the function fails, call the **GetLastError** function to get extended error information.

Example

This method checks if the phone is requesting any passwords during startup. If PIN code is requested, the user is prompted to enter it. The given password is sent to the phone using the **EnterPw** method.

```
Private Sub CheckDevice()  
    On Error GoTo ErrorTrap  
  
    Dim SecAdapter As STTNGS3A_SLib.Security_Suite3  
    Dim pwtype As PASSWORD_TYPE  
  
    Set SecAdapter = New STTNGS3A_SLib.Security_Suite3  
  
    With SecAdapter  
        Dim Pass As String, newpin As String  
  
        'Is the phone requesting passwords?  
        pwtype = .GetPwStatus  
  
        'PIN is wanted, query the code from the user  
        'and enter it to the phone.  
        If pwtype = SEC_PW_PIN Then  
            Pass = InputBox("Type PIN code:")  
            If Not Pass = "" Then  
                Call .EnterPw(SEC_PW_PIN, Pass, "")  
            End If  
        End If  
    End With  
  
ErrorTrap:  
End Sub
```

```
        End If
    End If
End With

SecAdapter.Terminate
Set SecAdapter = Nothing

Exit Sub

ErrorTrap:
    'Inform the user about failure
    Dim err As NmpAdapterError
    err = SecAdapter.GetLastError
    Select Case err
        Case errWrongPassword
            MsgBox ("Invalid password. Exiting...")
        End
        Case Else
            MsgBox ("Error #" & err & " in checking device.")
        End Select
    End Select
End Sub
```

4.5.1.3 EnterPw

This method enters a password (PIN, PIN2, PUK, PUK2, Security code) into the phone.

Member of STTNGS3A_SLib.Security_Suite3.ISecurity

Syntax

```
[helpstring("Enter password method")]  
HRESULT EnterPw(  
    [in] PASSWORD_TYPE type,  
    [in] BSTR pwCode,  
    [in] BSTR newPIN  
);
```

The **EnterPw** method syntax has these parts:

Part	Description
<i>type</i>	Input. A value or constant of type <code>PASSWORD_TYPE</code> specifying the type of password to be entered.
<i>pwCode</i>	Input. An ASCII string containing the password of the selected type.
<i>newPIN</i>	Input. An ASCII string containing the new PIN code when entering PUK or PIN2 code when entering PUK2 code.

Remarks

For this method to have an effect, the phone must be requesting a password. More importantly, the password request must be noticeable with the **GetPwStatus** method. This is usually the case only when the phone has been turned on.

When not entering PUK or PUK2 code, the value of *newPin* has no meaning. If the method fails, call the **GetLastError** function to get extended error information.

The maximum length of PIN codes varies on SIM cards. The length of the security code is 5 digits.

Example

See example of **GetPwStatus**

4.6 Network_Suite3 Component

This component contains interfaces for controlling the phone's network properties.

4.6.1 INetwork Interface

The Network_Suite3 interface contains properties, methods and functions for querying network status.

Method	Description
GetRegistrationStatus	Queries the network registration status.
NetworkSearchStart	Start searching sequence for available networks.
NetworkSelectionStart	Start selection of wanted operator.
StartListeningNetworkEvents	Informs the component object that the client is ready to receive outgoing network method calls from the object.

4.6.1.1 GetRegistrationStatus

This method queries the current status of network registration.

Member of STTNGS3A_SLib.Network_Suite3.INetwork **Syntax**

```
[helpstring("Get network registration method")]
HRESULT GetRegistrationStatus(NETWORK_REG_STATUS* state,
    NETWORK_SELECTION_MODE* sel, short* cellID,
    short* lac, long* operCode, BSTR* operName);
```

The **GetRegistrationStatus** method syntax has these parts:

Part	Description
<i>state</i>	Output. A reference to a value of type NETWORK_REG_STATUS containing the network registration status.
<i>sel</i>	Output. A reference to a value of type NETWORK_SELECTION_MODE containing the network selection mode.
<i>cellID</i>	Output. A reference to an integer containing the cell identification number.
<i>lac</i>	Output. A reference to an integer containing the location area code.
<i>opercode</i>	Output. A reference to a long value containing the operator code.
<i>opername</i>	Output. A reference to a string containing the operator name.

Remarks

If the method fails, call the **GetLastError** (4.2.1.3) function to get extended error information.

Example

This method queries the network registration status and displays the currently selected operator's name and code on the form.

```
Private NetworkAdapter As New STTNGS3A_SLib.Network_Suite3

Private Sub ReadNetwork()
    On Error GoTo ErrorTrap

    Dim st As NETWORK_REG_STATUS
    Dim sel As NETWORK_SELECTION_MODE
    Dim cell As Integer, lac As Integer
    Dim code As Long
    Dim name As String

    'Query the current network status
    With NetworkAdapter
```



```
        Call .GetRegistrationStatus(st, sel, cell, lac, code, name)
    End With

    'Save the information to the frame
    NameLabel = name
    CodeLabel = code

    Exit Sub

ErrorTrap:
    'Inform user about the failure
    Dim err As NmpAdapterError
    err = NetworkAdapter.GetLastError
    MsgBox ("Error #" & err & " in getting network status.")
End Sub
```

4.6.1.2 NetworkSearchStart

Starts sequence for search available networks. Results can be found by event **AvailableNetworksResponse**.

Member of STTNGS3A_SLib.Network_Suite3.INetwork

Syntax

```
[helpstring("Start network searching")]  
    HRESULT NetworkSearchStart();
```

4.6.1.3 NetworkSelectionStart

Starts sequence for select wanted network. Results can be found by event **NetworkSelectResponse**.

Member of STTNGS3A_SLib.Network_Suite3.INetwork **Syntax**

```
[helpstring("Start network selection")]  
HRESULT NetworkSelectionStart(long opercode,  
    NETWORK_SELECTION_MODE* selmode);
```

The **NetworkSelectionStart** method syntax has these parts:

Part	Description
<i>opercode</i>	Input. The code of wanted operator. Ignored if <i>selmode</i> is AUTOMATIC.
<i>selmode</i>	Input. Selection mode, MANUAL or AUTOMATIC.

4.6.1.4 StartListeningNetworkEvents

This method informs the component object that the client is ready to receive outgoing method calls (events) from the object concerning network status.

Member of STTNGS3A_SLib.Network_Suite3.INetwork

Syntax

```
[helpstring("method StartListeningNetworkEvents")]  
HRESULT StartListeningNetworkEvents();
```

Remarks

Call this method to start receiving SettingsAdapter Network notifications if some of the following changes happen: AvailableNetworksResponse, NetworkSelectResponse, RegistrationStatusChange, SignalStrengthChanges.

To be able to receive outgoing method calls, the client must implement and register an outgoing interface sink.

A client thread that has no message pump or a thread that may be blocked in some Win32 wait function is not allowed to declare itself an event sink by calling **StartListeningNetworkEvents**. Otherwise, the event calls queued by the component object become blocked and the component server is likely to deadlock.

4.6.2 INetworkNotify Interface

Contains event sources for client to listen to and to get information about several network related changes.

Event	Description
AvailableNetworksResponse	Occurs when the phone finds an available network during the network search.
NetworkSelectResponse	Occurs in response to a network selection request.
RegistrationStatusChange	Occurs when the network registration status changes, e.g., when the phone connects to a network.
SignalStrengthChanges	Occurs when the currently selected network's signal strength changes.

4.6.2.1 AvailableNetworksResponse

Occurs when the phone finds an available network during the network search.

Member of STTNGS3A_SLib.Network_Suite3.INetworkNotify

Syntax

```
[helpstring("Available Networks Response")]  
HRESULT AvailableNetworksResponse(  
    [in] long operCode, [in] NETWORK_REG_STATUS regStatus,  
    [in] VARIANT_BOOL bLast,[in] BSTR operName,[in] long lErrorCode);
```

The **AvailableNetworksResponse** event syntax has these parts:

Part	Description
<i>opercode</i>	A long value containing the code of the available operator.
<i>regStatus</i>	A value of type NETWORK_REG_STATUS containing the network registration status.
<i>bLast</i>	A boolean value indicating whether the event will be the last for this network search.
<i>lErrorCode</i>	A long value containing the possible error code.

Remarks

For this event to take place, the network search must be started using the **NetworkSearchStart** method. If suitable networks are found, the phone will fire one event for each of the found operators.

4.6.2.2 NetworkSelectResponse

Occurs in response to a network selection request.

Member of STTNGS3A_SLib.Network_Suite3.INetworkNotify

Syntax

```
[helpstring("Network Select Response")]  
HRESULT NetworkSelectResponse([in] NETWORK_REG_STATUS regStatus,  
[in] NETWORK_SELECTION_MODE selSMode, [in] short cellID,  
[in] short locatCode,[in] long operCode,[in] BSTR operoName,[in] long lErrorCode);
```

The **NetworkSelectResponse** event syntax has these parts:

Part	Description
<i>regStatus</i>	A value of type NETWORK_REG_STATUS containing the current network registration status.
<i>selSMode</i>	A value of type NETWORK_SELECTION_MODE containing the current network selection mode.
<i>cellID</i>	An integer containing the cell identifier.
<i>locatCode</i>	An integer containing the location area code.
<i>opercode</i>	A long value containing the code of the selected operator.
<i>operoName</i>	An ASCII string containing the name of the selected operator.
<i>lErrorCode</i>	A long value receiving the possible error code.

Remarks

For this event to take place the network selection must be activated with the **NetworkSelectionStart** method.

4.6.2.3 RegistrationStatusChange

Occurs when the network registration status changes, e.g., when the phone connects to a network.

Member of STTNGS3A_SLib.Network_Suite3.INetworkNotify

Syntax

```
[helpstring("Registration Status Change")]  
HRESULT RegistrationStatusChange(  
[in] NETWORK_REG_STATUS regStatus, [in] NETWORK_SELECTION_MODE  
selSMODE,[in] short cellID,[in] short locatCode,[in] long operCode,[in] BSTR operoName);
```

The **RegistrationStatusChange** event syntax has these parts:

Part	Description
<i>regStatus</i>	A value of type NETWORK_REG_STATUS containing the network registration status.
<i>selSMODE</i>	A value of type NETWORK_SELECTION_MODE containing the network selection mode.
<i>cellID</i>	An integer value containing the cell identifier.
<i>locatCode</i>	An integer value containing the location area code.
<i>operCode</i>	A long value containing the currently selected operator's code.
<i>operoName</i>	An ASCII string containing the name of the currently selected operator.

Example

This event handler gathers currently selected operator's name and code and writes them on the form.

```
Private WithEvents NetworkAdapter As STTNGS3A_SLib.Network_Suite3  
  
Private Sub NetworkAdapter_RegistrationStatusChange(ByVal regStatus As  
STTNGS3A_SLib.NETWORK_REG_STATUS, ByVal selSMODE As  
STTNGS3A_SLib.NETWORK_SELECTION_MODE, ByVal cellID As Integer, _  
ByVal locatCode As Integer, ByVal operCode As Long, _  
ByVal operoName As String)  
  
    'Save information to the frame  
    NameLabel.Caption = operoName  
    CodeLabel = operCode  
  
End Sub
```


4.6.2.4 SignalStrengthChanges

Occurs when the currently selected network's signal strength changes.

Member of STTNGS3A_SLib.Network_Suite3.INetworkNotify

Syntax

```
[helpstring("Signal Strength Changes")]  
    HRESULT SignalStrengthChanges([in] long nSignal);
```

The **SignalStrenthChanges** event syntax has these parts:

Part	Description
<i>nSignal</i>	A long value containing the new signal strength.

4.7 Enumerated DATA Types

Type	Description
CapabilityOpt	Contains the capabilities a phone may have.
DevNotifyOpt	Contains the possible states of the phone device.
EnchBoolean	Contains the three states some properties may have.
MemTypeOpt	Contains the selectable memory types.
NETWORK_REG_STATUS	Contains the network registration statuses.
NETWORK_SELECTION_MODE	Contains the network selection modes.
NmpAdapterError	Contains Nokia PC Connectivity SDK's error codes.
PASSWORD_TYPE	Contains the password types.
PHONE_STATUS	Contains the possible statuses of the phone.

4.7.1 CapabilityOpt

Contains the capabilities a phone may have.

Member of STTNGS3A_SLib

The following table summarizes the capabilities.

Type	Description
CAPAB_ALARM	Alarm.
CAPAB_ALTERNATING_CALL	Alternating calls (combined data/fax ⇔ voice)
CAPAB_ANALOG_DATA_CALL	Analog data calls (not used in GSM).
CAPAB_AOC	Advice of Charge.

CAPAB_CALENDAR	Calendar.
CAPAB_CALL_SS	Call independent supplementary services (call forwarding, call barring, etc).
CAPAB_CB_MSG_ROUTING	Cell broadcast message routing.
CAPAB_CIRCUIT_SWITCH_CALL	Circuit switched data/fax calls.
CAPAB_CLOCK	Clock.
CAPAB_DIG_AUDIO	Digital audio.
CAPAB_DTMF_RECEIV	DTMF receiving (not used in GSM yet).
CAPAB_DTMF_SEND	DTMF sending.
CAPAB_FAST_DATA	14.4 kbps data calls.
CAPAB_FDNSIM	Fixed dialing supported by SIM.
CAPAB_HALF_RATE_DATA	Half rate data.
CAPAB_IN_CALL_SS	In-call supplementary services (e.g., hold, multi-party, etc.). See GSM 04.08.
CAPAB_KEYBOARD_EVENT	Keyboard events.
CAPAB_KEYBOARD_SIM	Keyboard simulation (not supported by Nokia Data Suite 3.0 adapter interfaces).
CAPAB_NETWORK_SEL	Network selection (not supported by Nokia Data Suite 3.0 adapter interfaces).
CAPAB_PACK_DATA_CALL	Packet data calls.
CAPAB_PB_UPDATE	Phonebook entries reading/updating.
CAPAB_REG_STATUS_EVENTS	Registration status events (when network status changes, does not include signal strength).
CAPAB_SCM_SPEED_UPDAT	Phonebook speed dial list reading/updating.
CAPAB_SECURITY	Modifiable security settings.
CAPAB_SHOW_INDIGATOR_EVENT	Display indicator events (not supported by Nokia Data Suite 3.0 adapter interfaces). Only indicator is Incoming SMS from network, when SMS memories are full.

CAPAB_SHOW_TEXT_EVENT	Display text events (not supported by Nokia Data Suite 3.0 adapter interfaces).
CAPAB_SIM_ACCESS	Reading from and writing to a SIM card.
CAPAB_SIM	See remarks.
CAPAB_SIMLSB	See remarks.
CAPAB_SIMMSB	See remarks.
CAPAB_SM_ROUTING	MT Short message (SMS) routing.
CAPAB_SM_SENDING	MO Short message (SMS) sending.
CAPAB_SMT_MSG_SERVER	Smart messaging (NBS) server. Support for messages specified in Narrow Band Socket specification.
CAPAB_UI_SETTING	Access to UI settings.

Remarks

The CAPAB_SIM, CAPAB_SIMLSB and CAPAB_SIMMSB types together indicate the phase of the SIM. The phase can be determined by reading the **IsCapability** property with each of the SIM capability types and gathering the results into a 3-bit vector. The return value of the IsCapability property must be converted to a bit value (TRUE = 1, FALSE = 0).

All the valid 3-bit combinations are listed in the table below:

CAPAB_SIMMSB	CAPAB_SIM	CAPAB_SIMLSB	Meaning
0	0	0	No SIM / SIM not ready
0	0	1	Phase 1
0	1	0	Phase 2
0	1	1	Phase 2+
1	0	0	Phase 4

Note: The capabilities CAPAB_SIM, CAPAB_SIMLSB and CAPAB_SIMMSB can be read with Nokia PC Connectivity SDK interfaces, but the values will be undetermined.

4.7.2 DevNotifyOpt

Contains the possible states of the phone device.

Member of STTNGS3A_SLib The following table summarizes the device states:

Type	Description
ATTACHED	Device is attached to the PC.
DISCONNECTED	Device is attached to the PC, but is operating in AT-mode (not supported by Nokia 51xx/61xx phones).
REMOVED	Device is removed from the PC.
UNKNOWN	Device status is unknown.

4.7.3 EnchBoolean

Contains the three states some properties may have.

Member of STTNGS3A_SLib

The following table summarizes the states:

Type	Description
VAL_OFF	Off.
VAL_ON	On.
VAL_PRESET	Preset value.

4.7.4 MemTypeOpt

Contains the selectable memory types.

Member of STTNGS3A_SLib The following table summarizes the memory types:

Type	Description
FDN	Fixed Dialing Numbers (FDN) memory.
PHONE	The memory of the phone.
SIM	SIM memory.

4.7.5 NETWORK_REG_STATUS

Contains the network registration statuses.

Member of STTNGS3A_SLib

The following table summarizes the statuses:

Type	Description
NET_RS_HOME	Device is registered to home network.
NET_RS_NO_SERV_NOTSEARCHING	Device is not in service and is not searching for network.
NET_RS_NO_SERV_SEARCHING	Device is not in service, but is searching for available networks.
NET_RS_ROAM	Device is registered to a roamed network.

4.7.6 NETWORK_SELECTION_MODE

Contains the network selection modes.

Member of STTNGS3A_SLib The following table summarizes the modes:

Type	Description
NET_SM_AUTOMATIC	Automatic network selection.
NET_SM_MANUAL	Manual network selection.
NET_SM_UNKNOWN	Network selection mode is unknown.

4.7.7 NmpAdapterError

Contains Nokia PC Connectivity SDK's error codes.

Member of STTNGS3A_SLib

The following table summarizes the error codes:

Error	Code	Description
errNoError	0	Method call completed successfully.
errCalendarNotSupported	5377	Phone device has no calendar.
errCalendarUnknownNoteType	5378	Calendar note type is unknown.
errCalendarUnknownItemType	5379	Calendar item type is unknown.
errCalendarComponentCreation	5380	Calendar data component creation failed.
errCalendarItemRead	5381	Calendar item read failed.
errCalendarItemWrite	5382	Calendar item write failed.
errCalendarItemDelete	5383	Calendar item delete failed.
errCalendarCallEmpty	5384	Call type item has empty fields.
errCallNoActiveCall	5633	No active call.
errCallNoDualModeCall	5634	No dual-mode call.
errCallAlreadyActive	5635	Call is already active.
errCallSignallingFailure	5636	Signaling failure.
errCallInvalidMode	5637	Invalid call mode.
errCbSettingSetFailed	5889	Cell broadcast settings set general failure.
errCbInvalidLanguage	5890	Invalid language code.
errCbInvalidTopic	5891	Invalid topic code.
errCbTooManyLang	5892	Too many languages specified.
errCbTooManyTopic	5893	Too many topics specified.
errPnInvalidMemory	6145	Specified memory type is not accessible.
errPnNumberTooLong	6146	Too many characters in contact number.
errPnNameTooLong	6147	Too many characters in contact name.
errPnInvalidCharacters	6148	Name or number contains illegal characters.
errPnMemoryFull	6149	Memory is full, no new items can be added.
errPnNotAvail	6150	Contacts cannot be modified.
errPnTimestampNotavail	6151	Time stamp does not exist.
errPnCallergroupsNotsupported	6152	Caller groups are not supported by the phone device.
errPnInvalidIconFormat	6153	Icon is of unsupported format.

errPnEntryLocked	6154	Specified entry is locked.
errPnSpeedkeyNotassigned	6155	Specified key not assigned as speed key.
errPnEmpty	6156	Contact is empty.
errDataNotAvail	6401	Requested data is not available.
errSsUnknownSubscriber	6402	Unknown subscriber.
errSsBearerServNotProvision	6403	Bearer service not provisioned.
errSsTeleServNotProvision	6404	Tele service not provisioned.
errSsCUGReject	6405	Closed user group rejected.
errSsIllegalSsOperation	6406	Illegal supplementary services operation.
errSsErrorStatus	6407	Operation not legal with current status.
errSsNotAvailable	6408	Supplementary service not available.
errSsSubscriptionViolation	6409	Subscriber violation.
errSsIncompatibility	6410	Incompatibility error.
errSsSpecificError	6411	Supplementary service specific error.
ErrSsSystemFailure	6412	System failure.
ErrSsDataMissing	6413	Data missing.
ErrSsUnexpectedDataValue	6414	Unexpected data value.
ErrSsPasswordRegisFailure	6415	Password registration failed.
ErrSsNegativePasswordCheck	6416	Wrong supplementary service password.
ErrSsFacilityNotSupported	6417	Supplementary service facility is not supported.
ErrSsResourcesNotAvailable	6418	Supplementary service resources are not available.
ErrSsMaxnumOfMptyPartExceed	6419	The maximum number of parties in a multi-party call has been exceeded.
ErrSsCallBarred	6420	The active call is barred.
ErrSsMaxnumOfPwAttViolation	6421	Maximum number of wrong password attempts.
ErrSsAbsentSubscriber	6422	Subscriber is absent.
ErrSsUSSDBusy	6423	USSD service is busy.
ErrSsUnknownAlphabet	6424	Unknown alphabet.
ErrWrongPassword	6425	Wrong password.
ErrPasswordNotRequired	6426	Password is not required.
ErrUpdateImpossible	6427	Update is impossible.
ErrNetCallActive	6428	Call is active.
ErrNetNoMsgToQuit	6429	No message to quit.
ErrNetUnableToQuit	6430	Unable to quit.
ErrNetAccessDenied	6431	Access to network is denied.
ErrSsMMError	6432	MM error.
ErrSsMsgError	6433	Messaging error.
ErrSsMMRelease	6434	MM release.
ErrSsActivationDataLost	6435	Data to be activated is not

		found from network.
ErrSsServiceBusy	6436	Service tried to be activated is busy.
ErrSsDataError	6437	The data passed is invalid.
ErrSsTimerExpired	6438	Network timeout.
ErrSsPWEErrorEnterPassword	6439	Password must be entered with command.
ErrSsPWEErrorEnterNewPassword	6440	New password must be entered.
ErrSsPWEErrorEnterNewPasswordAgain	6441	New password must be entered again.
ErrSsPWEErrorBadPassword	6442	Password entered cannot be accepted.
ErrSsPWEErrorBadPasswordFormat	6443	The format of the entered password is invalid.
ErrSsPReturnErrorProblem	6444	Problems with password in supplementary services.
ErrSsPUnrecognizedComp	6445	Unrecognized component.
ErrSsPMistypedComp	6446	Mistyped component.
ErrSsPBadlyStructuredComp	6447	Badly structured component.
ErrSsPDuplicateInvokeID	6448	Duplicate invoke ID.
ErrSsPUnrecognizedOperation	6449	Unrecognized operation.
ErrSsPMistypedInvParameter	6450	Mistyped parameter.
ErrSsPResourceLimitation	6451	Resource limitation.
ErrSsPInitatingRelease	6452	Initiating release.
ErrSsPUnrecognizedLinkedID	6453	Unrecognized linked ID.
ErrSsPLinkedRespUnexpected	6454	Linked response unexpected.
ErrSsPUnexpectedLinkedOper	6455	Unexpected linked operation.
ErrSsPUnrecognizedInvokeID	6456	Unrecognized invoke ID.
ErrSsPReturnResultUnexpected	6457	Return result unexpected.
ErrSsPMistypedResParameter	6458	Mistyped parameter.
ErrSsPReturnErrorUnexpected	6459	Unexpected returned error.
ErrSsPUnrecognizedError	6460	Unrecognized error.
ErrSsPUnexpectedError	6461	Unexpected error.
ErrSsPMistypedErrParameter	6462	Mistyped error parameter.
ErrSmsCreateFailed	6657	Short Message component creation failed.
ErrSmsCannotSendMTMessages	6658	SMS Deliver or Status Report types are not allowed.
ErrSmsInvalidType	6659	Invalid SMS type.
ErrSmsInvalidDataCodingScheme	6660	Invalid data coding scheme.
ErrSmsInvalidUserDataLength	6661	User data length is too big.
ErrSmsInvalidUserDataFormat	6662	Invalid user data format.
errSmsInvalidUserData	6663	User data was not legally formatted.
ErrSmsInvalidUserDataHeaderLength	6664	Invalid user data header length is too big.
ErrSmsInvalidSCTimeStamp	6665	Time stamp set by SMS Service Center is invalid.
ErrSmsTooLongSCAddress	6666	SMS Service Center number is too long.

ErrSmsInvalidValidityPeriod	6667	Invalid validity period.
ErrSmsTooLongOtherEndAddress	6668	Phone number is too long.
ErrSmsInvalidParameterSetIndex	6669	Given index is out of range.
ErrSmsTypeMOUndefined	6670	SMS MO Undefined type is not supported.
ErrSmsTypeMTUndefined	6671	SMS MT Undefined type is not supported.
ErrSmsTypeCommand	6672	SMS Command type is not supported.
ErrSmsNoSCAddress	6673	SMS Service Center number missing.
ErrSmsDefaultSetNameUsed	6674	Empty SMS set name was forwarded.
errProtocolError	6675	e.g., SC Number missing.
ErrDbXXXX	6913	MPDB error.
ErrInvalidLocation	7681	Specified location index is out of range.
ErrInvalidParameter	7682	One of the parameters is invalid.
ErrReserved	7683	Requested service is reserved.
ErrMemoryFull	7684	Memory operation cannot be done.
ErrEmptyLocation	7685	Memory location was empty.
ErrInvalidNumber	7686	Invalid number.
errCallCostLimitReached	7687	Call cost limit has been reached.
ErrRLP	7688	Radio Link Protocol (RLP) error.
ErrCommunicationError	7689	Communication error between the device and the PC.
ErrNotSupported	7690	Requested operation is not supported.
ErrMpApiNotAvail	7691	No connection to the Mobile Phone API (MPAPI).
ErrDeviceFailure	7692	Failure in device.
ErrNoSim	7693	No SIM card in phone.
ErrTerminalNotReady	7694	Device is not ready to handle message.
ErrSignallingFailure	7695	Signaling failure with the network.
ErrPhoneNotConnected	7696	Phone is not connected.
ErrPinRequired	7697	PIN code required.
ErrPukRequired	7698	PUK code required.
ErrPin2Required	7699	PIN2 code required.
ErrPuk2Required	7700	PUK2 code required.
ErrSecurityCodeRequired	7701	Security code required.
ErrBarred	7702	The operation was barred by the operator.
ErrSIMRejected	7703	SIM rejected.
ErrUnknown	7935	Unknown error.

4.7.8 MediaType

Contains known media types.

Member of STTNGS3A_SLib

The following table summarizes the media types:

Media type	Description
MEDIA_UNKNOWN	Unknown media
MEDIA_DRIVER	FBUS connection using Drivers DIOC interface
MEDIA_RS232	FBUS communication using COM port directly
MEDIA_IRDA	Infra Red connection

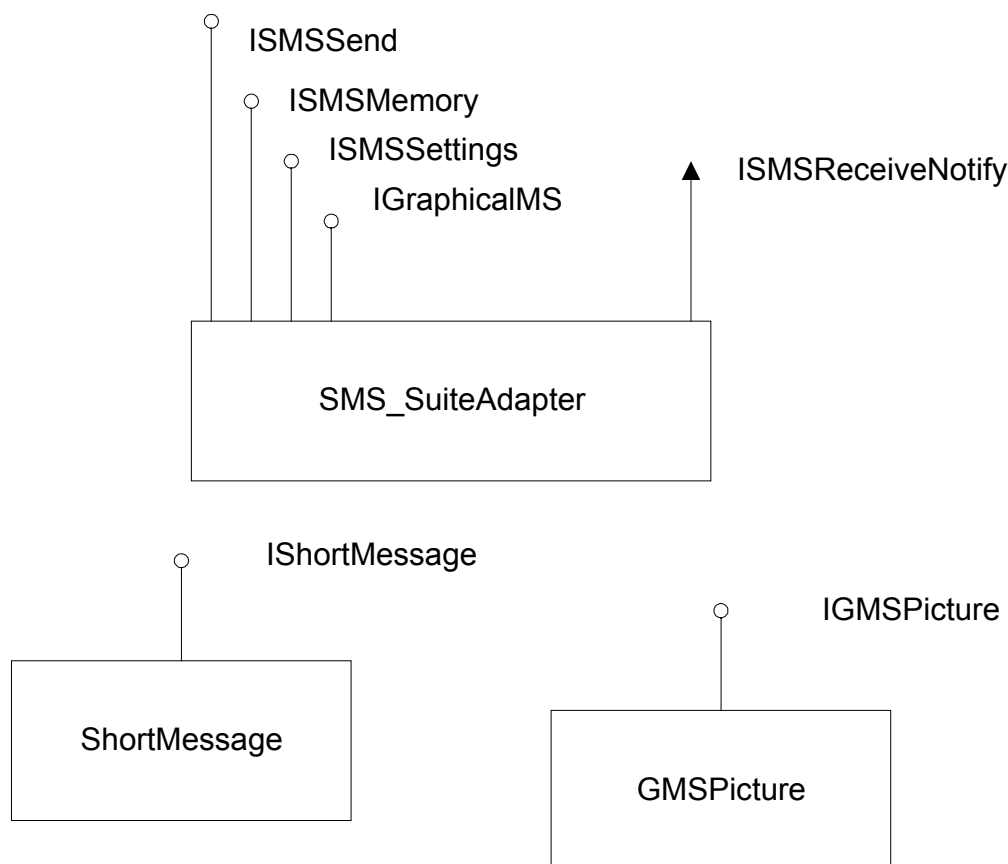
5. SHORT MESSAGE LIBRARY (SMS3ASUITELIB)

5.1 Overview

The Short Message Library contains components for handling GSM short messages (SMSs). Components contain interfaces for sending messages, adjusting SMS settings on the phone, handling SMS memory on the phone, as well as receiving SMS related notifications from the phone.

5.2 Object Model

Picture below represents Sms3aS.dll module. Module contains implementation for three objects, SMS_SuiteAdapter, ShortMessage and GMSPicture. SMS_SuiteAdapter offers methods for handling message traffic with the phone. ShortMessage and GMSPicture are pure data components for short messages and graphical messages.



Picture 2. Components and interfaces of SMS3AsuiteLib.

5.3 SMS_SuiteAdapter Component

This component contains methods, functions, and events to send and receive short messages, handle SMS memory, and set short message settings on the phone.

5.3.1 ISMSSend Interface

The ISMSSend interface contains methods and functions for sending a short message to the GSM network. It contains events used to notify the client when an SMS related event takes place on the phone.

Method	Description
CreateShortMsg	Creates and returns a new short message object.
GetLastError	Returns the code of the latest error.
Send	Sends a short message into GSM network.
StartListeningEvents	Informs the object that the client is ready to receive outgoing method calls from the object.

5.3.1.1 CreateShortMsg

This method creates a new short message object.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSend,
SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method CreateShortMsg")]  
HRESULT CreateShortMsg(  
    [out, retval] IShortMessage **ppSMS  
);
```

The **CreateShortMsg** method syntax has these parts:

Part	Description
<i>ppSMS</i>	A short message object of type ShortMessage.

Remarks

CreateShortMsg method returns a new short message object. If the function fails, call the **GetLastError** (5.3.1.2) method to get extended error information.

The object is initially filled with default parameters.

The created short message object can be, e.g., sent to the GSM network with the **Send** (5.3.1.3) method or stored in the phone's SMS memory with the **Store** (5.3.3.12) method.

Example

This example uses the **CreateShortMsg** function to create a test message to be sent to the destination address immediately when the form loads. The short message is sent using the **Send** (5.3.1.3) method.

```
Private SMSSend As SMS3ASuiteLib.SMS_SuiteAdapter  
  
Private Sub Form_Load()  
  
    Dim ShortMsg As SMS3ASuiteLib.ShortMessage  
  
    'enable error-handling routine  
    On Error GoTo ErrorHandler  
  
    'assign an object reference to a COM class  
    Set SMSSend = New SMS3ASuiteLib.SMS_SuiteAdapter  
  
    'create the message  
    Set ShortMsg = SMSSend.CreateShortMsg
```

```
'message body
ShortMsg.UserDataText = "Just testing my new Nokia Suite"

'destination number
ShortMsg.OtherEndAddress = "+358201234567"

'short message center number
ShortMsg.SCAddress = "+358207654321"

'send the message
Call SMSSend.Send(ShortMsg)

'avoid error handler
Exit Sub

'error handling routine
ErrorHandler:

    Dim SMSError As NmpAdapterError
    SMSError = SMSSend.GetLastError

    Select Case SMSError
        Case errPhoneNotConnected
            MsgBox "Error: The phone is not connected. "
        Case Else
            'handle other situations here
    End Select

End Sub

Private Sub Form_Unload(Cancel As Integer)

    'release the reference
    SMSSend.Terminate
    Set SMSSend = Nothing

End Sub
```


5.3.1.2 GetLastError

This method returns the Component Library's latest error code value.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSend

Syntax

```
[helpstring("method GetLastError")]  
HRESULT GetLastError(  
    [out, retval] NmpAdapterError *pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	Error code variable of type NmpAdapterError.

Remarks

The error code is maintained on a per library basis. For detailed information on error codes, see description of type NmpAdapterError (4.7.7).

5.3.1.3 Send

This method sends a short message to the GSM network.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSend **Syntax**

```
[helpstring("method Send")]  
HRESULT Send(  
    [in] IShortMessage *pSMS  
);
```

The **Send** method syntax has these parts:

Part	Description
<i>pSMS</i>	Input. Object reference to the short message to be sent.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The short message object must be created before sending using the **CreateShortMsg** (5.3.1.1) function.

To send graphical messages (GMS), see IGraphicalMS::SaveGMS

Example

See **CreateShortMsg** example.

5.3.1.4 StartListeningEvents

This method informs the component object that the client is ready to receive outgoing method calls from the object.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSend,
SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory,
SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSettings

Syntax

```
[helpstring("method StartListeningEvents")]  
    HRESULT StartListeningEvents(  
    );
```

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) method to get extended error information.

To be able to receive outgoing method calls, the client must implement and register an outgoing interface sink.

A client thread that has no message pump or a thread that may be blocked in a Win32 wait function is not allowed to declare itself an event sink by calling **StartListeningEvents** (5.3.1.4). Otherwise, the event calls queued by the component object become blocked and the component server is likely to deadlock.

Example

In the following example, the application indicates that it is ready to receive events by placing a **StartListeningEvents** (5.3.1.4) call to the component.

```
Public puSMS_SuiteAdapter As SMS3ASuiteLib.SMS_SuiteAdapter  
  
Private Sub Form_Load()  
    Set puSMS_SuiteAdapter = New SMS3ASuiteLib.SMS_SuiteAdapter  
    Call puSMS_SuiteAdapter.StartListeningEvents  
End Sub
```

5.3.2 ISMSReceiveNotify

This interface belongs to the SMSAdapter component. The software components that are registered with this interface are able to receive the notifications that this interface offers.

Event	Description
ShortMsgReceived	A new short message has been received.
ShortMsgSent	The short message has been sent to the GSM network.
SMSMemoryFull	SMS memory is full.

5.3.2.1 ShortMsgReceived

Occurs after a new short message has been received.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSReceiveNotify

Syntax

```
[helpstring("method ShortMsgReceived")]  
HRESULT ShortMsgReceived(  
    [in] SMS_MEMORY_LOCATION SMSMemory,  
    [in] long index,  
    [in] IShortMessage *pSMS  
);
```

The **ShortMsgReceived** event syntax has these parts:

Part	Description
<i>SMSMemory</i>	The identifier of SMS memory of type SMS_MEMORY_LOCATION. Tells which SMS memory the received message has been stored in.
<i>index</i>	The SMS memory index in which the short message has been stored.
<i>pSMS</i>	Reference to the received short message object of type ShortMessage.

Remarks

Short message related events are not sent before the **StartListeningEvents** (5.3.1.4) method has been called.

Example

This example uses **ShortMsgReceived** event to display a message box when a new short message is received. The phone number of the sender is shown in the message box.

```
Private WithEvents SMSNotify As SMS3ASuiteLib.SMS_SuiteAdapter  
  
Private Sub SMSNotify_ShortMsgReceived(  
    ByVal SMSMemory As SMS3ASuiteLib.SMS_MEMORY_LOCATION, _  
    ByVal index As Long, _  
    ByVal pSMS As SMS3ASuiteLib.ShortMessage)  
    MsgBox "Message from " & pSMS.OtherEndAddress  
End Sub
```

5.3.2.2 ShortMsgSent

Occurs when the short message has been sent to the GSM network either using the **Send** (5.3.1.3) method or the phone user interface.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSReceiveNotify

Syntax

```
[helpstring("method ShortMsgSent")]  
HRESULT ShortMsgSent(  
    [in] IShortMessage *pSMS
```

The **ShortMsgSent** event syntax has these parts:

Part	Description
<i>pSMS</i>	Reference to the sent short message object.

Remarks

Short message related events are not sent before the **StartListeningEvents** (5.3.1.4) method has been called.

Example

This example uses **ShortMsgSent** event to display a message box after the short message has been sent to the GSM network. The message box displays the short message destination number.

```
Private WithEvents SMSNotify As SMS3ASuiteLib.SMS_SuiteAdapter  
  
Private Sub SMSNotify_ShortMsgSent( _  
    ByVal pSMS As SMS3ASuiteLib.ShortMessage)  
  
    MsgBox "Short message sent to " & pSMS.OtherEndAddress  
End Sub
```

5.3.2.3 SMSMemoryFull

Occurs when the SMS memory (either phone memory or SIM memory) is full.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSReceiveNotify

Syntax

```
[helpstring("method SMSMemoryFull")]  
HRESULT SMSMemoryFull(  
    [in] SMS_MEMORY_LOCATION SMSMemory  
);
```

The **SMSMemoryFull** event syntax has these parts:

Part	Description
<i>SMSMemory</i>	The identifier of SMS memory of type SMS_MEMORY_LOCATION. Tells which SMS memory is full.

Remarks

Short message related events are not sent before the **StartListeningEvents** (5.3.1.4) method has been called.

Example

This example uses **SMSMemoryFull** event to display a message box when either phone or SIM SMS memory is full.

```
Private WithEvents SMSNotify As SMS3ASuiteLib.SMS_SuiteAdapter  
  
Private Sub SMSMemNotify_SMSMemoryFull( _  
    ByVal SMSMemory As SMS3ASuiteLib.SMS_MEMORY_LOCATION)  
  
    'check which memory is full  
    Select Case SMSMemory  
        Case PHONE_MEMORY  
            MsgBox "Phone SMS memory is full."  
        Case SIM_MEMORY  
            MsgBox "SIM SMS memory is full."  
        Case DEFAULT_MEMORY  
            MsgBox "Default SMS memory is full."  
    End Select  
End Sub
```

5.3.3 ISMSMemory Interface

The ISMSMemory interface contains methods and functions for handling SMS memory. It has methods for reading, writing, and deleting short messages as well as for querying SMS memory status, capacity, and the amount of messages stored in the SMS memory.

Method	Description
Delete	Deletes the given short message from the SMS memory.
GetCapacityInME	Returns the total number of short message memory positions in phone memory.
GetCapacityInSIM	Returns the total number of short message memory positions in SIM memory.
GetCapacityOfParamSets	Returns the number of SMS parameter sets.
GetMemoryConfiguration	Returns the configuration of the SMS memory.
GetNumberOfMessagesInME	Returns the total number of short messages stored in the phone's memory.
GetNumberOfMessagesInSIM	Returns the total number of short messages stored in the SIM memory.
GetNumberOfUnreadInME	Returns the number of unread short messages stored in the phone's memory.
GetNumberOfUnreadInSIM	Returns the number of unread short messages stored in the SIM memory.
GetSMSMemoryStatus	Returns all above mentioned memory information with a single method call.
Read	Reads the given short message from the SMS memory.
StartListeningEvents	Informs the object that the client is ready to receive outgoing method calls from the object.
Store	Stores a short message in the SMS memory.
CreateShortMsg	Creates and returns a new short message instance.
GetLastError	Returns the code number of the latest error.

5.3.3.1 Delete

This method deletes the given short message that is stored in the SMS memory.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method Delete")]
HRESULT Delete(
    [in] SMS_MEMORY_LOCATION SMSMemory,
    [in] long SMSIndex
);
```

The **Delete** method syntax has these parts:

Part	Description
<i>SMSMemory</i>	Input. Value or constant of type. SMS_MEMORY_LOCATION indicating the SMS memory type where the short message will be deleted from.
<i>SMSIndex</i>	Input. A long integer expression that evaluates the short message index in the SMS memory.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

Example

```
Private SMSMemory As SMS3ASuiteLib.ISMSMemory

Private Sub DeleteMessage(
    SMSMemType As SMS_MEMORY_LOCATION,
    MemoryIndex As Long)

    Dim Question As String
    Question = "Are you sure to delete the short message from the"

    If SMSMemType = PHONE_MEMORY Then
        Question = Question & " phone memory"
    ElseIf SMSMemType = SIM_MEMORY Then
        Question = Question & " SIM memory"
    Else
        Question = Question & " default memory"
    End If

    Question = Question & " location " & MemoryIndex & "?"

    Dim Response
    Response = MsgBox(Question, vbYesNo)
```

```
    If Response = vbNo Then
        Exit Sub
    End If

    On Error GoTo ErrorTrap

    'delete the message
    Call SMSMemory.Delete(SMSMemType, MemoryIndex)

    Exit Sub

ErrorTrap:

    'inform the user about failure
    MsgBox "Error " & SMSMemory.GetLastError & " in deleting the short mes-
sage."

End Sub
```

5.3.3.2 GetCapacityInME

This method tells the total number of memory locations in the phone's SMS memory.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method GetCapacityInME")]  
HRESULT GetCapacityInME(  
    [out] long *pMESMSMemorySize  
);
```

The **GetCapacityInME** method syntax has these parts:

Part	Description
<i>pMESMSMemorySize</i>	Output. Reference to the long integer variable in which the number of memory positions will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The analogous method for the SIM SMS memory is the **GetCapacityInSim** (5.3.3.3) method.

See also the **GetSMSMemoryStatus** (5.3.3.10) method.

Example

The example method below uses the **GetCapacityInMe** method and the **GetCapacityInSim** method to query memory capacities of the phone.

The **GetNumberOfMessagesInMe** method and the **GetNumberOfMessagesInSIM** method are used to query the total number of short messages stored in the memory.

The **GetNumberOfUnreadInME** (5.3.3.8) method and the **GetNumberOfUnreadInSIM** (5.3.3.9) method are used to query the number of unread messages.

The values of all the above mentioned properties are displayed in corresponding text label controls.

```
Private SMSMemory As SMS3ASuiteLib.ISMSMemory  
  
Private Sub MemoryCapacity()  
    Dim Number As Long  
  
    On Error GoTo ErrorTrap
```

```
'the total memory capacity
Call SMSMemory.GetCapacityInME(Number)
MECapacity.Caption = Number

Call SMSMemory.GetCapacityInSIM(Number)
SIMCapacity.Caption = Number

'the total number of messages
Call SMSMemory.GetNumberOfMessagesInME(Number)
MessagesInME.Caption = Number

Call SMSMemory.GetNumberOfMessagesInSIM(Number)
MessagesInSIM.Caption = Number

'the number of unread messages
Call SMSMemory.GetNumberOfUnreadInME(Number)
UnreadInME.Caption = Number

SMSMemory.GetNumberOfUnreadInSIM(Number)
UnreadInSIM.Caption = Number

Exit Sub

ErrorTrap:

    'inform the user about failure
    MsgBox "Error " & SMSMemory.GetLastError & " in querying memory capacity."

End Sub
```

5.3.3.3 GetCapacityInSIM

This method tells the total number of memory locations in the SIM card's SMS memory.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method GetCapacityInSIM")]  
HRESULT GetCapacityInSIM(  
    [out] long *pSIMSMSMemorySize  
);
```

The **GetCapacityInSIM** method syntax has these parts:

Part	Description
<i>pSIMSMS-MemorySize</i>	Output. Reference to the long integer variable in which the number of memory positions will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The analogous method for the phone SMS memory is the **GetCapacityInMe** (5.3.3.2) method.

See also the **GetSMSMemoryStatus** (5.3.3.10) method.

Example

See **GetCapacityInMe** (5.3.3.2)

5.3.3.4 GetCapacityOfParamSets

This method retrieves the number of SMS parameter sets.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory **Syntax**

```
[helpstring("method GetCapacityOfParamSets")]  
HRESULT GetCapacityOfParamSets(  
    [out] long *pAmountOfParamSets  
);
```

The **GetCapacityOfParamSets** method syntax has these parts:

Part	Description
<i>pAmountOfParamSets</i>	Output. Reference to the long integer variable in which the number of parameter sets will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The number of SMS parameter sets depends on the SIM card type.

Example

This example function calls the **GetCapacityOfParamSets** method to query and return the number of parameter sets on the SIM card.

```
Private SMSMemory As SMS3ASuiteLib.ISMSMemory  
  
Private Function NumberOfParameterSets() As Long  
    On Error GoTo ErrorTrap  
    Call SMSMemory.GetCapacityOfParamSets( NumberOfParameterSets)  
    Exit Function  
ErrorTrap:  
    MsgBox "Error " & SMSMemory.GetLastError & _  
        " in GetCapacityOfParamSets."  
    NumberOfParameterSets = 0  
End Function
```

5.3.3.5 GetMemoryConfiguration

This method retrieves the configuration of the SMS memory.

Reserved for future use.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method GetMemoryConfiguration")]  
HRESULT GetMemoryConfiguration(  
    [out] long *pMemoryStatus  
);
```

The **GetMemoryConfiguration** method syntax has these parts:

Part	Description
<i>pMemoryStatus</i>	Output. Reference to the long integer variable in which the memory configuration will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

5.3.3.6 GetNumberOfMessagesInME

This method tells the total number of short messages stored in the phone's SMS memory.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method GetNumberOfMessagesInME")]  
HRESULT GetNumberOfMessagesInME(  
    [out] long *pAmountOfMessagesInMESMSMemory  
);
```

The **GetNumberOfMessagesInME** method syntax has these parts:

Part	Description
<i>pAmountOfMessagesInMESMSMemory</i>	Output. Reference to the long integer variable in which the number of short messages will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The analogous method for the SIM SMS memory is the **GetNumberOfMessagesInSIM** (5.3.3.7) method.

See also the **GetSMSMemoryStatus** (5.3.3.10) method.

Example

See **GetCapacityInME** (5.3.3.2)

5.3.3.7 GetNumberOfMessagesInSIM

This method tells the total number of short messages stored in the phone's SMS memory.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory **Syntax**

```
[helpstring("method GetNumberOfMessagesInSIM")]  
HRESULT GetNumberOfMessagesInSIM(  
    [out] long *pAmountOfMessagesInSIMSMSMemory  
);
```

The **GetNumberOfMessagesInSIM** method syntax has these parts:

Part	Description
<i>pAmountOfMessagesIn-SIMSMSMemory</i>	Output. Reference to the long integer variable in which the number of short messages will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The analogous method for the phone SMS memory is the **GetNumberOfMessagesInMe** (5.3.3.6) method.

See also the **GetSMSMemoryStatus** (5.3.3.10) method.

Example

See **GetCapacityInME** (5.3.3.2)

5.3.3.8 GetNumberOfUnreadInME

This method tells the number of unread short messages stored in the phone's SMS memory.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method GetNumberOfUnreadInME")]  
HRESULT GetNumberOfUnreadInME(  
    [out] long *pAmountOfUnreadMessagesInMESMSMemory  
);
```

The **GetNumberOfUnreadInME** method syntax has these parts:

Part	Description
<i>pAmountOf-Unread-MessagesIn-MESMS-Memory</i>	Output. Reference to the long integer variable in which the number of short messages will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The analogous method for the SIM SMS memory is the **GetNumberOfUnreadInSIM** (5.3.3.9) method.

See also the **GetSMSMemoryStatus** (5.3.3.10) method.

Example

See **GetCapacityInME** (5.3.3.2)

5.3.3.9 GetNumberOfUnreadInSIM

This method tells the number of unread short messages stored in the SIM card's SMS memory.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method GetNumberOfUnreadInSIM")]  
HRESULT GetNumberOfUnreadInSIM(  
    [out] long *pAmountOfUnreadMessagesInSIMSMSMemory  
);
```

The **GetNumberOfUnreadInSIM** method syntax has these parts:

Part	Description
<i>pAmountOfUnread-MessagesInSIMSMSMemory</i>	Output. Reference to the long integer variable in which the number of short messages will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The analogous method for the SIM SMS memory is the **GetNumberOfUnreadInME** (5.3.3.8) method.

See also the **GetSMSMemoryStatus** (5.3.3.10) method.

Example

See **GetCapacityInME** (5.3.3.2)

5.3.3.10 GetSMSMemoryStatus

This method tells the following facts about both the phone and the SIM SMS memories:

- Total number of memory positions in the SMS memory
- Total number of messages stored in the SMS memory
- Total number of unread messages in the SMS memory
- Configuration of the SMS memory
- Number of SMS parameter sets on the SIM card

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method GetSMSMemoryStatus")]
HRESULT GetSMSMemoryStatus(
    [out] long *pMESize,
    [out] long *pSIMSize,
    [out] long *pMessagesInME,
    [out] long *pMessagesInSIM,
    [out] long *pUnreadInME,
    [out] long *pUnreadInSIM,
    [out] long *pMemoryConfiguration,
    [out] long *pCapacityOfParamSets
);
```

The **GetSMSMemoryStatus** method syntax has these parts:

Part	Description
<i>pMESize</i>	Output. Reference to the long integer variable in which the total number of memory positions in the phone's SMS memory will be stored.
<i>pSIMSize</i>	Output. Reference to the long integer variable in which the total number of memory positions in the SIM card's SMS memory will be stored.
<i>pMessagesIn-ME</i>	Output. Reference to the long integer variable in which the total number of short messages stored in the phone's SMS memory will be stored.
<i>pMessagesIn-SIM</i>	Output. Reference to the long integer variable in which the total number of short messages stored in the SIM card's SMS memory will be stored.
<i>pUnreadInME</i>	Output. Reference to the long integer variable in which the number of unread short messages stored in the phone's SMS memory will be stored.
<i>pUnreadInSIM</i>	Output. Reference to the long integer variable in which the number of unread short messages stored in the SIM card's SMS memory will be stored.
<i>pMemory-Configuration</i>	Output. Reference to the long integer variable in which the memory configuration will be stored.

<i>pCapacityOf-ParamSets</i>	Output. Reference to the long integer variable in which the number of parameter sets will be stored.
------------------------------	--

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

Example

The following example method uses the **GetSMSMemoryStatus** method to query and display the above mentioned features of the SMS memory.

```
Private SMSMemory As SMS3ASuiteLib.ISMSMemory

Private Sub MemoryStatus()

    Dim MESSize As Long, SIMSize As Long
    Dim METotal As Long, SIMTotal As Long
    Dim MEUnread As Long, SIMUnread As Long
    Dim MemConf As Long, ParamSets As Long

    On Error GoTo ErrorTrap

    'get all the SMS memory information at once
    Call SMSMemory.GetSMSMemoryStatus(MESSize, SIMSize, _
        METotal, SIMTotal, MEUnread, SIMUnread, MemConf, ParamSets)

    MECapacity.Caption = MESSize
    SIMCapacity.Caption = SIMSize
    MessagesInME.Caption = METotal
    MessagesInSIM.Caption = SIMTotal
    UnreadInME.Caption = MEUnread
    UnreadInSIM.Caption = SIMUnread
    MemoryConfiguration.Caption = MemConf
    SMSParamSets.Caption = ParamSets

    Exit Sub

ErrorTrap:

    'inform the user about failure
    MsgBox "Error " & SMSMemory.GetLastError & _
        " in querying memory status."

End Sub
```

5.3.3.11 Read

This method reads a short message from a given SMS memory location.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method Read")]
HRESULT Read(
    [in] SMS_MEMORY_LOCATION SMSMemory,
    [in] long SMSIndex,
    [out] IShortMessage **ppSMS,
    [in] BOOL MarkMessageAsRead
);
```

The **Read** method syntax has these parts:

Part	Description
<i>SMS-Memory</i>	Input. A value of type SMS_MEMORY_LOCATION indicating the type of the SMS memory from which the short message will be read.
<i>SMSIndex</i>	Input. A long integer expression that evaluates the SMS memory index from which the short message will be read.
<i>ppSMS</i>	Output. Reference to a ShortMessage object which, on return, will contain the read short message.
<i>MarkMessageAsRead</i>	Input. A long integer expression that indicates whether the message will be marked as a read message or not.

Remarks

To set a short message as read, set *MarkMessageAsRead* parameters to 1. Otherwise, set it to 0.

Example

The example method below uses the **Read** method to read a short message either from the phone memory or from the SIM memory. The memory location from which to read is also given as a parameter.

```
Private SMSMemory As SMS3ASuiteLib.ISMSMemory

Private Sub ReadMessage( _
    SMSMemType As SMS_MEMORY_LOCATION, _
    MemoryIndex As Long, _
    MarkMessageAsRead As Long)

    Dim Msg As SMS3ASuiteLib.ShortMessage

    On Error GoTo ErrorTrap
```

```
'read the message
Call SMSMemory.Read(SMSMemType, MemoryIndex, Msg, _
    MarkMessageAsRead)

Dim MessageText As String

MessageText = "Message from " & Msg.OtherEndAddress & ": " & _
    Msg.UserDataText
If MarkMessageAsRead Then
    MessageText = MessageText & _
        " Message has been marked as read."
End If

MsgBox MessageText
Exit Sub

ErrorTrap:

'inform the user about failure
MsgBox "Error " & SMSMemory.GetLastError & _
    " in reading the short message."

End Sub
```

5.3.3.12 Store

This method stores a short message to a given SMS memory location.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSMemory

Syntax

```
[helpstring("method Store")]  
HRESULT Store(  
    [in] SMS_MEMORY_LOCATION SMSMemory,  
    [in] long SMSIndex,  
    [in] IShortMessage *pSMS  
);
```

The **Store** method syntax has these parts:

Part	Description
<i>SMSMemory</i>	Input. A value or constant of type SMS_MEMORY_LOCATION indicating the type of the SMS memory into which the short message will be stored.
<i>SMSIndex</i>	Input. A long integer expression that evaluates the SMS memory index into which the short message will be stored.
<i>pSMS</i>	Input. A ShortMessage object which will be stored in the SMS memory.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The short message object must be created before storing using the **CreateShortMsg** (5.3.1.1) function.

Example

The example method below uses the **Store** method to store the short message either in the phone memory or in the SIM memory.

```
Private SMSMemory As SMS3ASuiteLib.ISMSMemory  
Private ShortMsg As SMS3ASuiteLib.ShortMessage  
  
Private Sub StoreMessage( _  
    SMSMemType As SMS_MEMORY_LOCATION, _  
    MemoryIndex As Long, Msg As SMS3ASuiteLib.ShortMessage)  
  
    On Error GoTo ErrorTrap  
  
    'store the message  
    Call SMSMemory.Store(SMSMemType, MemoryIndex, Msg)  
  
    'inform the user about success
```



```
If SMSMemType = PHONE_MEMORY Then
    MsgBox "The short message has been stored " & _
        "in the phone memory in location " & MemoryIndex
ElseIf SMSMemType = SIM_MEMORY Then
    MsgBox "The short message has been stored " & _
        "in the SIM memory in location " & MemoryIndex
Else
    MsgBox "The short message has been stored " & _
        "in the default memory in location " & MemoryIndex
End If

Exit Sub

ErrorTrap:

    'inform the user about failure
    MsgBox "Error " & SMSMemory.GetLastError & _
        " in storing the short message."

End Sub
```

5.3.4 ISMSSettings Interface

The ISMSSettings interface contains methods and functions for querying and adjusting SMS related settings in the phone. These settings include, e.g., message center number, message validity period, and the protocol used for sending the message.

Method	Description
GetCommonSMSSettings	Returns common SMS settings from the phone.
GetSMSParametersSet	Returns a set of SMS parameters including protocol identifier, data coding scheme, SMS validity period and SMSC number.
SetCommonSMSSettings	Sets common SMS settings in the phone.
SetRoutingParameters	Reserved for future use.
SetSMSParametersSet	Adjusts a set of SMS parameters including protocol identifier, data coding scheme, SMS validity period and SMSC number.
StartListeningEvents	Informes the object that the client is ready to receive outgoing method calls from the object.
Terminate	Disconnects from the component server.
GetLastError	Returns the code number of the latest error.

5.3.4.1 GetCommonSMSSettings

This method is used to retrieve two common SMS settings, status report request mode and reply path mode, from the phone.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSettings **Syntax**

```
[helpstring("method GetCommonSMSSettings")]
HRESULT GetCommonSMSSettings(
    [out] long *pStatusReportRequest,
    [out] long *pReplyPath
);
```

The **GetCommonSMSSettings** method syntax has these parts:

Part	Description
<i>pStatusReportRequest</i>	Output. Reference to a long integer variable in which the indication whether the status report is requested will be stored.
<i>pReplyPath</i>	Output. Reference to a long integer variable in which the indication whether the recipient of the sent short message is allowed to use sender's SMSC with reply message will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

If status reports are requested for sent short messages, the *pStatusReportRequest* parameter equals 1. Otherwise, it equals 0.

If the recipient is allowed to send a reply message through the sender's SMSC, the *pReplyPath* parameter equals 1. Otherwise, it equals 0. Allowing the recipient to use the sender's SMSC to send a reply message means that the sender is willing to pay reply costs.

The analogous Set- method is **SetCommonSMSSettings** (5.3.4.3).

Example

This example method uses the **GetCommonSMSSettings** method to query the status report request mode and the reply path mode and to adjust the states of the corresponding check box controls.

```
Private SMSSettings As SMS3ASuiteLib.ISMSSettings

Private Sub GetCommonSettings()

    Dim StatusReportRequest As Long, ReplyPath As Long
```

```
On Error GoTo ErrorTrap

'get the settings
Call SMSSettings.GetCommonSMSSettings( _
    StatusReportRequest, ReplyPath)

'take care of the check box states
CB_RequestStatusReport.Value = StatusReportRequest
CB_AcceptReplyCost.Value = ReplyPath

Exit Sub

ErrorTrap:

'inform the user about failure
MsgBox "Error " & SMSSettings.GetLastError & _
    " in querying SMS settings."

End Sub
```

5.3.4.2 GetSMSParametersSet

This method returns a desired set of SMS parameters from the phone. The returned parameters are the parameter set name, protocol identifier, data coding scheme, validity period and SMSC phone number.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSettings **Syntax**

```
[helpstring("method GetSMSParametersSet")]
HRESULT GetSMSParametersSet(
    [in] long ParameterSetIndex,
    [out] BSTR *pParameterSetName,
    [out] BYTE *pProtocolIdentifier,
    [out] BYTE *pDataCodingScheme,
    [out] BYTE *pValidityPeriod,
    [out] BSTR *pSCAddress
);
```

The **GetSMSParametersSet** method syntax has these parts:

Part	Description
<i>ParameterSet-Index</i>	Input. A long integer value indicating the index of the SMS parameter set.
<i>pParameter-SetName</i>	Output. Reference to the string variable in which the name of the parameter set will be stored.
<i>pProtocol-Identifier</i>	Output. Reference to the byte variable in which the protocol identifier will be stored.
<i>pDataCoding-Scheme</i>	Output. Reference to the byte variable in which the data coding scheme will be stored.
<i>pValidity-Period</i>	Output. Reference to the byte variable in which the short message validity period will be stored.
<i>pSCAddress</i>	Output. Reference to the string variable in which the SMSC phone number will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

Invalid **ParameterSetIndex** values lead to either **errInvalidLocation** (if value is less than 256) or **errSmsInvalidParameterSetIndex** error (if equal to or greater than 256).

The corresponding Set- method is **SetSMSParametersSet** (5.3.4.4).

Maximum **ParameterSetName** length depends on the SIM used. No error is returned if the parameter set name is truncated.

For detailed information about protocol identifiers, see the **ProtocolIdentifier** property.

For detailed information about data coding schemes, see the **DataCodingScheme** property.

For detailed information about validity periods, see the **ValidityPeriodFormat** property.

Example

The following example method uses the **GetSMSParametersSet** method to query an SMS parameter set from the phone and display the parameters.

Below, useful constants have been defined for different message validity periods and protocol identifiers.

```
Private SMSSettings As SMS3ASuiteLib.ISMSSettings

'SMS validity periods
Const SMS_MAX_TIME = 255
Const SMS_ONE_WEEK = 173
Const SMS_THREE_DAYS = 169
Const SMS_24_HOURS = 167
Const SMS_SIX_HOURS = 71
Const SMS_ONE_HOUR = 11

'SMS protocol identifiers
Const SMS_PROTOCOL_TEXT = 0
Const SMS_PROTOCOL_FAX = 34
Const SMS_PROTOCOL_PAGER = 38
Const SMS_PROTOCOL_EMAIL = 50

Private Sub GetParametersSet(ByVal ParameterSetIndex As Long)

    Dim ParameterSetName As String, SCAddress As String
    Dim ProtocolIdentifier As Byte, DataCodingScheme As Byte
    Dim ValidityPeriod As Byte

    'query the settings from the phone
    Call SMSSettings.GetSMSParametersSet( _
        ParameterSetIndex, ParameterSetName, ProtocolIdentifier, _
        DataCodingScheme, ValidityPeriod, SCAddress)

    Parameters.Caption = "SMS parameters for set #" & _
        ParameterSetIndex
    SetName.Caption = ParameterSetName

    Select Case ProtocolIdentifier
        Case SMS_PROTOCOL_TEXT
            ProtocolID.Caption = "Text"
        Case SMS_PROTOCOL_FAX
            ProtocolID.Caption = "Fax"
        Case SMS_PROTOCOL_PAGER
            ProtocolID.Caption = "Pager"
        Case SMS_PROTOCOL_EMAIL
            ProtocolID.Caption = "Email"
    End Select

    DCS.Caption = DataCodingScheme

    Select Case ValidityPeriod
        Case SMS_MAX_TIME
```

```
        Validity.Caption = "Max. time"
    Case SMS_ONE_WEEK
        Validity.Caption = "One week"
    Case SMS_THREE_DAYS
        Validity.Caption = "Three days"
    Case SMS_24_HOURS
        Validity.Caption = "24 hours"
    Case SMS_SIX_HOURS
        Validity.Caption = "Six hours"
    Case SMS_ONE_HOUR
        Validity.Caption = "One hour"
End Select

SMSCNumber.Caption = SCAddress

End Sub
```

5.3.4.3 SetCommonSMSSettings

This method is used to set two common SMS settings, status report request mode and reply path mode, in the phone.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSettings **Syntax**

```
[helpstring("method SetCommonSMSSettings")]
HRESULT SetCommonSMSSettings(
    [in] long StatusReportRequest,
    [in] long ReplyPath
);
```

The **SetCommonSMSSettings** method syntax has these parts:

Part	Description
<i>Status-Report-Request</i>	Input. A long integer expression indicating whether the status report is requested.
<i>ReplyPath</i>	Input. A long integer expression indicating whether the recipient of the sent short message is allowed to use sender's SMSC with reply message.

Remarks

If you want to receive status reports for sent short messages, set the *StatusReportRequest* parameter to 1. Otherwise, set it to 0.

If you want to allow the recipient to send a reply message through your SMSC, set the *ReplyPath* parameter to 1. Otherwise, set it to 0. Allowing the recipient to use your SMSC to send a reply message means that you are willing to pay reply costs.

The analogous Get- method is **GetCommonSMSSettings** (5.3.4.1).

Example

The example method below uses the **SetCommonSMSSettings** method to set the status report request mode and the reply path mode according to the corresponding check box control states.

```
Private SMSSettings As SMS3ASuiteLib.ISMSSettings

Private Sub SetCommonSettings()

    On Error GoTo ErrorTrap

    'set the settings according to the corresponding
    'check box control states
    Call SMSSettings.SetCommonSMSSettings(_
        CB_RequestStatusReport.Value, CB_AcceptReplyCost.Value)

Exit Sub
```


ErrorTrap:

```
'inform the user about failure
MsgBox "Error " & SMSSettings.GetLastError & _
      " in setting SMS settings."
```

End Sub

5.3.4.4 SetSMSParametersSet

This method sets a desired set of SMS parameters for the phone. The parameters set are parameter set name, protocol identifier, data coding scheme, validity period and SMSC phone number.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.ISMSSettings **Syntax**

```
[helpstring("method SetSMSParametersSet")]  
HRESULT SetSMSParametersSet(  
    [in] long ParameterSetIndex,  
    [in] BSTR ParameterSetName,  
    [in] BYTE ProtocolIdentifier,  
    [in] BYTE DataCodingScheme,  
    [in] BYTE ValidityPeriod,  
    [in] BSTR SCAddress  
);
```

The **SetSMSParametersSet** method syntax has these parts:

Part	Description
<i>ParameterSet-Index</i>	Input. A long integer value that evaluates to the index of the SMS parameter set.
<i>Parameter-SetName</i>	Input. A string containing the name of the parameter set.
<i>Protocol-Identifier</i>	Input. A byte expression that evaluates to protocol identifier.
<i>DataCoding-Scheme</i>	Input. A byte expression that evaluates to data coding scheme.
<i>ValidityPeriod</i>	Input. A byte expression that evaluates to short message validity period.
<i>SCAddress</i>	Input. A string containing the SMSC phone number.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

Invalid **ParameterSetIndex** values lead to either an `errInvalidLocation` or `errSmsInvalid-ParameterSetIndex` error.

The corresponding Get- method is **GetSMSParametersSet** (5.3.4.2).

Maximum **ParameterSetName** length depends on the SIM used. No error is returned if the parameter set name is truncated.

For detailed information about protocol identifiers, see the **ProtocolIdentifier** property. An invalid ProtocolIdentifier parameter results in an error code, which may refer to another parameter.

For detailed information about data coding schemes, see the **DataCodingScheme** property.

For detailed information about validity periods, see the **ValidityPeriodFormat** property.

Example

The following example uses the **SetSMSParametersSet** method to set the parameters of the first SMS parameter set. The name of the set is set to "FirstSet"; protocol to text; validity period to 24 hours; and SMSC phone number to +358207654321. Default value 0 is used with the data coding scheme as recommended.

Below, useful constants have been defined for different message validity periods and protocol identifiers.

```
'SMS validity periods
Const SMS_MAX_TIME = 255
Const SMS_ONE_WEEK = 173
Const SMS_THREE_DAYS = 169
Const SMS_24_HOURS = 167
Const SMS_SIX_HOURS = 71
Const SMS_ONE_HOUR = 11

'SMS protocol identifiers
Const SMS_PROTOCOL_TEXT = 0
Const SMS_PROTOCOL_FAX = 34
Const SMS_PROTOCOL_PAGER = 38
Const SMS_PROTOCOL_EMAIL = 50

'data coding scheme
Const SMS_DCS_DEFAULT = 0

Private Sub Form_Load()

    Dim SMS As SMS3ASuiteLib.SMS_SuiteAdapter
    Dim SMSSettings As SMS3ASuiteLib.ISMSSettings

    On Error GoTo ErrorTrap

    Set SMS = New SMS3ASuiteLib.SMS_SuiteAdapter
    Set SMSSettings = SMS

    Call SMSSettings.SetSMSParametersSet(1, "FirstSet", _
        SMS_PROTOCOL_TEXT, SMS_DCS_DEFAULT, SMS_24_HOURS, _
        "+358207654321")

    GoTo Success

ErrorTrap:

    MsgBox "Error " & SMSSettings.GetLastError & _
        " in SMS parameter setting."

Success:
```

```
    Set SMSSettings = Nothing

    SMS.Terminate
    Set SMS = Nothing

End Sub
```

5.3.5 IGraphicalMS Interface

This interface contains methods for handling the phone's GMS (Graphical Messaging System) capabilities. Its features include creating a picture object and storing it in the phone. More information about GMS can be found from Nokia Forum at <http://forum.nokia.com/main/>.

Method	Description
CreateGMSObjebt	Create a new, empty GMS message object.
FindNextGMS	Get the next GMS message from the phone.
SaveGMS	Save a GMS message in the phone.

5.3.5.1 CreateGMSObject

This method creates an empty data component for one graphical message.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.IGraphicalMS

Syntax

```
[helpstring("method CreateGMSObject")]  
    HRESULT CreateGMSObject(  
        [out, retval] IGMSPicture ** ppIPicture  
    );
```

The **CreateGMSObject** method syntax has these parts:

Part	Description
<i>ppIPicture</i>	Output. A reference to an empty graphical message data object.

Remarks

Before use, the object must be filled via the properties found from the GMSPicture component's IGMSPicture interface. See description of GMSPicture data componet for more details.

5.3.5.2 FindNextGMS

This method retrieves the next graphical message from the phone, pointed by index.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.IGraphicalMS **Syntax**

```
[helpstring("method FindNextGMS")]  
HRESULT FindNextGMS([in] VARIANT_BOOL updateStatus,  
[in, out] STORAGE_STATUS *status,  
[in, out] short *pNext,  
[out, retval] IGMSPicture ** ppIPicture  
);
```

The FindNextGMS method syntax has these parts:

Part	Description
<i>updateStatus</i>	Input. Update messages memory status to read or not.
<i>status</i>	Input/Output. Upon return, tells the memory status of the GMS message
<i>pNext</i>	Input/Output. Index to start the GMS search. Index of the returned GMS message is returned. Index starts from 1 (one).
<i>ppIPicture</i>	Output. A reference to graphical message data object.

Remarks

See description of STORAGE_STATUS data type for more details about different statuses.

5.3.5.3 SaveGMS

This method saves graphical message given as parameter.

Member of SMS3ASuiteLib.SMS_SuiteAdapter.IGraphicalMS **Syntax**

```
[helpstring("method SaveGMS")]  
HRESULT SaveGMS([in] STORAGE_STATUS status,  
[in] IGMSPicture * pIPicture,  
[in, out] short *pIndex  
);
```

The SaveGMS method syntax has these parts:

Part	Description
<i>status</i>	Memory status of the GMS message.
<i>pIPicture</i>	GMSPicture object to be saved to phone.
<i>pIndex</i>	Location index, where to store message. 0xffff for first free location. Returns the index where actually saved.

Remarks

If this graphical message will be saved in the phone memory, all Short Message related parameters (properties) in the IGMSPicture interface are ignored. A message is saved by using index 0 (zero) and will be located in the first free location in the phone. Status-parameter can be, e.g., NOT_SENT_FROM_PHONE.

Sending over air can be made by ISMSSend::Send(IShortMessage*) method. Some of IShortMessage-properties have to be set. These are:

- SMSC-number
- Recipient's phone number
- Validity period
- Data Coding Scheme (value 245, see GSM 03.40 for more detailed information)
- Protocol Identifier (0 (zero), see GSM 03.40 for more detailed information)
- Message type (SUBMIT_MESSAGE)
- User Data Format (DATA_CODING_SCHEME_BASED)

In addition, User Data Header and Message Data Header (and lengths) must be set.

The message data (IShortMessage::UserData) is pure *PicData* of IGMSPicture-interface. Two different cases can be separated; GMS-picture fits into one SMS or not. If the picture doesn't fit in one message, concatenated SMS is used for delivering. The next example describes how the message begins in these different cases.

When the size of picture including user data header is up to 140 8-bit characters.

BYTE[0] == 0x06	Length of user data header after this byte.
BYTE[1] == 0x05	Port number related byte. See GSM 03.40
BYTE[2] == 0x04	Port number related byte. See GSM 03.40
BYTE[3] == 0x15	Destination NBS-port High
BYTE[4] == 0x83	Destination NBS-port Low
BYTE[5] == 0x15	Originator NBS-port High
BYTE[6] == 0x83	Originator NBS-port Low

.....

When the picture size is over 140 8-bit characters. The maximum size of one GMS-message is the size of 3 (three) concatenated SMSs.

BYTE[0] == 0x0b	Length of user data header after this byte.
BYTE[1] == 0x00	0 (zero)
BYTE[2] == 0x03	How many bytes still left in this concatenated header
BYTE[3] ==	0x00 – 0xFF; the reference identifier of message.
BYTE[4] ==	The number of needed SMS-messages in this delivery.
BYTE[5] ==	The serial number of message.
BYTE[6] == 0x05	Port number related byte. See GSM 03.40
BYTE[7] == 0x04	Port number related byte. See GSM 03.40
BYTE[8] == 0x15	Destination NBS-port High
BYTE[9] == 0x83	Destination NBS-port Low
BYTE[10] == 0x15	Originator NBS-port High
BYTE[11] == 0x83	Originator NBS-port Low

.....

5.4 ShortMessage Component

This component contains properties of a short message and functions to adjust various short message attributes.

A ShortMessage component is a data component that is used to deliver the data of a single short message to/from the phone.

5.4.1 IShortMessage Interface

An IShortMessage interface is the only interface of the ShortMessage component and contains the properties of the short message object and a few access methods to query or adjust some short message parameters.

A new short message object is always instantiated using the **CreateShortMsg** (5.3.1.1) method, which is included both in SMS_SuiteAdapter and in ISMSMemory interfaces. Reference to the existing short message is received with the **Read** (5.3.3.11) method through ISMSMemory interface. A **ShortMsgReceived** (5.3.2.1) event included in an SMS_SuiteAdapter interface also delivers an object reference to the newly received short message.

Method	Description
get_SCTimeStamp	Returns the time when the SMSC received the message.
get_VValidityPeriodAbsolute	Returns the short message validity period in the SMSC in absolute format.
get_VValidityPeriodEnhanced	Reserved for future use.
put_SCTimeStamp	Changes the time when the SMSC received the message.
put_VValidityPeriodAbsolute	Sets the short message validity period in the SMSC by using absolute format.
put_VValidityPeriodEnhanced	Reserved for future use.

Property	Description
CommandMessageNumber	Message identification number given by GSM network.
CommandType	Indication of command message type.

DataCodingScheme	Data coding scheme.
MessageReference	GSM network's identification number for the SMS.
MessageType	The type of short message.
OtherEndAddress	The phone number of the short message sender (for received messages) or recipient (for messages to be sent).
ProtocolIdentifier	Identifier of the protocol used to send the short message.
ReplyPath	Indication of the sender's willingness to pay reply message costs.
SCAddress	SMSC phone number.
Status	Indication of the status of the previously sent short message.
StatusReportQualifier	Indication of the message type to which the status report relates.
StatusReportRequest	Indication of the sender's willingness to receive status messages about sent short messages.
StorageStatus	Short message storage status in the SMS memory.
UserData	The message field used if the short message is coded with 8 bit data.
UserDataFormat	The format that is used to code the short message data.
UserDataHeader	Special information field of the SMS (optional).
UserDataHeaderLength	The length of the UserDataHeader property in bytes.
UserDataLength	The length of the UserData property in bytes.
UserDataText	The actual short message text.
ValidityPeriodFormat	Indication of which format is used to express short message validity period in the SMSC.
ValidityPeriodRelative	Short message validity period in the SMSC in relative format.

5.4.1.1 get_SCTimeStamp

This method is used to retrieve the absolute time when the short message was received by the SMSC and to retrieve the SMSC time zone.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Syntax

```
[helpstring("method get_SCTimeStamp")]
HRESULT get_SCTimeStamp(
    [out] long *pYear,
    [out] long *pMonth,
    [out] long *pDay,
    [out] long *pHour,
    [out] long *pMinute,
    [out] long *pSecond,
    [out] long *pTimeZone
);
```

The **get_SCTimeStamp** method syntax has these parts:

Part	Description
<i>pYear</i>	Output. Reference to a long integer variable in which the year will be stored.
<i>pMonth</i>	Output. Reference to a long integer variable in which the month will be stored.
<i>pDay</i>	Output. Reference to a long integer variable in which the day will be stored.
<i>pHour</i>	Output. Reference to a long integer variable in which the hour will be stored.
<i>pMinute</i>	Output. Reference to a long integer variable in which the minutes will be stored.
<i>pSecond</i>	Output. Reference to a long integer variable in which the seconds will be stored.
<i>pTimeZone</i>	Output. Reference to a long integer variable in which the SMSC time zone indicator will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The time zone of SMSC is calculated as follows: 15 minutes * *object.pTimeZone*. For example, in Finland the time zone is 15 minutes * 8 = 120 minutes = 2 hours ahead of Greenwich time.

The corresponding Put-method is **put_SCTimeStamp** (5.4.1.3).

See GSM 03.40 for more detailed information.

Example

This example method uses the **get_SCTimeStamp** method to query the time at which the received short message given as a parameter arrived to the SMSC.

```
Private Sub ShowSCTimeStamp(Msg As SMS3ASuiteLib.ShortMessage)

    Dim Year As Long, Month As Long, Day As Long
    Dim Hour As Long, Minute As Long, Second As Long
    Dim TimeZone As Long

    On Error Resume Next

    'get the time stamp
    Call Msg.get_SCTimeStamp(Year, Month, Day, Hour, Minute, _
        Second, TimeZone)

    Dim TimeStamp As String
    TimeStamp = Day & "/" & Month & "/" & Year & " " & Hour & _
        "." & Minute & ":" & Second & "."

    Dim TimeDifference As String
    TimeDifference = Chr(10) & "(Difference " & 15 * _
        TimeZone / 60 & " hours to Greenwich time)"

    'display the time stamp and timing difference
    MsgBox "This message arrived to SMSC at " & _
        TimeStamp & TimeDifference

End Sub
```

5.4.1.2 get_ValidityPeriodAbsolute

This method is used to retrieve the absolute time when the short message will go out of date in the SMSC.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Syntax

```
[helpstring("method get_ValidityPeriodAbsolute")]
HRESULT get_ValidityPeriodAbsolute(
    [out] long *pYear,
    [out] long *pMonth,
    [out] long *pDay,
    [out] long *pHour,
    [out] long *pMinute,
    [out] long *pSecond,
    [out] long *pTimeZone
);
```

The **get_ValidityPeriodAbsolute** method syntax has these parts:

Part	Description
<i>pYear</i>	Output. Reference to a long integer variable in which the year will be stored.
<i>pMonth</i>	Output. Reference to a long integer variable in which the month will be stored.
<i>pDay</i>	Output. Reference to a long integer variable in which the day will be stored.
<i>pHour</i>	Output. Reference to a long integer variable in which the hour will be stored.
<i>pMinute</i>	Output. Reference to a long integer variable in which the minutes will be stored.
<i>pSecond</i>	Output. Reference to a long integer variable in which the seconds will be stored.
<i>pTimeZone</i>	Output. Reference to a long integer variable in which the SMSC time zone indicator will be stored.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The time zone of SMSC is calculated as follows: 15 minutes * *object.pTimeZone*. For example, in Finland the time zone is 15 minutes * 8 = 120 minutes = 2 hours ahead of Greenwich time.

The corresponding Put- method is **put_ValidityPeriodAbsolute** (5.4.1.4).

Example

This example uses the **get_ValidityPeriodAbsolute** method to display date and time when the given short message will become old.

```
Private Sub ValidityPeriod(Msg As SMS3ASuiteLib.ShortMessage)

    Dim Year As Long, Month As Long, Day As Long
    Dim Hour As Long, Minute As Long, Second As Long
    Dim TimeZone As Long

    On Error Resume Next

    'get the time stamp
    Call Msg.get_ValidityPeriodAbsolute(Year, Month, Day, _
        Hour, Minute, Second, TimeZone)

    Dim Validity As String
    Validity = Day & "/" & Month & "/" & Year & " " & Hour & _
        "." & Minute & ":" & Second & "."

    Dim TimeDifference As String
    TimeDifference = Chr(10) & "(Difference " & 15 * _
        TimeZone / 60 & " hours to Greenwich time)"

    'display the time stamp and timing difference
    MsgBox "This message will go old at " & Validity & TimeDifference

End Sub
```

5.4.1.3 put_SCTimeStamp

This method is used to set the absolute time when the short message was received by the SMSC and to set the SMSC time zone.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Syntax

```
[helpstring("method put_SCTimeStamp")]
HRESULT put_SCTimeStamp(
    [in] long year,
    [in] long month,
    [in] long day,
    [in] long hour,
    [in] long minute,
    [in] long second,
    [in] long timeZone
);
```

The **put_SCTimeStamp** method syntax has these parts:

Part	Description
<i>year</i>	Input. A long integer expression that evaluates to the year when the short message was received by the SMSC. Valid range is 0–99.
<i>month</i>	Input. A long integer expression that evaluates to the month when the short message was received by the SMSC. Valid range is 1–12.
<i>day</i>	Input. A long integer expression that evaluates to the day when the short message was received by the SMSC. Valid range is 1–31.
<i>hour</i>	Input. A long integer expression that evaluates to the hour when the short message was received by the SMSC. Valid range is 0–23.
<i>minute</i>	Input. A long integer expression that evaluates to the minute when the short message was received by the SMSC. Valid range is 0–59.
<i>second</i>	Input. A long integer expression that evaluates to the second when the short message was received by the SMSC. Valid range is 0–59.
<i>timeZone</i>	Input. A long integer expression that evaluates to the SMSC time zone indicator. Valid range is 0–96.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The SMSC time zone indicator is set as follows: *object.timeZone* = difference to Greenwich time in minutes / 15 minutes. For example, Finnish time is two hours ahead of Greenwich time, so the time zone must be 120 minutes / 15 minutes = 8 in Finland.

The corresponding Get- method is **get_SCTimeStamp** (5.4.1.1).

See GSM 03.40 for more detailed information.

5.4.1.4 put_ValidityPeriodAbsolute

This method is used to set the absolute time when the short message will go out of date in the SMSC.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Syntax

```
[helpstring("method put_ValidityPeriodAbsolute")]  
HRESULT put_ValidityPeriodAbsolute(  
    [in] long year,  
    [in] long month,  
    [in] long day,  
    [in] long hour,  
    [in] long minute,  
    [in] long second,  
    [in] long timeZone  
);
```

The **put_ValidityPeriodAbsolute** method syntax has these parts:

Part	Description
<i>year</i>	Input. A long integer expression that evaluates to the year when the short message will go out of date in SMSC. Valid range 0–99.
<i>month</i>	Input. A long integer expression that evaluates to the month when the short message will go out of date in SMSC. Valid range 1–12.
<i>day</i>	Input. A long integer expression that evaluates to the day when the short message will go out of date in SMSC. Valid range 1–31.
<i>hour</i>	Input. A long integer expression that evaluates to the hour when the short message will go out of date in SMSC. Valid range 0–23.
<i>minute</i>	Input. A long integer expression that evaluates to the minute when the short message will go out of date in SMSC. Valid range 0–59.
<i>second</i>	Input. A long integer expression that evaluates to the second when the short message will go out of date in SMSC. Valid range 0–59.
<i>timeZone</i>	Input. A long integer expression that evaluates to the SMSC time zone indicator. Valid range 0–96.

Remarks

If the method fails, call the **GetLastError** (5.3.1.2) function to get extended error information.

The SMSC time zone indicator is set as follows: *object.timeZone* = difference to Greenwich time in minutes / 15 minutes. For example, Finnish time is two hours ahead of Greenwich time, so the time zone must be 120 minutes / 15 minutes = 8 in Finland.

The corresponding Get- method is **get_ValidityPeriodAbsolute** (5.4.1.2).

Example

The following example method uses **put_ValidityPeriodAbsolute** method to set the validity period of the short message to a value given as a parameter.

```
Private ShortMsg As SMS3ASuiteLib.ShortMessage

Private Sub SetValidity(ByVal VP As Date)

    Dim VPYear As Integer
    Dim TimeZone As Long

    'year is expressed with two numbers
    'e.g., 99 instead of 1999
    VPYear = Year(VP) - 100 * Int(Year(VP) / 100)
    VPYear = 0

    'Finnish time is 2 hours (120 minutes) ahead
    'Greenwich time
    TimeZone = 120 / 15

    'set validity period format
    ShortMsg.ValidityPeriodFormat = VP_ABSOLUTE

    'set the validity period
    Call ShortMsg.put_ValidityPeriodAbsolute(VPYear, Month(VP), _
        Day(VP), Hour(VP), Minute(VP), Second(VP), TimeZone)

End Sub
```

5.4.1.5 CommandMessageNumber (rw)

This property contains the reference (number) to the message that the command message concerns.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage Type: **Byte**

Default value: 0

Syntax

```
[propget, helpstring("property CommandMessageNumber")]  
HRESULT CommandMessageNumber(  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property CommandMessageNumber")]  
HRESULT CommandMessageNumber(  
    [in] unsigned char newVal  
);
```

Remarks

This property is command message specific information.

See GSM 03.40 for more detailed information.

5.4.1.6 CommandType (rw)

This property contains a command that wants to be executed by the short message service center (SMSC) using the command message.

Type: **Byte**

Default value: 0

Member of SMS3ASuiteLib.ShortMessage.IShortMessage **Syntax**

```
[propget, helpstring("property CommandType")]  
HRESULT CommandType(  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property CommandType")]  
HRESULT CommandType(  
    [in] unsigned char newVal  
);
```

Remarks

This property is command message specific information.

See GSM 03.40 for more detailed information.

5.4.1.7 DataCodingScheme (rw)

This property contains information, as specified in GSM 03.38 including SMS class, user data coding and compression information.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Byte**

Default value: 0

Syntax

```
[propget, helpstring("property DataCodingScheme")]  
HRESULT DataCodingScheme(  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property DataCodingScheme")]  
HRESULT DataCodingScheme(  
    [in] unsigned char newVal  
);
```

Remarks

Using the default value 0 is highly recommended.

See GSM 03.38 for more detailed information.

5.4.1.8 MessageReference (rw)

This property contains the identification number of a short message given by the GSM network.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Byte**

Default value: 0

Syntax

```
[propget, helpstring("property MessageReference")]  
HRESULT MessageReference(  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property MessageReference")]  
HRESULT MessageReference(  
    [in] unsigned char newVal  
);
```

Remarks

With this property, it is possible to make references between short messages; e.g., it is possible to find out which short message the received status report concerns. You can also discard duplicated short messages by examining their message identification numbers and time stamps.

See GSM 03.40 for more detailed information.

Example

The following code snippet shows how to display message reference number in the text control.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
    ...  
    'message reference  
    Reference.Caption = cSMS.MessageReference  
    ...  
End Sub
```

5.4.1.9 MessageType (rw)

This property contains the short message type, i.e., the function of the message.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: SMS_MESSAGE_TYP

Default value: SUBMIT_MESSAGE

Syntax

```
[propget, helpstring("property MessageType")]  
HRESULT MessageType(  
    [out, retval] SMS_MESSAGE_TYPE *pVal  
);
```

```
[propput, helpstring("property MessageType")]  
HRESULT MessageType(  
    [in] SMS_MESSAGE_TYPE newVal  
);
```

Remarks

See GSM 03.40 for more detailed information.

Example

The following method displays the short message type to the user in a text control. If the message is a status report, it is checked whether the report is sent for the command message or for the submit message.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
    Select Case cSMS.MessageType  
  
        Case DELIVER_MESSAGE  
            SMSType.Caption = "Deliver message"  
  
        Case STATUS_REPORT_MESSAGE  
  
            If cSMS.StatusReportQualifier = COMMAND_REQUESTED Then  
                SMSType.Caption = "Status report for command " & _  
                    "message - status " & cSMS.Status  
            ElseIf cSMS.StatusReportQualifier = SUBMIT_REQUESTED Then  
                SMSType.Caption = "Status report for submit " & _  
                    "message - status " & cSMS.Status  
            End If  
  
        Case SUBMIT_MESSAGE  
            SMSType.Caption = "Submit"  
  
        Case COMMAND_MESSAGE  
            SMSType.Caption = "Command"
```



```
Case MO_UNDEFINED_MESSAGE
    SMSType.Caption = "Undefined MO"

Case MT_UNDEFINED_MESSAGE
    SMSType.Caption = "Undefined MT"

End Select

End Sub
```

5.4.1.10 OtherEndAddress (rw)

This property contains the phone number of the sender for MT messages and the phone number of the receiver for MO messages.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **String**

Default value: Empty string

Syntax

```
[propget, helpstring("property OtherEndAddress")]  
HRESULT OtherEndAddress(  
    [out, retval] BSTR *pVal  
);
```

```
[propput, helpstring("property OtherEndAddress")]  
HRESULT OtherEndAddress(  
    [in] BSTR newVal  
);
```

Remarks

This property must always be set before sending the short message using the **Send** (5.3.1.3) method.

Example

See example of CreateShortMsg method.

5.4.1.11 ProtocolIdentifier (rw)

This property contains the higher layer protocol being used when delivering the short message or indicates interworking with a certain type of telematic device.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Byte**

Default value: 0 (text)

Syntax

```
[propget, helpstring("property ProtocolIdentifier")]  
HRESULT ProtocolIdentifier(  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property ProtocolIdentifier")]  
HRESULT ProtocolIdentifier(  
    [in] unsigned char newVal  
);
```

Remarks

It is highly recommended to use the default value 0 (text).

See GSM 03.40 for more detailed information.

Example

The following method displays the message protocol in the text control.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
  
    Select Case cSMS.ProtocolIdentifier  
        Case SMS_PROTOCOL_TEXT  
            PID.Caption = "Text"  
        Case SMS_PROTOCOL_FAX  
            PID.Caption = "Fax"  
        Case SMS_PROTOCOL_PAGER  
            PID.Caption = "Pager"  
        Case SMS_PROTOCOL_EMAIL  
            PID.Caption = "Email"  
    End Select  
  
End Sub
```

5.4.1.12 ReplyPath (rw)

This property contains an indication of whether the sender agrees to pay reply message costs.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Long**

Default value: 0

Syntax

```
[propget, helpstring("property ReplyPath")]  
HRESULT ReplyPath(  
    [out, retval] BOOL *pVal  
);
```

```
[propput, helpstring("property ReplyPath")]  
HRESULT ReplyPath(  
    [in] BOOL newVal  
);
```

Remarks

When this property equals 1, the recipient is allowed to send a reply message through the sender's short message center. When it equals 0, the recipient has to use his own short message center to reply.

If the reply cost is accepted, the recipient is allowed to reply to the message, even if he does not subscribe to the SMS service.

See GSM 03.40 for more detailed information.

Example

The following method shows whether the message sender is willing to pay the reply message.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
    If cSMS.ReplyPath = 1 Then  
        ReplyCost.Caption = "Yes"  
    Else  
        ReplyCost.Caption = "No"  
    End If  
End Sub
```

5.4.1.13 SCAddress (rw)

This property contains the GSM operator's short message center (SMSC) phone number, which is needed to send short messages. The number is expressed in the international phone number format.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **String**

Default value: Empty string

Syntax

```
[propget, helpstring("property SCAddress")]  
HRESULT SCAddress(  
    [out, retval] BSTR *pVal  
);
```

```
[propput, helpstring("property SCAddress")]  
HRESULT SCAddress(  
    [in] BSTR newVal  
);
```

Remarks

This property must always be set before sending the short message, using the Send method. It is not stored in the phone.

Example

See **CreateShortMsg** (5.3.1.1)

5.4.1.14 Status (rw)

This property contains an indication of the status of the previously sent short message.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Byte**

Default value: 0

Syntax

```
[propget, helpstring("property Status")]  
HRESULT Status(  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property Status")]  
HRESULT Status(  
    [in] unsigned char newVal  
);
```

Remarks

This property is status report message specific information.

See GSM 03.40 for more detailed information.

5.4.1.15 StatusReportQualifier (rw)

This property contains an indication of whether the status report is for the submit message or for the command message.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: REPORT_QUALIFIER

Default value: SUBMIT_REQUESTED

Syntax

```
[propget, helpstring("property StatusReportQualifier")]  
HRESULT StatusReportQualifier(  
    [out, retval] REPORT_QUALIFIER *pVal  
);
```

```
[propput, helpstring("property StatusReportQualifier")]  
HRESULT StatusReportQualifier(  
    [in] REPORT_QUALIFIER newVal  
);
```

Remarks

This property is status report message specific information.

See GSM 03.40 for more detailed information.

Example

See **MessageType** (5.4.1.9)

5.4.1.16 **StatusReportRequest (rw)**

This property tells if the status report message is requested for the current message.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Long**

Default value: 0

Syntax

```
[propget, helpstring("property StatusReportRequest")]  
HRESULT StatusReportRequest(  
    [out, retval] long *pVal  
);
```

```
[propput, helpstring("property StatusReportRequest")]  
HRESULT StatusReportRequest(  
    [in] long newVal  
);
```

Remarks

Set this property to 1 if you want to receive a status report message. Otherwise, set it to 0.

The status report confirms that the SMSC has received the short message or that it was impossible to deliver the short message to the SMSC. See GSM 03.40 for more detailed information.

Example

The following method shows whether the status report for the current message is requested after sending it.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
    'status report request  
    If cSMS.StatusReportRequest = 1 Then  
        StatusReport.Caption = "Yes"  
    Else  
        StatusReport.Caption = "No"  
    End If  
End Sub
```


5.4.1.17 **StorageStatus (rw)**

This property contains the short message storage status in the SMS memory.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: STORAGE_STATUS

Default value: NOT_SENT_FROM_PHONE

Syntax

```
[propget, helpstring("property StorageStatus")]  
HRESULT StorageStatus(  
    [out, retval] STORAGE_STATUS *pVal  
);
```

```
[propput, helpstring("property StorageStatus")]  
HRESULT StorageStatus(  
    [in] STORAGE_STATUS newVal  
);
```

Example

The following method displays the short message storage status in the text control.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
  
    If cSMS.StorageStatus = NOT_SENT_FROM_PHONE Then  
        MessageStatus.Caption = "Not sent."  
    ElseIf cSMS.StorageStatus = SENT_FROM_PHONE Then  
        MessageStatus.Caption = "Sent."  
    ElseIf cSMS.StorageStatus = DELIVERED Then  
        MessageStatus.Caption = "Delivered."  
    ElseIf cSMS.StorageStatus = READ_FROM_PHONE Then  
        MessageStatus.Caption = "Read."  
    ElseIf cSMS.StorageStatus = NOT_READ_FROM_PHONE Then  
        MessageStatus.Caption = "Not read."  
    End If  
  
End Sub
```

5.4.1.18 UserData (rw)

This property contains the actual short message, when the data is in the data format (i.e., 8 bit coding format has been used).

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Byte**

Syntax

```
[propget, helpstring("property UserData")]  
HRESULT UserData(  
    [in] long index,  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property UserData")]  
HRESULT UserData(  
    [in] long index,  
    [in] unsigned char newVal  
);
```

The **UserData** property has these parts:

Part	Description
<i>index</i>	This data structure is filled byte by byte. Use loop and running index. Valid range of values is from 0 to UserDataLength-1.

5.4.1.19 UserDataFormat (rw)

This property tells the format that is used when coding the **UserData** property.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: USER_DATA_FORMAT

Default value: DEFAULT_ALPHABET_7_BIT

Syntax

```
[propget, helpstring("property UserDataFormat")]  
HRESULT UserDataFormat(  
    [out, retval] USER_DATA_FORMAT *pVal  
);
```

```
[propput, helpstring("property UserDataFormat")]  
HRESULT UserDataFormat(  
    [in] USER_DATA_FORMAT newVal  
);
```

Remarks

See GSM 03.38 and 03.40 for more detailed information.

Example

The following example method displays the short message data format in the text control.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
    'user data format  
    Select Case cSMS.UserDataFormat  
        Case DATA_CODING_SCHEME_BASED  
            UserDataFormat.Caption = "Data coding scheme based"  
        Case DEFAULT_ALPHABET_7_BIT  
            UserDataFormat.Caption = "Default alphabet (7-bit)"  
        Case DATA_8_BIT  
            UserDataFormat.Caption = "Data (8-bit)"  
        Case UNICODE_16_BIT  
            UserDataFormat.Caption = "Unicode (16-bit)"  
    End Select  
End Sub
```

5.4.1.20 **UserDataHeader (rw)**

With this property, it is possible to add user data header information in the short message.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Byte**

Syntax

```
[propget, helpstring("property UserDataHeader")]  
HRESULT UserDataHeader(  
    [in] long index,  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property UserDataHeader")]  
HRESULT UserDataHeader(  
    [in] long index,  
    [in] unsigned char newVal  
);
```

The **UserDataHeader** property has these parts:

Part	Description
<i>index</i>	This data structure is filled byte by byte. Use loop and running index.

Remarks

See GSM 03.40 for more detailed information.

5.4.1.21 UserDataHeaderLength (rw)

This property tells the length of the **UserDataHeader** property in bytes.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Long**

Default value: 0

Syntax

```
[propget, helpstring("property UserDataHeaderLength")]  
HRESULT UserDataHeaderLength(  
    [out, retval] long *pVal  
);
```

```
[propput, helpstring("property UserDataHeaderLength")]  
HRESULT UserDataHeaderLength(  
    [in] long newVal  
);
```

Remarks

See GSM 03.40 for more detailed information.

5.4.1.22 UserDataLength (rw)

This property tells the length of the **UserDataText** property in bytes.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Long**

Default value: 0

Syntax

```
[propget, helpstring("property UserDataLength")]  
HRESULT UserDataLength(  
    [out, retval] long *pVal  
);
```

```
[propput, helpstring("property UserDataLength")]  
HRESULT UserDataLength(  
    [in] long newVal  
);
```

Remarks

This property may be used, e.g., when allocating dynamic memory buffer to hold the data of received short message.

See GSM 03.40 for more detailed information.

5.4.1.23 UserDataText (rw)

This property contains the actual short message when the data is in the character format (i.e., either 7 bit or 16 bit coding format has been used).

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **String**

Default value: Empty string

Syntax

```
[propget, helpstring("property UserDataText")]  
HRESULT UserDataText(  
    [out, retval] BSTR *pVal  
);
```

```
[propput, helpstring("property UserDataText")]  
HRESULT UserDataText(  
    [in] BSTR newVal  
);
```

Remarks

The maximum length of the message is 160 characters.

Example

See **CreateShortMsg** (5.3.1.1)

.

5.4.1.24 **ValidityPeriodFormat (rw)**

This property contains the format used for presenting a short message validity period.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: VALIDITY_PERIOD_FORMAT

Default value: RELATIVE_FORMAT

Syntax

```
[propget, helpstring("property ValidityPeriodFormat")]  
HRESULT ValidityPeriodFormat(  
    [out, retval] VALIDITY_PERIOD_FORMAT *pVal  
);
```

```
[propput, helpstring("property ValidityPeriodFormat")]  
HRESULT ValidityPeriodFormat(  
    [in] VALIDITY_PERIOD_FORMAT newVal  
);
```

Example

The following example method sets the static frame name according to the validity period format and then calls the appropriate method to display the validity period itself.

```
Public Sub ShowSMSDetails(ByVal cSMS As SMS3ASuiteLib.ShortMessage)  
  
    'validity period format  
    Select Case cSMS.ValidityPeriodFormat  
  
        Case ABSOLUTE_FORMAT  
            VPFrame.Caption = "Absolute Validity Period"  
            Call VP_ShowAbsolute  
  
        Case RELATIVE_FORMAT  
            VPFrame.Caption = "Relative Validity Period"  
            Call VP_ShowRelative  
  
        Case NOT_PRESENT  
            VPFrame.Caption = "Validity Period"  
            VP.Caption = "Validity period information not present."  
  
    End Select  
  
End Sub
```


5.4.1.25 **ValidityPeriodRelative (rw)**

This property contains the relative time when the short message goes out of date and will be erased from the SMSC.

Member of SMS3ASuiteLib.ShortMessage.IShortMessage

Type: **Byte**

Default value: 255 (maximum time)

Syntax

```
[propget, helpstring("property ValidityPeriodRelative")]  
HRESULT ValidityPeriodRelative(  
    [out, retval] unsigned char *pVal  
);
```

```
[propput, helpstring("property ValidityPeriodRelative")]  
HRESULT ValidityPeriodRelative(  
    [in] unsigned char newVal  
);
```

Remarks

The valid range for ValidityPeriodRelative value is 0–255. See GSM 03.40 for more detailed information.

Example

The following method displays the short message validity period expressed in relative format. The validity period constants defined below apply to Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 mobile phones.

```
'SMS validity periods  
Const SMS_MAX_TIME = 255  
Const SMS_ONE_WEEK = 173  
Const SMS_THREE_DAYS = 169  
Const SMS_24_HOURS = 167  
Const SMS_SIX_HOURS = 71  
Const SMS_ONE_HOUR = 11  
  
Private Sub VP_ShowRelative()  
  
    'show relative validity period  
    Select Case prShortMessage.ValidityPeriodRelative  
  
        Case SMS_MAX_TIME  
            VP.Caption = "Maximum time"  
        Case SMS_ONE_WEEK  
            VP.Caption = "One week"  
        Case SMS_THREE_DAYS  
            VP.Caption = "Three days"
```

```
        Case SMS_24_HOURS
            VP.Caption = "24 hours"
        Case SMS_SIX_HOURS
            VP.Caption = "Six hours"
        Case SMS_ONE_HOUR
            VP.Caption = "One hour"

    End Select
End Sub
```

5.5 GMSPICTURE Component

GMSPicture component is a data component containing one GMS message. Its properties are accessible through interface IGMSPicture. This component is intended to be used with the **SaveGMS()** method.

Properties of IGMSPicture can be divided into two parts. SMS message related properties and graphical picture data properties. SMS properties (**Msg....**) have proper default values and usually there is no need to change these. SMS related parameters are not described here but in case of any change, an explanation can be found from ShortMessage section.

5.5.1 IGMSPicture Interface

Property	Description
MsgRejectDuplicates	Initialized by default value: VARIANT_TRUE
MsgReplyPath	Initialized by default value: VARIANT_FALSE
MsgStatusReportRequest	Initialized by default value: VARIANT_FALSE
MsgOriginatorAddress	Initialized by default value: empty string
MsgProtocolID	Initialized by default value: zero
MsgDCS	Initialized by default value: zero
MsgValidityPeriodFormat	Initialized by default value: NOT_PRESENT
MsgValidityPeriod	Initialized by default value: empty string
MsgText	Initialized by default value: empty string
PicInfoField	Additional information: see Smart Messaging Specification, Graphical Logos and Icons section, description of InfoField.
PicDepth	Number of colors or gray shades.
PicWidth	Horizontal width of the bitmap in pixels.
PicHeight	Vertical height of the bitmap in pixels.
PicDataLength	Byte count of PicData.
PicData	OTA-bitmap bytes.

5.5.1.1 PicInfoField (rw)

Contains additional information about palette, compression etc. See Smart Messaging Specification, Graphical Logos and Icons section, description of *InfoField*.

Member of SMS3ASuiteLib.GMSPicture.IGMSPicture

Type: **Short**

Syntax

```
[propget, helpstring("property PicInfoField")]  
HRESULT PicInfoField(  
    [out, retval] short *pVal);
```

```
[propput, helpstring("property PicInfoField")]  
HRESULT PicInfoField(  
    [in] short newVal);
```

5.5.1.2 PicDepth (rw)

Number of colors or gray shades. See Smart Messaging Specification, Graphical Logos and Icons section, description of *Depth*.

Member of SMS3ASuiteLib.GMSPicture.IGMSPicture

Type: **Short**

Syntax

```
[propget, helpstring("property PicDepth")]  
HRESULT PicDepth(  
    [out, retval] short *pVal);  
[propput, helpstring("property PicDepth")]  
HRESULT PicDepth(  
    [in] short newVal);
```

5.5.1.3 PicWidth (rw)

Horizontal width of the bitmap in pixels. See Smart Messaging Specification, Graphical Logos and Icons section, description of *Width*.

Member of SMS3ASuiteLib.GMSPicture.IGMSPicture

Type: **Short**

Syntax

```
[propget, helpstring("property PicWidth")]  
HRESULT PicWidth(  
    [out, retval] short *pVal);  
[propput, helpstring("property PicWidth")]  
HRESULT PicWidth(  
    [in] short newVal);
```

5.5.1.4 PicHeight (rw)

Vertical height of the bitmap in pixels. See Smart Messaging Specification, Graphical Logos and Icons section, description of *Height*.

Member of SMS3ASuiteLib.GMSPicture.IGMSPicture

Type: **Short**

Syntax

```
[propget, helpstring("property PicHeight")]  
HRESULT PicHeight(  
    [out, retval] short *pVal);  
[propput, helpstring("property PicHeight")]  
HRESULT PicHeight(  
    [in] short newVal);
```


5.5.1.5 PicDataLength (rw)

The number of OTA-bytes included this picture.

Member of SMS3ASuiteLib.GMSPicture.IGMSPicture

Type: **Short**

Syntax

```
[propget, helpstring("property PicDataLength")]  
HRESULT PicDataLength(  
    [out, retval] short *pVal);
```

```
[propput, helpstring("property PicDataLength")]  
HRESULT PicDataLength(  
    [in] short newVal);
```

5.5.1.6 PicData (rw)

OTA-bytes of picture.

Member of SMS3ASuiteLib.GMSPicture.IGMSPicture

Type: **Short**

Syntax

```
[propget, helpstring("property PicData")]  
HRESULT PicData(  
    [in] short index,  
    [out, retval] short *pVal);
```

```
[propput, helpstring("property PicData")]  
HRESULT PicData(  
    [in] short index,  
    [in] short newVal);
```

Remarks

The **PicData** property has these parts:

Part	Description
<i>index</i>	This data structure is filled byte by byte. Use loop and running index. Valid range of values is from 0 to PicDataLength-1.

5.6 Enumerated DATA Types

Type	Description
REPORT_QUALIFIER	Contains types of status report messages.
SMS_MEMORY_LOCATION	Contains SMS memory types.
SMS_MESSAGE_TYPE	Contains different message types of SMS.
STORAGE_STATUS	Contains different SMS storage status identifiers.
USER_DATA_FORMAT	Contains the format used when coding the short message data.
VALIDITY_PERIOD_FORMAT	Contains short message validity period format types.

5.6.1 REPORT_QUALIFIER

Contains types of status report messages.

Member of SMS3ASuiteLib

The following table summarizes the possible status report types:

Type	Description
COMMAND_REQUESTED	The status report is the result of the command message.
SUBMIT_REQUESTED	The status report is the result of the submit message.

5.6.2 SMS_MEMORY_LOCATION

Contains SMS memory types.

Member of SMS3ASuiteLib

The following table summarizes the possible SMS memory types:

Type	Description
DEFAULT_MEMORY	Default SMS memory. Uses the memory type selected in the phone.
PHONE_MEMORY	SMS memory in the phone.
SIM_MEMORY	SMS memory on the SIM card.

5.6.3 SMS_MESSAGE_TYPE

Contains different message types of SMS.

Member of SMS3ASuiteLib

The following table summarizes the possible SMS types:

Type	Description
DELIVER_MESSAGE	Normal incoming SMS.
STATUS_REPORT_MESSAGE	The status of the sent message.
SUBMIT_MESSAGE	Normal outgoing SMS.
COMMAND_MESSAGE	Command message.
MO_UNDEFINED_MESSAGE	Undefined sent (mobile originated) message.
MT_UNDEFINED_MESSAGE	Undefined received (mobile terminated) message.

5.6.4 STORAGE_STATUS

Contains different SMS storage status identifiers.

Member of SMS3ASuiteLib

The following table summarizes the possible SMS storage statuses:

Type	Description
NOT_SENT_FROM_PHONE	The short message is stored in the SMS memory but has not been sent yet.
SENT_FROM_PHONE	The short message has been stored in the SMS memory and sent.
DELIVERED	The short message has been delivered to the recipient.
READ_FROM_PHONE	The received short message has been read.
NOT_READ_FROM_PHONE	The short message has been received but not read yet.

5.6.5 USER_DATA_FORMAT

Contains the format used when coding the short message data.

Member of SMS3ASuiteLib

The following table summarizes the possible data formats:

Type	Description
DATA_CODING_SCHEME_BASED	Data coding scheme is used to express the coding of the user data.
DEFAULT_ALPHABET_7_BIT	7 bit coding is used.
DATA_8_BIT	8 bit coding is used.
UNICODE_16_BIT	16 bit coding is used.

5.6.6 VALIDITY_PERIOD_FORMAT

Contains short message validity period format types.

Member of SMS3ASuiteLib

The following table summarizes the possible format types:

Type	Description
NOT_PRESENT	Reserved for future use.
RELATIVE_FORMAT	The validity period is presented in relation to the time when the short message was delivered to SMSC.
ENHANCED_FORMAT	Reserved for future use.
ABSOLUTE_FORMAT	The validity period is presented using absolute date and time when the short message will go out of date.

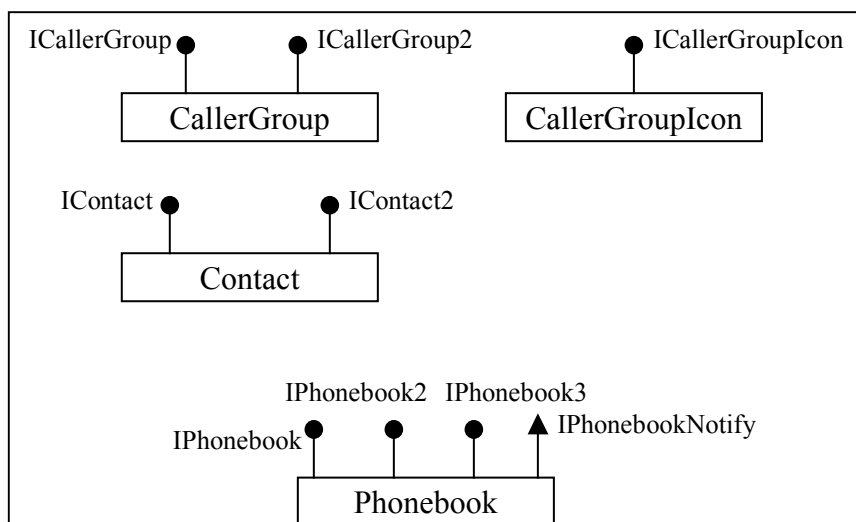
6. PHONEBOOK MEMORY LIBRARY (PHONEBOOKADAPTERDS3)

6.1 Overview

The Phonebook Library (PhonebookAdapterDS3) contains components for handling contacts and caller groups. With this library, it is possible to create applications that can handle contacts and caller groups for Nokia 51xx, 61xx, 62xx, 71xx, 8210, 8810, 8850 or 8890 phones.

6.1.1 Object Model

The picture below represents SCM3AS.dll module. The module contains implementation for four components: Phonebook, Contact, CallerGroup and CallerGroupIcon. Components are described in more detail in the next chapters.



Picture 3. Components and interfaces of the Phonebook module.

Only the Phonebook component can be instantiated normally using, e.g., CoCreateInstance API function. Contact and CallerGroup components can only be instantiated using the Phonebook method. The CallerGroupIcon can be instantiated using the CallerGroup method.

Some interfaces can be used only with specific phone models. In general, the IContact2, ICallerGroup2 and IPhonebook3 interfaces operate with 62xx and 71xx series phones, while the default interfaces operate with Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones. See interface documentation later in this document for details.

6.2 Phonebook Component

This component (PhonebookSuite3) contains three incoming interfaces: IPhonebook, IPhonebook2 and IPhonebook3. It also defines an outgoing (notify) interface IPhonebookNotify.

IPhonebook and IPhonebook2 interfaces are for Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones. IPhonebook3 and IPhonebookNotify are for Nokia 62xx and 71xx series phones.

6.2.1 IPhonebook Interface

This interface provides phonebook information and contact and caller group objects for Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones.

Method	Description
<hr/>	
CreateContact	Create empty contact object.
GetContact	Get phonebook entry from specified memory & location.
SetContact	Store contact in specified memory & location.
DeleteContact	Delete phonebook entry from specified memory & location.
GetSpeedDial	Get speed dial assignment for specified key.
SetSpeedDial	Get speed dial assignment of specified key.
GetMemoryInUse	Get default memory, where new contacts are added when inserted using phone UI.
SetMemoryInUse	Set default memory, where new contacts are added when inserted using phone UI.
GetSupportCaps	Get information about supported features: is phonebook editable, are caller groups supported.
GetMemoryCaps	Get information about specified memory: total, free and full amount of entries.
GetEntryCaps	Get the maximum lengths of contact name & number strings that fit in specified memory.
GetCallerGroup	Get caller group object containing the settings of specified caller group.
SetCallerGroup	Store caller group settings in the phone.
GetLastError	Get last error code.

6.2.1.1 CreateContact

Create empty contact object. When creating new contact, first create an empty contact object with this method, fill it, and finally store it in the phone using the SetContact method.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT CreateContact(  
    [out, retval] IContact ** ppContact  
);
```

The **CreateContact** method syntax has these parts:

Part	Description
<i>ppContact</i>	Output. Empty contact object.

6.2.1.2 GetContact

Get phonebook entry from specified memory and location.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetContact(  
    [in] PhonebookMemory memory,  
    [in] long location,  
    [out, retval] IContact ** ppContact  
);
```

The **GetContact** method syntax has these parts:

Part	Description
<i>memory</i>	Input. Memory from where entry is read.
<i>location</i>	Input. One-based location of entry.
<i>ppContact</i>	Output. Returned contact object.

Remarks

If requested contact was empty, error is returned. In such a case, GetLastError returns errPnEmpty.

Example

```
Sub DoSomething  
    Dim pContact As IContact  
    Set pContact = pIPhonebook.GetContact(MEMORY_SIM, 1)  
    ' do something with pContact...  
    Set pContact = Nothing  
End Sub
```

6.2.1.3 SetContact

Store contact on specified memory and location.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT SetContact(  
    [in] long location,  
    [in] IContact * pContact  
);
```

The **SetContact** method syntax has these parts:

Part	Description
<i>location</i>	Input. One-based location of entry.
<i>pContact</i>	Input. Contact object to store.

Remarks

Memory where contact is stored is as specified in IContact::Memory. Name and number string lengths should be adjusted based on information retrieved using method GetEntryCaps.

6.2.1.4 DeleteContact

Delete phonebook entry from specified memory and location.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT DeleteContact(  
    [in] PhonebookMemory memory,  
    [in] long location  
);
```

The **DeleteContact** method syntax has these parts:

Part	Description
<i>memory</i>	Input. Memory where to delete entry.
<i>location</i>	Input. Location of entry to delete.

Remarks

Entry is deleted by writing empty contact on it.

6.2.1.5 GetSpeedDial

Get speed dial assignment for specified key.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetSpeedDial(  
    [in] long key,  
    [in, out] PhonebookMemory * pMemory,  
    [in, out] long * pLocation  
);
```

The **GetSpeedDial** method syntax has these parts:

Part	Description
<i>key</i>	Input. Phone keypad key, range is 2–9.
<i>pMemory</i>	Input/output. Memory of assigned contact.
<i>pLocation</i>	Input/output. Location of assigned contact.

6.2.1.6 SetSpeedDial

Set speed dial assignment of specified key.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT SetSpeedDial(  
    [in] long key,  
    [in] PhonebookMemory memory,  
    [in] long location  
);
```

The **SetSpeedDial** method syntax has these parts:

Part	Description
<i>key</i>	Input. Phone keypad key, range is 2–9.
<i>memory</i>	Input. Memory of assigned contact.
<i>location</i>	Input. Location of assigned contact.

6.2.1.7 GetMemoryInUse

Get default memory where new contacts are added when inserted using phone UI.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetMemoryInUse(  
    [in, out] PhonebookMemory * pMemory  
);
```

The **GetMemoryInUse** method syntax has these parts:

Part	Description
<i>pMemory</i>	Input/output. Memory in use (MEMORY_ME or MEMORY_SIM).

6.2.1.8 SetMemoryInUse

Set default memory where new contacts are added when inserted using the phone UI.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT SetMemoryInUse(  
    [in] PhonebookMemory memory  
);
```

The **SetMemoryInUse** method syntax has these parts:

Part	Description
<i>memory</i>	Input. Memory in use (MEMORY_ME or MEMORY_SIM).

6.2.1.9 GetSupportCaps

Get information about supported features: is phonebook editable, are caller groups supported.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetSupportCaps(  
    [in, out] VARIANT_BOOL * pPhonebookEditable,  
    [in, out] VARIANT_BOOL * pCallergroupsSupported,  
    [in, out] VARIANT_BOOL * pSpeedDialAccess  
);
```

The **GetSupportCaps** method syntax has these parts:

Part	Description
<i>pPhonebookEditable</i>	Input/output. VARIANT_TRUE if phonebook is editable.
<i>pCallergroupsSupported</i>	Input/output. VARIANT_TRUE if caller groups are supported.
<i>pSpeedDialAccess</i>	Input/output. VARIANT_TRUE if speed dial settings are accessible.

6.2.1.10 GetMemoryCaps

Get information about specified memory: total, free and full amount of entries.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetMemoryCaps(  
    [in] PhonebookMemory memory,  
    [in, out] long * pTotal,  
    [in, out] long * pFull,  
    [in, out] long * pFree  
);
```

The **GetMemoryCaps** method syntax has these parts:

Part	Description
<i>memory</i>	Input. Memory whose capabilities are queried.
<i>pTotal</i>	Input/output. Total amount of entries (free + full).
<i>pFull</i>	Input/output. Amount of full entries.
<i>pFree</i>	Input/output. Amount of free entries.

Remarks

Full entries are not necessarily in consecutive order.

6.2.1.11 GetEntryCaps

Get the maximum lengths of contact name & number strings that fit in specified memory.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetEntryCaps(  
    [in] PhonebookMemory memory,  
    [in, out] long * pMaxNameLength,  
    [in, out] long * pMaxNumberLength  
);
```

The **GetEntryCaps** method syntax has these parts:

Part	Description
<i>memory</i>	Input. Memory to query.
<i>pMaxNameLength</i>	Input/output. Maximum length of name.
<i>pMaxNumberLength</i>	Input/output. Maximum length of number.

Remarks

This method may take a while, so it is recommended to call this once and store values for any later use.

6.2.1.12 **GetCallerGroup**

Get caller group object containing the settings of specified caller group.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetCallerGroup(  
    [in] long group,  
    [out, retval] ICallerGroup ** ppCallerGroup  
);
```

The **GetCallerGroup** method syntax has these parts:

Part	Description
<i>group</i>	Input. Caller group Id number.
<i>ppCallerGroup</i>	Output. Caller group object.

6.2.1.13 **SetCallerGroup**

Store caller group settings in the phone.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT SetCallerGroup(  
    [in] ICallerGroup * pCallerGroup  
);
```

The **SetCallerGroup** method syntax has these parts:

Part	Description
<i>pCallerGroup</i>	Input. Caller group object.

6.2.1.14 GetLastError

Returns the latest error code. If some method of interface IPhonebook returns a failure as its HRESULT value, this method can be used to retrieve a more specific error code.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook

Syntax

```
HRESULT GetLastError(  
    [out, retval] NmpAdapterError * pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	Output. Error code of type NmpAdapterError.

Remarks

This method should be called right after a method returns failure as its HRESULT value. If some other method is called, the code returned by this method may be erratic.

Example

```
Sub DoSomething  
On Error GoTo ErrorTrap  
    Dim pContact As IContact  
    Set pContact = pIPhonebook.CreateContact  
    ' do something with pContact...  
    Exit Sub  
ErrorTrap:  
    MsgBox "Error code:" & Str( pIPhonebook.GetLastError )  
End Sub
```

6.2.2 IPhonebook2 Interface

This interface extends the IPhonebook-interface by providing device identification information (such as IMEI-code etc).

Method	Description
GetDevIdentificationInfo	Get identification information of the device.
EnterSecurityCode	Enter security code.

6.2.2.1 GetDevIdentificationInfo

Get identification information of the device.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook2

Syntax

```
HRESULT GetDevIdentificationInfo(  
    [out] BSTR * ppsIMEI,  
    [out] BSTR * ppsType,  
    [out] BSTR * ppsVerSW,  
    [out] BSTR * ppsVerHW  
);
```

The **GetDevIdentificationInfo** method syntax has these parts:

Part	Description
<i>ppsIMEI</i>	Output. Phone IMEI code.
<i>ppsType</i>	Output. Phone type.
<i>ppsVerSW</i>	Output. Software version.
<i>ppsVerHW</i>	Output. Hardware version.

Example

```
Sub DoSomething  
    Dim sIMEI, sType, sVerSW, sVerHW  
  
    pIPhonebook2.GetDevIdentificationInfo sIMEI, sType, sVerSW, sVerHW  
  
    If sType = "NSE-3" Then  
        MsgBox "Nokia 6110"  
    End If  
End Sub
```


6.2.2.2 EnterSecurityCode

Enter security code. Use this method when an operation fails with code `errSecurityCodeRequired` (MPAPI_ERROR_SECURITY_CODE_REQUIRED, 143).

Member of `PhonebookAdapterDS3.PhonebookSuite3.IPhonebook2`

Syntax

```
HRESULT EnterSecurityCode(  
    [in] BSTR sCode  
);
```

The **EnterSecurityCode** method syntax has these parts:

Part	Description
<i>sCode</i>	Input. Security code.

Remarks

It is normal for an operation to fail because of a missing security code when the phone security level is "memory". If the user entered the wrong password (`errWrongPassword`), the application needs to ask for it again.

6.2.3 IPhonebook3 Interface

This interface provides phonebook information and contact and caller group objects for 62xx and 71xx series phones.

Method	Description
<hr/>	
CreateContact	Create empty contact object.
GetContact	Retrieve contact object from phone.
SetContact	Store contact object into phone.
DeleteContact	Delete contact from phone.
GetSpeedDial	Retrieve speed dial setting.
SetSpeedDial	Set speed dial setting.
GetMemoryInUse	Retrieve information about which memory is the default.
SetMemoryInUse	Select which memory is used by default.
GetMemoryCaps	Retrieve information about used and free memory.
GetCallerGroup	Retrieve caller group information.
SetCallerGroup	Set caller group information.
ResetCallCounters	Reset call counters.
GetLastError	Retrieve last error code.
IsMemoryAvailable	Retrieve information about memory availability.
StartListeningEvents	Start listening events through IPhonebookNotify.
StopListeningEvents	Stop listening events through IPhonebookNotify.

6.2.3.1 GetLastError

Returns the latest error code. If some method of interface IPhonebook3 returns a failure as its HRESULT value, this method can be used to retrieve a more specific error code.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT GetLastError(  
    [out, retval] NmpAdapterError * pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	Output. Error code of type NmpAdapterError.

Remarks

This method should be called right after a method returns failure as its HRESULT value. If some other method is called, the code returned by this method may be erratic.

Example

```
Sub DoSomething  
On Error GoTo ErrorTrap  
    pIPhonebook3.SetMemoryInUse PHYS_MEM_SIM  
    Exit Sub  
ErrorTrap:  
    MsgBox "Error code:" & Str( pIPhonebook3.GetLastError )  
End Sub
```

6.2.3.2 GetContact

Get phonebook entry from specified memory and location.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT GetContact(  
    [in] SearchMode searchMode,  
    [in] SearchKey searchKey,  
    [in] PhysicalMem PhysicalMemory,  
    [in] LogicalMem LogicalMemory,  
    [in, out] long *location,  
    [in] BSTR searchString,  
    [out, retval] IContact2 **ppContact  
);
```

The **GetContact** method syntax has these parts:

Part	Description
<i>searchMode</i>	Input. Search mode used: absolute, first, next, previous.
<i>searchKey</i>	Input. Search key used: location, name, number.
<i>PhysicalMemory</i>	Input. Physical memory from where entry is read (SIM card, phone, external memory etc.).
<i>LogicalMemory</i>	Input. Logical memory from where entry is read (phone directory, call stacks, voice mailbox etc.).
<i>location</i>	Input/output. Location of entry, starting from one. If name or number based search is used, return indicates the location where contact was found.
<i>searchString</i>	Input. String containing searched name or number. Must be an empty string if location based search is used.
<i>ppContact</i>	Output. Returned contact object.

Remarks

If requested contact was empty, error is returned and GetLastError returns errPnEmpty.

Example

```
` Read phone directory contents  
Private Sub ReadPhoneContacts()  
    Dim pIPnd3 As IPhonebook3  
    Dim pIContact2 As IContact2
```

```
Dim location As Long
Dim pMaxAmount As Long, pFull As Long, pFractation As Long

' query IPhonebook3-interface (g_pPnd is Phonebook-object)
Set pIPnd3 = g_pPnd

' when search mode is MODE_FIRST, location can be zero
location = 0

' get the amount of contacts
pIPnd3.GetMemoryCaps PHYS_MEM_PHONE, LOG_MEM_PD, _
    pMaxAmount, pFull, pFractation

' read first contact
Set pIContact2 = pIPnd3.GetContact(MODE_FIRST, KEY_LOCATION, _
    PHYS_MEM_PHONE, LOG_MEM_PD, location, "")
ShowContact pIContact2
Set pIContact2 = Nothing

' read the rest using MODE_NEXT search mode
For x = 1 To pFull - 1 Step 1
    Set pIContact2 = pIPnd3.GetContact(MODE_NEXT, KEY_LOCATION, _
        PHYS_MEM_PHONE, LOG_MEM_PD, location, "")
    ShowContact pIContact2
    Set pIContact2 = Nothing
Next x

Set pIPnd3 = Nothing
Exit Sub
End Sub

' Display contact name & number
Private Sub ShowContact(pContact As IContact2)
Dim sName As String, sNumber As String

pContact.FieldIndex = 1
If pContact.ValueType = CONTACT_NAME Then
    sName = pContact.StringValue
End If

pContact.FieldIndex = 2
If pContact.ValueType = CONTACT_PHONE_NUMBER Then
    sNumber = pContact.StringValue
End If

MsgBox sName & " " & sNumber, vbOKOnly, _
    "Contact in " & Str(pContact.location)
End Sub
```

6.2.3.3 SetContact

Stores contact in specified memory and location.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT SetContact(  
    [in, out] long * location,  
    [in] IContact2 * pContact  
);
```

The **SetContact** method syntax has these parts:

Part	Description
<i>location</i>	Input/output. Location index where to store the contact. Value 0 indicates that any location is acceptable.
<i>pContact</i>	Input. Contact to store.

Remarks

Memory where contact is stored is specified in IContact2::LogicalMemory and IContact2::PhysicalMemory. Name and number string lengths should not exceed the capability of target memory/location.

Example

```
` Store contact into any free location on SIM  
Sub DoSomething  
    Set pIContact2 = pIPhonebook3.CreateContact  
    pIContact2.LogicalMemory = LOG_MEM_PD  
    pIContact2.PhysicalMemory = PHYS_MEM_PHONE  
    pIContact2.AddField CONTACT_NAME, "My Name", 0  
    pIContact2.AddField CONTACT_PHONE_NUMBER, "+358505554477", 0  
    pIPhonebook3.SetContact 0, pIContact2  
    Set pIContact2 = Nothing  
End Sub
```

6.2.3.4 DeleteContact

Delete phonebook entry from specified location and memory.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT DeleteContact(  
    [in] PhysicalMem PhysicalMemory,  
    [in] LogicalMem LogicalMemory,  
    [in] long location  
);
```

The **DeleteContact** method syntax has these parts:

Part	Description
<i>PhysicalMemory</i>	Input. Physical memory from where entry is deleted.
<i>LogicalMemory</i>	Input. Logical memory from where entry is deleted.
<i>location</i>	Input. Location of entry to delete.

Remarks

Entry is deleted by writing empty contact on it.

Example

```
Sub DoSomething  
    pIPhonebook3.DeleteContact PHYS_MEM_SIM, LOG_MEM_PD, 23  
End Sub
```

6.2.3.5 GetSpeedDial

Get speed dial assignment for specified key.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT GetSpeedDial(  
    [in] long key,  
    [in, out] PhysicalMem *PhysicalMemory,  
    [in, out] LogicalMem *LogicalMemory,  
    [in, out] long * pLocation  
);
```

The **GetSpeedDial** method syntax has these parts:

Part	Description
<i>key</i>	Input. Phone keypad key, range 2–9.
<i>PhysicalMemory</i>	Input/output. Physical memory of assigned contact.
<i>LogicalMemory</i>	Input/output. Logical memory of assigned contact.
<i>pLocation</i>	Input/output. Location of assigned contact.

Example

```
` Display the location of contact assigned to keypad key 3  
Sub DoSomething  
    Dim pMem As PhysicalMem  
    Dim lMem As LogicalMem  
    Dim loc As Integer  
    pIPhonebook3.GetSpeedDial 3, pMem, lMem, loc  
    MsgBox "Location:" & Str(loc)  
End Sub
```


6.2.3.6 SetSpeedDial

Set speed dial assignment of specified key.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT SetSpeedDial(
    [in] long key,
    [in] PhysicalMem PhysicalMemory,
    [in] LogicalMem LogicalMemory,
    [in] long location
);
```

The **SetSpeedDial** method syntax has these parts:

Part	Description
<i>key</i>	Input. Phone keypad key, range is 2–9.
<i>PhysicalMemory</i>	Input. Physical memory of assigned contact.
<i>LogicalMemory</i>	Input. Logical memory of assigned contact.
<i>location</i>	Input. Location of assigned contact.

Remarks

In older phone software versions this method writes the number to the right location but the only way to get the number with the phone UI is to write n# where n is the number of the speed key.

Example

```
` Assigns keypad key 3 to contact in location 23
Sub DoSomething
    pIPhonebook3.SetSpeedDial 3, PHYS_MEM_SIM, _
                                LOG_MEM_PD, 23
End Sub
```

6.2.3.7 GetMemoryInUse

Get default memory where new contacts are added when inserted using the phone UI.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT GetMemoryInUse(  
    [out] PhysicalMem *PhysicalMemory  
);
```

The **GetMemoryInUse** method syntax has these parts:

Part	Description
<i>PhysicalMemory</i>	Output. Physical memory in use.

Example

```
Sub DoSomething  
    Dim pMem As PhysicalMem  
    pIPhonebook3.GetMemoryInUse pMem  
    If pMem = PHYS_MEM_SIM Then  
        MsgBox "SIM memory in use"  
    End If  
End Sub
```

6.2.3.8 SetMemoryInUse

Set default memory, where new contacts are added when inserted using the phone UI.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT SetMemoryInUse(  
    [in] PhysicalMem PhysicalMemory  
);
```

The **SetMemoryInUse** method syntax has these parts:

Part	Description
<i>PhysicalMemory</i>	Input. Physical memory in use.

Example

```
Sub DoSomething  
    pIPhonebook3.SetMemoryInUse PHYS_MEM_SIM  
End Sub
```

6.2.3.9 GetMemoryCaps

Get information about specified memory: total and full amount of entries.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT GetMemoryCaps(  
    [in] PhysicalMem PhysicalMemory,  
    [in] LogicalMem LogicalMemory,  
    [in, out] long * pMaxAmount,  
    [in, out] long * pFull,  
    [in, out] long * pFractation  
);
```

The **GetMemoryCaps** method syntax has these parts:

Part	Description
<i>PhysicalMemory</i>	Input. Physical memory whose capabilities are queried.
<i>LogicalMemory</i>	Input. Logical memory whose capabilities are queried.
<i>pMaxAmount</i>	Input/output. Maximum amount of entries (free + full).
<i>pFull</i>	Input/output. Amount of full entries.
<i>pFractation</i>	Input/output. Fractation level of used memory (0[empty] – 100[full]).

Remarks

Full entries are not necessarily in consecutive order.

Example

```
` Display the amount of free entries on SIM  
Sub DoSomething  
    Dim nMax, nFull, nFrac  
    pIPhonebook3.GetMemoryCaps PHYS_MEM_SIM, LOG_MEM_DIRECT_NBRs, _  
        nMax, nFull, nFrac  
    MsgBox "Free entries:" & Str(nMax - nFull)  
End Sub
```

6.2.3.10 GetCallerGroup

Get caller group object containing the settings of specified caller group.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT GetCallerGroup(  
    [in] long group,  
    [out, retval] ICallerGroup2 ** ppCallerGroup  
);
```

The **GetCallerGroup** method syntax has these parts:

Part	Description
<i>group</i>	Input. Caller group Id number.
<i>ppCallerGroup</i>	Output. Caller group object.

Example

```
Sub DoSomething  
    Dim pIcg2 As ICallerGroup2  
    Set pIcg2 = pIPhonebook3.GetCallerGroup(1)  
    MsgBox "Caller group name:" & pIcg2.Name  
End Sub
```

6.2.3.11 SetCallerGroup

Store caller group settings in the phone.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT SetCallerGroup(  
    [in] ICallerGroup2 * pCallerGroup  
);
```

The **SetCallerGroup** method syntax has these parts:

Part	Description
<i>pCallerGroup</i>	Input. Caller group object.

Example

```
Sub DoSomething  
    Dim pIcg2 As ICallerGroup2  
    Set pIcg2 = pIPhonebook3.GetCallerGroup(1)  
    pIcg2.Name = "NextOfKin"  
    pIcg2.ToneID = 68  
    pIPhonebook3.SetCallerGroup pIcg2  
End Sub
```

6.2.3.12 CreateContact

Create empty contact object. When creating new contact, first create empty contact object with this method, fill it, and finally store it in the phone using SetContact method.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT CreateContact(
    [out, retval] IContact2 ** ppContact
);
```

The **CreateContact** method syntax has these parts:

Part	Description
<i>ppContact</i>	Output. Empty contact object.

Example

```
Sub DoSomething
    Dim pIContact2 As IContact2
    Set pIContact2 = pIPhonebook3.CreateContact()

    ' fill contact object
    pIContact2.FieldIndex = 1
    pIContact2.ValueType = CONTACT_NAME
    pIContact2.StringValue = "John Doe"
    pIContact2.FieldIndex = 2
    pIContact2.ValueType = CONTACT_PHONE_NUMBER
    pIContact2.StringValue = "+358515550000"
    pIContact2.LogicalMemory = LOG_MEM_PD
    pIContact2.PhysicalMemory = PHYS_MEM_PHONE

    ' store contact object to phone
    pIPhonebook3.SetContact 0, pIContact2
End Sub
```

6.2.3.13 IsMemoryAvailable

Checks if queried memory is available.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT IsMemoryAvailable(  
    [in] LogicalMem LogicalMemory,  
    [out, retval] VARIANT_BOOL *status  
);
```

The **IsMemoryAvailable** method syntax has these parts:

Part	Description
<i>LogicalMemory</i>	Input. Logical memory to be queried.
<i>status</i>	Output. Status of memory

Example

```
Sub DoSomething  
    Dim bMem As Boolean  
    bMem = pIPhonebook3.IsMemoryAvailable(LOG_MEM_DIRECT_NBRS)  
    If bMem = True Then  
        MsgBox "Direct number memory available"  
    End If  
End Sub
```


6.2.3.14 StartListeningEvents

After this method is called, the adapter starts to send events to the client.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

HRESULT StartListeningEvents();

Remarks

The client must register to listen for events in the normal manner in addition to calling this method.

6.2.3.15 StopListeningEvents

After this method is called, the adapter stops event sending.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT StopListeningEvents();
```

Remarks

The client must unregister from listening to events in the normal manner in addition to this method.

6.2.3.16 **ResetCallCounters**

Resets selected call counters.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT ResetCallCounters(  
    [in] VARIANT_BOOL ResetLCS,  
    [in] VARIANT_BOOL ResetMCS,  
    [in] VARIANT_BOOL ResetRCS  
);
```

The **ResetCallCounters** method syntax has these parts:

Part	Description
<i>ResetLCS</i>	Input. Resets Last Dialed Calls Stack if TRUE.
<i>ResetMCS</i>	Input. Resets Missed Calls Stack if TRUE.
<i>ResetRCS</i>	Input. Resets Received Calls Stack if TRUE.

6.2.4 IPhonebookNotify Interface

This notify interface is called by the PhonebookSuite3 component when it needs to relay some information to the client. The client implements this interface and registers to listen for events. The client also needs to call IPhonebook3::StartListeningEvents.

When the client wishes to stop receiving events, it must call IPhonebook3::StopListeningEvents in addition to unregistering.

This interface can only be used in conjunction with IPhonebook3 interface.

Method	Description
ContactAdded	Indicates that contact is added to memory.
ContactDeleted	Indicates that contact is deleted from memory.
ContactModified	Indicates that contact is modified.

6.2.4.1 ContactAdded

Indicates that contact is added to memory.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebookNotify

Syntax

```
HRESULT ContactAdded(
    [in] PhysicalMem physicalMemory,
    [in] LogicalMem logicalMemory,
    [in] long location
);
```

The **ContactAdded** method syntax has these parts:

Part	Description
<i>physicalMemory</i>	Input. Physical memory concerned.
<i>logicalMemory</i>	Input. Logical memory concerned.
<i>location</i>	Input. Contact location concerned.

Example

```
Sub Phonebook3_ContactAdded(ByVal PhysicalMem physicalMemory,
                           ByVal LogicalMem logicalMemory,
                           ByVal long location)
    Dim pIContact2 As IContact2
    Set pIContact2 = Phonebook3.GetContact(MODE_FIRST, KEY_LOCATION, _
                                           physicalMemory, logicalMemory, _
                                           location, "")
    pIContact2.FieldIndex = 1
    MsgBox "New contact:" & pIContact2.StringValue
End Sub
```

6.2.4.2 ContactDeleted

Indicates that contact was deleted from memory.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebookNotify

Syntax

```
HRESULT ContactDeleted(  
    [in] PhysicalMem physicalMemory,  
    [in] LogicalMem logicalMemory,  
    [in] long location  
);
```

The **ContactDeleted** method syntax has these parts:

Part	Description
<i>physicalMemory</i>	Input. Physical memory concerned.
<i>logicalMemory</i>	Input. Logical memory concerned.
<i>location</i>	Input. Contact location concerned.

Example

```
Sub Phonebook3_ContactDeleted(ByVal PhysicalMem physicalMemory,  
                              ByVal LogicalMem logicalMemory,  
                              ByVal long location)  
    MsgBox "Deleted contact from location:" & Str(location)  
End Sub
```

6.2.4.3 ContactModified

Indicates that contact is modified.

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebookNotify

Syntax

```
HRESULT ContactModified(  
    [in] PhysicalMem physicalMemory,  
    [in] LogicalMem logicalMemory,  
    [in] long location  
);
```

The **ContactModified** method syntax has these parts:

Part	Description
<i>physicalMemory</i>	Input. Physical memory concerned.
<i>logicalMemory</i>	Input. Logical memory concerned.
<i>location</i>	Input. Contact location concerned.

Example

```
Sub Phonebook3_ContactModified(ByVal PhysicalMem physicalMemory,  
                               ByVal LogicalMem logicalMemory,  
                               ByVal long location)  
    Dim pIContact2 As IContact2  
    Set pIContact2 = Phonebook3.GetContact(MODE_FIRST, KEY_LOCATION, _  
                                           physicalMemory, logicalMemory, _  
                                           location, "")  
    pIContact2.FieldIndex = 1  
    MsgBox "Modified contact:" & pIContact2.StringValue  
End Sub
```

6.3 Contact Component

Contact component contains information of one contact (name, number etc.). The IContact interface is for contacts for Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones, while the IContact2 interface is for contacts for 62xx and 71xx series phones.

6.3.1 IContact Interface

Automation compatible interface for client applications (Nokia 51xx, 61xx, 8210, 8810, 8850 or 8890). Provides data of one phonebook entry, i.e. "contact".

Property	Description
<hr/>	
Name	Name of contact.
Number	Phone number of contact.
GroupID	Caller group number of contact.
Memory	Memory of contact (phone or SIM).
Location	Location number of contact.
TimeStamp	Time stamp of contact.
IsValidTimeStamp	Identifies if TimeStamp property contains valid time stamp.
Tag	Data field for application to use.

Method	Description
<hr/>	
GetLastError	Returns last error code.

6.3.1.1 Name (rw)

Name of contact.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT Name(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT Name(  
    [in] BSTR newVal  
);
```

Remarks

Maximum name length depends on target memory.

6.3.1.2 Number (rw)

Phone number of contact.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT Number(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT Number(  
    [in] BSTR newVal  
);
```

Remarks

Maximum number length depends on target memory. String may contain illegal characters, such as spaces, but they are removed when storing a contact in the phone.

6.3.1.3 GroupID (rw)

Caller group number of contact. By assigning contacts into caller groups and setting each caller group a unique ringing tone, it is easy to identify an incoming call. This is especially important if CLI mask is used.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT GroupID(  
    [out, retval] long *pVal  
);
```

```
HRESULT GroupID(  
    [in] long newVal  
);
```

Remarks

Valid caller group numbers depend on the product. Not all products support caller groups.

6.3.1.4 Memory (rw)

Memory of contact.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT Memory(  
    [out, retval] PhonebookMemory *pVal  
);
```

```
HRESULT Memory(  
    [in] PhonebookMemory newVal  
);
```

Remarks

Not all memory types may be supported by a specific product.

6.3.1.5 Location (r)

Location number of contact. Read-only property. Maximum index depends on product and SIM card.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT Location(  
    [out, retval] long *pVal  
);
```

Remarks

This is ignored when storing a contact in the phone. The location is specified in a parameter of IPhonebook::SetContact.

6.3.1.6 TimeStamp (r)

Time stamp of contact. Read-only property. May be valid only if Memory is MEMORY_MISSED, MEMORY_RECEIVED or MEMORY_DIALED.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT TimeStamp(  
    [out, retval] DATE *pVal  
);
```

6.3.1.7 IsValidTimeStamp (r)

Identifies if TimeStamp property contains valid time stamp. Read-only property.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT IsValidTimeStamp(  
    [out, retval] VARIANT_BOOL *pVal  
);
```


6.3.1.8 Tag (rw)

Data field for application to use. Not used internally.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT Tag(  
    [out, retval] long *pVal  
);
```

```
HRESULT Tag(  
    [in] long newVal  
);
```

6.3.1.9 GetLastError

Returns the latest error code. If some method of interface IContact returns a failure as its HRESULT value, this method can be used to retrieve a more specific error code.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact

Syntax

```
HRESULT GetLastError(  
    [out, retval] NmpAdapterError * pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>PErr</i>	Output. Error code of type NmpAdapterError.

Remarks

This method should be called right after a method returns failure as its HRESULT value. If some other method is called, the code returned by this method may be erratic.

Example

```
Sub DoSomething  
On Error GoTo ErrorTrap  
    pIContact.Name = "John Doe"  
    Exit Sub  
ErrorTrap:  
    MsgBox "Error code:" & Str( pIContact.GetLastError )  
End Sub
```

6.3.2 IContact2 Interface

Automation-compatible interface for client applications (62xx and 71xx series phones). Provides data of one phonebook entry, i.e. "contact".

This new contact can hold several fields, for example several phone numbers etc. Usually the field with index one contains the name (of type CONTACT_NAME), and the one with index two contains the number (of type CONTACT_PHONE_NUMBER).

Property	Description
StringValue	String field's value.
NumberValue	Number field's value.
ValueType	Field's type.
AmountOfFields	Amount of fields in contact.
FieldIndex	Index of current field.
LogicalMemory	Logical memory of contact.
PhysicalMemory	Physical memory of contact.
Location	Location number of contact.
TimeStamp	Time stamp of contact.
IsValidTimeStamp	Identifies if TimeStamp-property contains valid time stamp.

Method	Description
GetLastError	Returns last error code.
AddField	Adds a new field to contact.
DeleteField	Removes field from contact.

6.3.2.1 StringValue (rw)

String field's value, empty string if not set.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT StringValue(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT StringValue(  
    [in] BSTR newVal  
);
```

6.3.2.2 NumberValue (rw)

Number field's value, zero if not set.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT NumberValue(  
    [out, retval] long *pVal  
);
```

```
HRESULT NumberValue(  
    [in] long newVal  
);
```

6.3.2.3 ValueType (rw)

The type of field. Default value CONTACT_UNKNOWN.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT ValueType(  
    [out, retval] FieldType *pVal  
);
```

```
HRESULT ValueType(  
    [in] FieldType newVal  
);
```

6.3.2.4 AmountOfFields (r)

Amount of fields in contact.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT AmountOfFields(  
    [out, retval] long *pVal  
);
```

6.3.2.5 FieldIndex (rw)

Index of current field, starting from one. If this value is changed, it affects the properties of ValueType, NumberValue and StringValue.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT FieldIndex(  
    [out, retval] long *pVal  
);
```

```
HRESULT FieldIndex(  
    [in] long newVal  
);
```


6.3.2.6 LogicalMemory (rw)

Logical memory of contact (personal directory, call stack, voice mailbox etc.). Normal contact information is stored in personal directory.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT LogicalMemory(  
    [out, retval] LogicalMem *pVal  
);
```

```
HRESULT LogicalMemory(  
    [in] LogicalMem newVal  
);
```

Remarks

Not all memory types may be supported by a specific product. Contacts are stored in LOG_MEM_PD –logical memory when physical memory is PHYS_MEM_PHONE.

Contacts on SIM card (physical memory PHYS_MEM_SIM) can be accessed only when logical memory is set to LOG_MEM_DEFAULT and SIM memory is in use (e.g., set through phone UI).

6.3.2.7 PhysicalMemory (rw)

Physical memory of contact (SIM card, phone memory or some external memory).

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT PhysicalMemory(  
    [out, retval] PhysicalMem *pVal  
);
```

```
HRESULT PhysicalMemory(  
    [in] PhysicalMem newVal  
);
```

Remarks

Not all memory types may be supported by a specific product. Applications mostly use PHYS_MEM_PHONE and PHYS_MEM_SIM.

6.3.2.8 Location (r)

Location number of contact, starting from one. Maximum index depends on product and SIM card.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT Location(  
    [out, retval] long *pVal  
);
```

Remarks

This is ignored when storing contact in the phone. Location is specified in a parameter of IPhonebook3::SetContact.

6.3.2.9 TimeStamp (r)

Time stamp of contact.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT TimeStamp(  
    [out, retval] DATE *pVal  
);
```

6.3.2.10 IsValidTimeStamp (r)

Identifies if TimeStamp property contains valid time stamp.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT IsValidTimeStamp(  
    [out, retval] VARIANT_BOOL *pVal  
);
```

6.3.2.11 GetLastError

Returns the latest error code. If some method of interface IContact2 returns a failure as its HRESULT value, this method can be used to retrieve a more specific error code.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT GetLastError(  
    [out, retval] NmpAdapterError * pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	Output. Error code of type NmpAdapterError.

Remarks

This method should be called right after a method returns failure as its HRESULT value. If some other method is called, the code returned by this method may be erratic.

Example

```
Sub DoSomething  
On Error GoTo ErrorTrap  
    pIContact2.FieldIndex = -23  
    Exit Sub  
ErrorTrap:  
    MsgBox "Error code:" & Str( pIContact2.GetLastError )  
End Sub
```

6.3.2.12 **AddField**

Adds a new field to contact.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT AddField(  
    [in] FieldType type,  
    [in] BSTR strValue,  
    [in] long nbrValue  
);
```

The **AddField** method syntax has these parts:

Part	Description
<i>type</i>	Input. Type of field.
<i>strValue</i>	Input. String value.
<i>nbrValue</i>	Input. Number value.

6.3.2.13 DeleteField

Removes field from contact.

Member of PhonebookAdapterDS3.PhonebookSuite3.IContact2

Syntax

```
HRESULT DeleteField(  
    [in] long index  
);
```

The **DeleteField** method syntax has these parts:

Part	Description
<i>index</i>	Input. Index of field to be removed.

6.4 CallerGroup Component

The CallerGroup-component contains the settings of one caller group. An application retrieves the caller group object through the interface method `IPhonebook::GetCallerGroup` or `IPhonebook3::GetCallerGroup`.

Contacts can be organized into caller groups. For example, by defining different ringing tones for each group, it is easy to instantly recognize what kind of a call is incoming.

6.4.1 ICallerGroup Interface

This interface is used to modify caller group settings for Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones.

Property	Description
GroupID	Caller group ID number.
Name	Caller group name.
ToneID	Caller group ringing tone.
IconActive	Specifies whether a caller group icon is displayed or not.
Icon	Caller group icon.
Tag	Application-specified value.

Method	Description
SetGroupID	Sets GroupID-property value (caller group ID number).
GetLastError	Returns last error code.

6.4.1.1 GroupID (r)

Caller group ID number.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup

Syntax

```
HRESULT GroupID(  
    [out, retval] long *pVal  
);
```

Remarks

Group ID value 255 stands for "no group".

6.4.1.2 Name (rw)

Caller group name.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup

Syntax

```
HRESULT Name(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT Name(  
    [in] BSTR newVal  
);
```

Remarks

To restore default name, set this to empty string.

6.4.1.3 ToneID (rw)

Caller group ringing tone.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup

Syntax

```
HRESULT ToneID(  
    [out, retval] long *pVal  
);
```

```
HRESULT ToneID(  
    [in] long newVal  
);
```

Remarks

Default value "preset" (255) means that the ringing tone specified by the profile setting is used. Available tones vary from one product to another.

6.4.1.4 IconActive (rw)

Specifies whether a caller group icon is displayed or not.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup

Syntax

```
HRESULT IconActive(  
    [out, retval] VARIANT_BOOL *pVal  
);
```

```
HRESULT IconActive(  
    [in] VARIANT_BOOL newVal  
);
```

6.4.1.5 Icon (rw)

Caller group icon.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup

Syntax

```
HRESULT Icon(  
    [out, retval] ICallerGroupIcon * *pVal  
);
```

```
HRESULT Icon(  
    [in] ICallerGroupIcon * newVal  
);
```

Remarks

This may be set to NULL, in which case icon is not modified.

6.4.1.6 Tag (rw)

Application-specified value. This value is not saved in the phone; the value exists only during the lifetime of the object.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup

Syntax

```
HRESULT Tag(  
    [out, retval] long *pVal  
);
```

```
HRESULT Tag(  
    [in] long newVal  
);
```


6.4.1.7 SetGroupID

Sets GroupID property value (caller group ID number).

Member of PhonebookAdapterDS3.PhonebookSuite3.IPhonebook3

Syntax

```
HRESULT SetGroupID(  
    [in] long group  
);
```

The **SetGroupID** method syntax has these parts:

Part	Description
<i>group</i>	Input. Caller group Id number.

Remarks

Application does not usually call this method.

6.4.1.8 GetLastError

Returns the latest error code. If some method of interface ICallerGroup returns a failure as its HRESULT value, this method can be used to retrieve a more specific error code.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup

Syntax

```
HRESULT GetLastError(  
    [out, retval] NmpAdapterError * pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	Output. Error code of type NmpAdapterError.

Remarks

This method should be called right after a method returns failure as its HRESULT value. If some other method is called, the code returned by this method may be erratic.

Example

```
Sub DoSomething  
On Error GoTo ErrorTrap  
    pICallerGroup.Name = "My Group"  
    Exit Sub  
ErrorTrap:  
    MsgBox "Error code:" & Str( pICallerGroup.GetLastError )  
End Sub
```

6.4.2 ICallerGroup2 Interface

This interface is used to modify caller group settings for 62xx and 71xx series phones.

Property	Description
GroupID	Caller group ID number.
Name	Caller group name.
ToneID	Caller group ringing tone.
IconActive	Specifies whether a caller group icon is displayed or not.
Icon	Caller group icon.
VibraID	Vibra ID number.

Method	Description
SetGroupID	Set GroupID property value (caller group ID number).
GetLastError	Returns last error code.

6.4.2.1 GroupID (r)

Caller group ID number.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT GroupID(  
    [out, retval] long *pVal  
);
```

Remarks

Group ID value 255 stands for "no group".

6.4.2.2 Name (rw)

Caller group name.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT Name(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT Name(  
    [in] BSTR newVal  
);
```

Remarks

To restore default name, set this to empty string.

6.4.2.3 ToneID (rw)

Caller group ringing tone.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT ToneID(  
    [out, retval] long *pVal  
);
```

```
HRESULT ToneID(  
    [in] long newVal  
);
```

Remarks

Default value "preset" (255) means that the ringing tone specified by the profile setting is used. Available tones vary from one product to another.

6.4.2.4 IconActive (rw)

Specifies whether a caller group icon is displayed or not.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT IconActive(  
    [out, retval] VARIANT_BOOL *pVal  
);
```

```
HRESULT IconActive(  
    [in] VARIANT_BOOL newVal  
);
```

6.4.2.5 Icon (rw)

Caller group icon.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT Icon(  
    [out, retval] ICallerGroupIcon * *pVal  
);
```

```
HRESULT Icon(  
    [in] ICallerGroupIcon * newVal  
);
```

Remarks

This may be set to NULL, in which case the icon is not modified.

6.4.2.6 VibraID (rw)

Vibra ID number.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT VibraID(  
    [out, retval] long *pVal  
);
```

```
HRESULT VibraID(  
    [in] long newVal  
);
```

6.4.2.7 SetGroupID

Sets GroupID property value (caller group ID number).

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT SetGroupID(  
    [in] long group  
);
```

The **SetGroupID** method syntax has these parts:

Part	Description
<i>group</i>	Input. Caller group Id number.

Remarks

Application does not usually call this method.

6.4.2.8 GetLastError

Returns the latest error code. If some method of interface ICallerGroup2 returns a failure as its HRESULT value, this method can be used to retrieve more specific error code.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroup2

Syntax

```
HRESULT GetLastError(  
    [out, retval] NmpAdapterError * pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	Output. Error code of type NmpAdapterError.

Remarks

This method should be called right after a method returns failure as its HRESULT value. If some other method is called, the code returned by this method may be erratic.

Example

```
Sub DoSomething  
On Error GoTo ErrorTrap  
    pICallerGroup2.VibraID = 74  
    Exit Sub  
ErrorTrap:  
    MsgBox "Error code:" & Str( pICallerGroup2.GetLastError )  
End Sub
```

6.5 CallerGroupIcon Component

Each caller group has an icon, which is displayed on the phone screen when an incoming call alert is active and the caller belongs to one of the caller groups.

Caller group icon is in OTA bitmap format. For more information see Smart Messaging Specification, Graphical Logos and Icons section.

6.5.1 ICallerGroupIcon Interface

Accesses caller group icon data. The icon can be handled with user-friendly properties: width, height and pixel. Alternatively, OTA bitmap is also accessible through methods GetOTAByte and SetOTAByte.

Header information must be set with method SetHeaderInfo before the properties, width, height or pixel, can be used.

Property	Description
Width	Width of caller group icon in pixels.
Height	Height of caller group icon in pixels.
Pixel	One caller group icon pixel.

Method	Description
GetOTAByte	Get one byte of icon data.
SetOTAByte	Set one byte of icon data.
GetOTAByteCount	Get the count of icon data bytes.
SetHeaderInfo	Set caller group icon header information.
GetLastError	Returns last error code.

6.5.1.1 Width (r)

Width of caller group icon in pixels.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT Width(  
    [out, retval] long *pVal  
);
```

Remarks

Header characteristics must be set with method SetHeaderInfo before this property is available.

6.5.1.2 Height (r)

Height of caller group icon in pixels.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT Height(  
    [out, retval] long *pVal  
);
```

Remarks

Header characteristics must be set with method SetHeaderInfo before this property is available.

6.5.1.3 Pixel (rw)

One caller group icon pixel.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT Pixel(  
    [in] long row,  
    [in] long col,  
    [out, retval] long *pVal  
);
```

```
HRESULT Pixel(  
    [in] long row,  
    [in] long col,  
    [in] long newVal  
);
```

Remarks

Pixel value of zero (0) is the background color and one (1) is the foreground color.

Header characteristics must be set with method SetHeaderInfo before this property is available. This property is not available if icon is either colored, compressed or animated.

6.5.1.4 GetOTAByte

Get one byte of icon data. Returned data is in raw OTA format.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT GetOTAByte(  
    [in] long index,  
    [out, retval] short * iconByte  
);
```

The **GetOTAByte** method syntax has these parts:

Part	Description
<i>index</i>	Input. Zero-based location index of data byte.
<i>iconByte</i>	Output. Icon data byte.

Remarks

OTA bitmap contains a header. The header bytes can be accessed with this method.

6.5.1.5 SetOTAByte

Set one byte of icon data. Data is in raw OTA format.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT SetOTAByte(  
    [in] long index,  
    [in] short iconByte  
);
```

The **SetOTAByte** method syntax has these parts:

Part	Description
<i>index</i>	Input. Zero-based location index of data byte.
<i>iconByte</i>	Input. Icon data byte.

Remarks

OTA bitmap contains a header. If caller group icon is first read from phone, client application should not set argument "index" greater than the value returned by GetOTAByteCount -1.

6.5.1.6 GetOTAByteCount

Get the count of icon data bytes. Returned count includes header bytes.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT GetOTAByteCount(  
    [out, retval] long * count  
);
```

The **GetOTAByteCount** method syntax has these parts:

Part	Description
<i>count</i>	Output. Amount of bytes in icon.

Remarks

Calling SetOTAByte will adjust this if needed. However, clients should not index over the reported count.

6.5.1.7 SetHeaderInfo

Set caller group icon header information. This must be called before the properties width, height or pixel can be used.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT SetHeaderInfo(  
    [in] long byteCount,  
    [in] VARIANT_BOOL extFieldPresent,  
    [in] long widthByteCount,  
    [in] long heightByteCount  
);
```

The **SetHeaderInfo** method syntax has these parts:

Part	Description
<i>byteCount</i>	Input. Amount of bytes in header.
<i>extFieldPresent</i>	Input. If True, header contains ext field.
<i>widthByteCount</i>	Input. Amount of bytes used to represent icon width.
<i>heightByteCount</i>	Input. Amount of bytes used to represent icon height.

Remarks

This method only tells component how to interpret the header. Call SetOTAByte to actually insert header bytes.

By default, the header info is set to "4, VARIANT_FALSE, 1, 1", which applies to Nokia 6110 GSM phone. Client must set header info if it differs from these default values.

6.5.1.8 GetLastError

Returns the latest error code. If some method of interface IPhonebook3 returns a failure as its HRESULT value, this method can be used to retrieve more specific error code.

Member of PhonebookAdapterDS3.PhonebookSuite3.ICallerGroupIcon

Syntax

```
HRESULT GetLastError(  
    [out, retval] NmpAdapterError * pErr  
);
```

The **GetLastError** method syntax has these parts:

Part	Description
<i>pErr</i>	Output. Error code of type NmpAdapterError.

Remarks

This method should be called right after a method returns failure as its HRESULT value. If some other method is called, the code returned by this method may be erratic.

Example

```
Sub DoSomething  
On Error GoTo ErrorTrap  
    pICallerGroupIcon.Pixel 23, 1098, 1  
    Exit Sub  
ErrorTrap:  
    MsgBox "Error code:" & Str( pICallerGroupIcon.GetLastError )  
End Sub
```

6.6 Enumerated DATA Types

This chapter lists enumeration types used by the components in SCM3AS module.

Type	Description
PhonebookMemory	Memory types of Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones.
FieldType	Contact field types.
SearchMode	Contact search modes.
SearchKey	Contact search keys.
LogicalMem	Logical memory types of 62xx and 71xx series phones.
PhysicalMem	Physical memory types of 62xx and 71xx series phones.

6.6.1.1 PhonebookMemory

Memory types of Nokia 51xx, 61xx, 8210, 8810, 8850 and 8890 phones.

Member of PhonebookAdapterDS3

The following table summarizes the possible memory types.

Value	Description
MEMORY_ME	Phone memory.
MEMORY_SIM	SIM card memory.
MEMORY_FDN	Fixed dialing numbers on SIM.
MEMORY_SDN	Service dialing numbers on SIM.
MEMORY_MSISDN	Own number list on SIM.
MEMORY_EMERGENCY	Emergency (location must be 0).
MEMORY_VOICEBOX	Voicebox on SIM (location must be 0).
MEMORY_DIALED	Dialed call stack.
MEMORY_RECEIVED	Received call stack.
MEMORY_MISSED	Missed call stack.
MEMORY_DEFAULT	Default memory (ME/SIM)

6.6.1.2 FieldType

Contact field types. Normal contact consists of two fields: CONTACT_NAME and CONTACT_PHONE_NUMBER.

Member of PhonebookAdapterDS3

The following table summarizes the possible field types.

Value	Description
CONTACT_UNKNOWN	Unknown Type.
CONTACT_NAME	Name.
CONTACT_ADDITIONAL_NAME	Additional name information.
CONTACT_NOTE	Note.
CONTACT_EMAIL_ADDRESS	E-Mail address.
CONTACT_POST_ADDRESS	Postal address.
CONTACT_VM_PASSWORD	Voice mail password.
CONTACT_STANDARD_NUMBER	Phone number.
CONTACT_HOME_NUMBER	Home phone number.
CONTACT_MOBILE_NUMBER	Mobile phone number
CONTACT_FAX_NUMBER	Fax number.
CONTACT_CAR_NUMBER	Car Phone number.
CONTACT_WORK_NUMBER	Work Phone number.
CONTACT_OFFICE_NUMBER	Office Phone number.
CONTACT_PAGER_NUMBER	Pager number.
CONTACT_VOICETAG_ID	Voice tag ID.
CONTACT_GROUP_ID	Group ID.

CONTACT_MODEM_NUMBER	Modem number.
CONTACT_PHONE_NUMBER	Phone number.
CONTACT_RING_TONE_ID	ID of record specific ringing tone.
CONTACT_CALL_COUNT	Number of calls related to record.
CONTACT_CALL_COST	Call costs related to record.
CONTACT_FORWARDING_INFO	Forwarding information.
CONTACT_VIBRA_ID	Vibration style.

6.6.1.3 SearchMode

Contact search mode.

Member of PhonebookAdapterDS3.

The following table summarizes the possible search mode types.

Value	Description
MODE_ABSOLUTE	Exact match.
MODE_FIRST	First matching.
MODE_LAST	Last matching.
MODE_NEXT	Next matching.
MODE_PREV	Previous matching.

6.6.1.4 SearchKey

Contact search key.

Member of PhonebookAdapterDS3

The following table summarizes the possible search key types.

Value	Description
KEY_LOCATION	Search by location.
KEY_NAME	Search by name.
KEY_NUMBER	Search by number.

6.6.1.5 LogicalMem

Logical memory types of 62xx and 71xx series phones. Logical memory consists of call stacks, voice mailbox etc. Normal contact information is stored in the personal directory (LOG_MEM_PD). Depending on physical memory, not all logical memory types may be available.

Member of PhonebookAdapterDS3

The following table summarizes the possible logical memory types.

Value	Description
LOG_MEM_UNKNOWN	Unknown memory.
LOG_MEM_LCS	Last dialed Calls Stack (LCS).
LOG_MEM_MCS	Missed Calls Stack (MCS).
LOG_MEM_RCS	Received Calls Stack (RCS).
LOG_MEM_NP	Notepad or Scratchpad (NP).
LOG_MEM_PD	Personal Directory on Phone (PD).
LOG_MEM_ADN	Abbreviated Dialing Number (ADN).
LOG_MEM_FDN	Fixed Dialing Number list (FDN).
LOG_MEM_OWN_NBR	Own Number.
LOG_MEM_VMBX	Voice Mailbox number.
LOG_MEM_TMP	Temporary memory
LOG_MEM_WAKE_UP_MSG	Wake-Up Message
LOG_MEM_SDN	Service Dialing Number list (SDN).
LOG_MEM_EMERGENCY	Emergency numbers.
LOG_MEM_SPEED_DIAL	Speed Dialing memory.
LOG_MEM_MOST_CALL	Most Called stack.

LOG_MEM_CLI_GRP_LST	List of CLI Groups and properties.
LOG_MEM_HEAD_SET	Head Set numbers list.
LOG_MEM_DIAL_LOCK	Memory Locked Dialing numbers list.
LOG_MEM_VMBX_SIM	SIM Voice Mailbox number.
LOG_MEM_DIRECT_NBRS	Direct Numbers memory.
LOG_MEM_DEFAULT	Default memory.
LOG_MEM_ALL	All logical Memories.

6.6.1.6 PhysicalMem

Physical memory types for 62xx and 71xx series phones. Physical memory is SIM, phone or some external memory.

Member of PhonebookAdapterDS3

The following table summarizes the possible physical memory types.

Value	Description
PHYS_MEM_UNKNOWN	Unknown memory.
PHYS_MEM_SIM	SIM memory.
PHYS_MEM_PHONE	Phone Memory.
PHYS_MEM_EXT1	External memory module 1.
PHYS_MEM_EXT2	External memory module 2.
PHYS_MEM_ANY	Any physical memory.
PHYS_MEM_ALL	All physical memories.

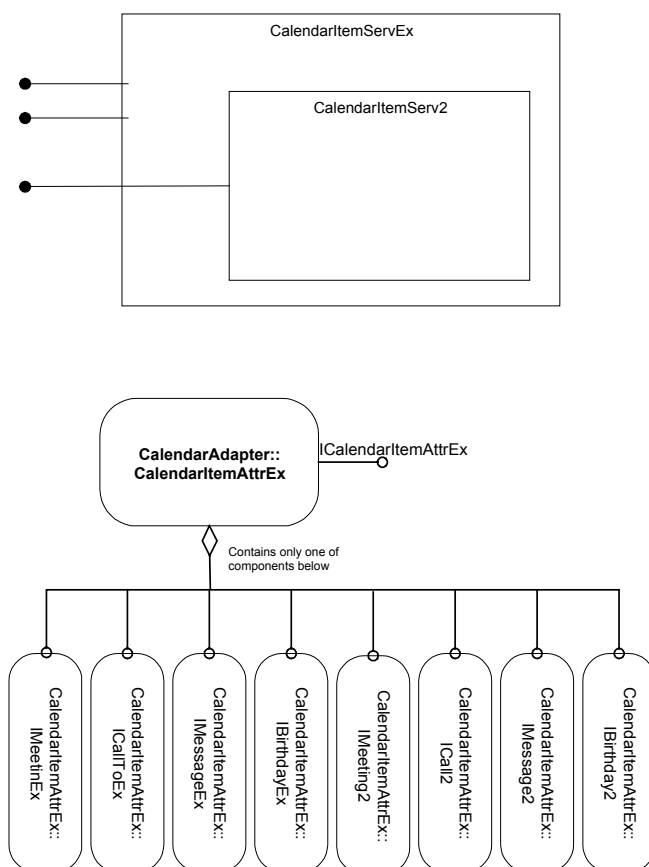
7. CALENDAR LIBRARY (CALADAPTERLIB)

7.1 Overview

The Calendar Library contains components for handling calendar events. Components contain interfaces for creating, writing, reading, and deleting calendar events of different types. Only the interfaces described in this document should be used. Some other interfaces might be visible in the components at design time, but they are not supported and should not be used. Before these components are used, the program must check if the phone supports the Calendar Library. This can be done with **IsCapability**, which is in the **IPhoneCapability** interface in the General Settings Library.

7.2 Object Model

The picture below represents Cal3aS.dll module. Objects are described in more details in the next chapters.



Picture 4. Components and interfaces of the Calendar module.

7.3 CALENDARITEMSERVEx Component

This component contains two incoming interfaces; ICalendarItemServEx and ICalendarItemServ2 and one outgoing (source) interface; IPhoneCalendarNotify2.

7.3.1 ICalendarItemServEx Interface

The ICalendarItemServEx interface contains methods for handling calendar related tasks for 61xx series phones. The interface is also a so-called default interface of the component. Basic functionality of the phone calendar is offered by the methods listed below.

Method	Description
CreateCalendarItem	Creates empty component for data of one calendar item.
ReadCalendarItem	Reads item from phone pointed by index.
DeleteCalendarItem	Deletes item from phone pointed by index.
WriteCalendarItem	Writes item into phone.
GetLastError	Returns last error code if error has happened. See particular documentation for further information about error codes.
Disconnect	Releases connection to phone.

7.3.1.1 CreateCalendarItem

Creates empty component for the data of one calendar item. The type of data component must be set before the created component can be used. See examples.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServEx

Syntax

```
HRESULT CreateCalendarItem(
    [out, retval] IUnknown **pCalendarItem
);
```

The **CreateCalendarItem** method syntax has these parts:

Part	Description
<i>pCalendarItem</i>	Output. An object that evaluates a CalendarItemAttrEx object.

Remarks

CreateCalendarItem function returns a new calendar event object. If the function fails, call the **GetLastError** function to get extended error information.

The created calendar event can be, e.g., stored in the phone's memory with the **WriteCalendarItem** method after it has been assigned desired values through one of the interfaces: IMessageEx, IMeetingEx, ICallToEx, IBirthdayEx.

Example

```
Public puICalendarItemServ As CALADAPTERLib.CalendarItemServEx
Public puICalendarItemAttr As CALADAPTERLib.CalendarItemAttrEx
Public puIMessage As CALADAPTERLib.IMessageEx

Private Sub CreateItem_Click()

    On Error GoTo ErrorTrap

    'create a new calendar item
    Set puICalendarItemAttr = puICalendarItemServ.CreateCalendarItem

    puICalendarItemAttr.Type = itMessage
    Set puIMessage = Nothing
    Set puIMessage = puICalendarItemAttr.TypeProperties
    puIMessage.MessageDate = Now
    puIMessage.AlarmTime = Now
    puIMessage.Message = ""
    MessageForm.Show

Exit Sub

ErrorTrap:
    MsgBox "Error: " & Err.Description
End Sub
```



```
ErrorTrap:
```

```
    MsgBox "Error #" & puICalendarItemServ.GetLastError
```

```
End Sub
```

7.3.1.2 ReadCalendarItem

Reads an item from the phone pointed by the index.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServEx

Syntax

```
HRESULT ReadCalendarItem(  
[in] long lIndex,  
[out, retval] IUnknown **pCalendarItem  
);
```

The **ReadCalendarItem** method syntax has these parts:

Part	Description
<i>lIndex</i>	Input. Index for phone calendar item to be read.
<i>pCalendarItem</i>	Output. An object that evaluates a CalendarItemAttrEx object.

Example

The following example uses the **ReadCalendarItem** function to read the given calendar item from the phone memory.

```
Public puICalendarItemServ As CALADAPTERLib.CalendarItemServEx  
Public puIndex As Long  
  
Private Sub Read_Click()  
    puIndex = Index.Text  
    On Error GoTo ErrorTrap  
  
    Set puICalendarItemAttr = Nothing  
    Set puICalendarItemAttr = _  
        puICalendarItemServ.ReadCalendarItem(puIndex)  
  
    Select Case puICalendarItemAttr.Type  
        Case itMessage  
            ...  
        Case itMeeting  
            ...  
        Case itBirthday  
            ...  
        Case itCallTo  
            ...  
    End Select  
    GoTo Success
```

```
ErrorTrap:  
    MsgBox "Error #" & puICalendarItemServ.GetLastError  
Success:  
End Sub
```

7.3.1.3 DeleteCalendarItem

This method deletes the given calendar event from the phone memory.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServEx

Syntax

```
HRESULT DeleteCalendarItem(  
[in] long lIndex  
);
```

The **DeleteCalendarItem** method syntax has these parts:

Part	Description
<i>lIndex</i>	Input. Index for phone calendar item to be deleted.

Example

The following example uses the **DeleteCalendarItem** method to delete the calendar item when the Delete button has been clicked. The calendar item index is read from the edit control named Index.

```
Public puIndex As Long  
  
Private Sub Delete_Click()  
    puIndex = Index.Text  
  
    On Error GoTo ErrorTrap  
  
    'delete the calendar item  
    Call puICalendarItemServ.DeleteCalendarItem(puIndex)  
  
    Exit Sub  
  
ErrorTrap:  
    MsgBox "Error #" & puICalendarItemServ.GetLastError  
  
End Sub
```

7.3.1.4 WriteCalendarItem

This function writes a calendar event in the phone's memory.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServEx

Syntax

```
HRESULT WriteCalendarItem(  
[in] IUnknown *pCalendarItem,  
[out, retval] long *lIndex  
);
```

The **WriteCalendarItem** function syntax has these parts:

Part	Description
<i>lIndex</i>	Output. A long integer variable. The memory index in which the calendar event was written is stored in this variable. Input. An object that evaluates a CalendarItemAttrEx object.

Remarks

Returning of the memory index is not supported by 61xx series phones. With these phones, the returned index is always zero.

If the function fails, call the **GetLastError** function to get extended error information.

Example

The following example uses the **WriteCalendarItem** function to write the message type calendar event in the phone's memory.

```
Private Sub Store_Click()  
  
    Dim Index As Long  
  
    On Error GoTo ErrorTrap  
  
    With MainForm  
        .puIMessage.MessageDate = MessageDate.Text  
        .puIMessage.AlarmTime = AlarmTime.Text  
        .puIMessage.Message = Message.Text  
  
        Index = _  
            .puICalendarItemServ.WriteCalendarItem(.puIMessage)
```

```
End With

MsgBox "Message saved in memory location " & Index & "."

GoTo Success

ErrorTrap:

MsgBox "Error #" & MainForm.puICalendarItemServ.GetLastError

Success:

Unload MessageForm

End Sub
```

7.3.1.5 GetLastError

This function returns the Component Library's latest error code value.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServEx

Syntax

```
HRESULT GetLastError(  
[out, retval] NmpAdapterError* ErrorValue  
);
```

The **GetLastError** function syntax has these parts:

Part	Description
<i>ErrorValue</i>	Output. Error code variable of types NmpAdapterError.

Remarks

The error code is maintained on a per library basis.

7.3.1.6 Disconnect

This method disconnects from the component server.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServEx

Syntax

```
HRESULT Disconnect();
```

Remarks

The **Disconnect** must always be called before releasing the latest occurrence of CALADAPTERLib library's object.

Example

The following example shows how to both create and terminate a CalendarItemServ object.

```
Public puICalendarItemServ As CALADAPTERLib.CalendarItemServEx

Private Sub Form_Load()

    Set puICalendarItemServ = New CALADAPTERLib.CalendarItemServEx

End Sub

Private Sub Form_Unload(Cancel As Integer)

    Call puICalendarItemServ.Disconnect
    Set puICalendarItemServ = Nothing

End Sub
```


7.3.2 CalendarItemServ2 Interface

The ICalendarItemServ2 interface contains methods for handling calendar-related tasks for 62xx and 71xx series phones. The basic functionality of the phone calendar is offered by methods and properties listed below.

Method	Description
CreateCalendarItem	Creates empty component for data of one calendar event.
ReadCalendarItem	Reads an event from phone pointed by index.
DeleteCalendarItem	Deletes event from phone pointed by index.
WriteCalendarItem	Writes event into phone.
UpdateCalendarItem	Updates event that already exist in phone.
NextCalendarItem	Gets next calendar event.
PrevCalendarItem	Gets previous calendar event.
GetUsedIds	Gets all used calendar IDs from phone.
GetNumberOfNotes	Depending on parameter, returns either number of all calendar items in phone or number of items in specified day.
GetLastError	Returns last error code if error has happened. See particular documentation for further information about error codes.
StartListeningEvents	Initializes notification-sending feature. See IPhoneCalendarNotify interface
StopListeningEvents	Turns off notification sending. See IPhoneCalendarNotify interface

Property	Description
get_ExpireTime	Gets count of days when calendar events are removed automatically from phone memory.
put_ExpireTime	Sets count of days when calendar events are removed automatically from phone memory.
get_DateFormat	Gets currently used format for DATE value from phone. See description of CLDR_DATE_FORMAT.
put_DateFormat	Sets currently used format for DATE value on the phone. See description of CLDR_DATE_FORMAT.
get_DateSeparator	Gets currently used separator for DATE value from phone. See description of CLDR_SEPARATOR.
put_DateSeparator	Sets currently used separator for DATE value on the phone. See description of CLDR_SEPARATOR.
get_StartOfWeek	Gets starting day of week from phone. See description of CLDR_WEEKSTART.
put_StartOfWeek	Sets starting day of week on the phone. See description of CLDR_WEEKSTART.

7.3.2.1 Type definitions

Type definitions described below are used by methods and properties, which can be found from ICalendarItemServ2 interface.

CLDR_DELETING_METHOD type can have values:

Value	Description
DELETE_INDEX	See 7.3.2.4
DELETE_OLDER_THAN_DAYS	See 7.3.2.4
DELETE_ALL	See 7.3.2.4

CLDR_SEARCH_EXTENSION type can have values:

Value	Description
CLDR_SEARCH_UNDEFINED	See 7.3.2.7 and 7.3.2.8
CLDR_SEARCH_ID_AND_TIME_UNDEFINED	See 7.3.2.7 and 7.3.2.8
CLDR_SEARCH_ALL_UNDEFINED	See 7.3.2.7 and 7.3.2.8

CLDR_DATE_FORMAT type can have values:

Value	Description
FORMAT_DD_MM_YYYY	30.12.1999
FORMAT_MM_DD_YYYY	12.30.1999
FORMAT_YYYY_MM_DD	1999.12.30

CLDR_SEPARATOR type can have values:

Value	Description
SEPARATOR_DOT	12.12.1999
SEPARATOR_SLASH	12/12/1999
SEPARATOR_DASH	12-12-1999

CLDR_WEEKSTART type can have values:

Value	Description
WEEKSTART_MONDAY	
WEEKSTART_SUNDAY	

7.3.2.2 CreateCalendarItem

See section 7.3.1.1

7.3.2.3 ReadCalendarItem

See section 7.3.1.2

7.3.2.4 DeleteCalendarItem

This method deletes the given calendar event from the phone memory.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT DeleteCalendarItem(  
[in] CLDR_DELETING_METHOD bDelete,  
[in] long lIndex  
);
```

The **DeleteCalendarItem** method syntax has these parts:

Part	Description
bDelete	Input. CLDR_DELETING_METHOD bDelete,
<i>lIndex</i>	Input. Index for phone calendar item to be deleted.

Remarks

If bDelete = DELETE_ALL, all calendar items will be deleted. In such cases lIndex parameter is ignored.

If bDelete = DELETE_OLDER_THAN_DAYS, then calendar items are deleted that have time-stamps older than the current day minus the number of days described by lIndex.

NOTE! When using this option, the maximum value of lIndex is 255. In other case **errCalendarNoDelete** is returned.

If bDelete = DELETE_INDEX, then calendar items are deleted that have index lIndex.

Example

See section 7.3.1.3

7.3.2.5 WriteCalendarItem

See section 7.3.1.4

7.3.2.6 UpdateCalendarItem

This function updates an existing calendar event.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT UpdateCalendarItem(  
[in] long lIndex,  
[in] IUnknown * pCalendarItem  
);
```

The **UpdateCalendarItem** function syntax has these parts:

Part	Description
<i>lIndex</i>	Input. A long integer variable. Index for phone calendar even to be updated.
<i>pCalendarItem</i>	Input. An object that evaluates a CalendarItemAttrEx object.

Example

See section 7.3.1.4

7.3.2.7 NextCalendarItem

This function reads the next calendar event by using different ways to find out which day to start.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT NextCalendarItem(  
[in] long lStart,  
[in] CLDR_SEARCH_EXTENSION lSearchExt,  
[in] DATE dStart,  
[out, retval] IUnknown ** pCalendarItem
```

The **NextCalendarItem** function syntax has these parts:

Part	Description
<i>lStart</i>	Input. Unique ID indicating previous event.
<i>lSearchExt,</i>	Input. Additional information for searching. See Remarks.
<i>dStart</i>	Input. Start from this day (which is included in search).
<i>pCalendarItem</i>	Output. An object that evaluates a CalendarItemAttrEx object.

Remarks

If *lSearchExt* is set to be `CLDR_SEARCH_ID_AND_TIME_UNDEFINED` the response will contain the first note within the specified date (*lStart* is ignored). Hereafter, the application can read a unique ID of the item of this specified day by using `ICalendarItemAttrEx::get_Id` property. Using this ID for the next call, the client can get the next note in line that matches the date. Now *lSearchExt* must be `CLDR_SEARCH_UNDEFINED`.
If *lSearchExt* is set to be `CLDR_SEARCH_ALL_UNDEFINED` the response will contain the first note in the calendar.

7.3.2.8 PrevCalendarItem

This function reads the next calendar event by using different ways to find out which day to start.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT NextCalendarItem(  
[in] long lStart,  
[in] CLDR_SEARCH_EXTENSION lSearchExt,  
[in] DATE dStart,  
[out, retval] IUnknown ** pCalendarItem
```

The **PrevCalendarItem** function syntax has these parts:

Part	Description
<i>lStart</i>	Input. Unique ID indicating next event.
<i>lSearchExt,</i>	Input. Additional information for searching. See Remarks.
<i>dStart</i>	Input. Start from this day (which is included in search).
<i>pCalendarItem</i>	Output. An object that evaluates a CalendarItemAttrEx object.

Remarks

If *lSearchExt* is set to be `CLDR_SEARCH_ID_AND_TIME_UNDEFINED` the response will contain the last note within the specified date (*lStart* is ignored). Hereafter, the application can read the unique ID of the item of this event by using `ICalendarItemAttrEx::get_Id` property. Using this ID for the next call, the client can get the previous note in line that matches the date. Now *lSearchExt* must be `CLDR_SEARCH_UNDEFINED`.
If *lSearchExt* is set to be `CLDR_SEARCH_ALL_UNDEFINED` the response will contain the last non repeatable note in the calendar.

7.3.2.9 GetUsedIDs

Event IDs are generated dynamically by the phone. In order to read, delete or update an event, the ID of this event must be known. This function gets all the used calendar IDs from phone as an array.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT GetUsedIds(
[out, retval] VARIANT * pId
);
```

The **GetUsedIDs** function syntax has these parts:

Part	Description
<i>pId</i>	Output. A variant containing all IDs which currently exist.

Example

```
Private Sub GetIds_Click()

    IdList.Clear
    Dim Indexies As Variant
    Dim Size As Long

    On Error GoTo ErrorTrap

    Indexies = puICalendarItemServ2.GetUsedIds

    Size = UBound(Indexies) - LBound(Indexies) + 1

    Dim X As Integer

    For X = LBound(Indexies) To UBound(Indexies)
        IdList.AddItem (CStr(Indexies(X)))
    Next

    Exit Sub

ErrorTrap:

    MsgBox "Error #" & puICalendarItemServ2.GetLastError

End Sub
```

7.3.2.10 GetNumberOfNotes

This function returns the number of existing calendar events from the phone.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT GetNumberOfNotes(  
[in] DATE dDay,  
[out, retval] short* pNmbrOfNotes  
);
```

The **GetNumberOfNotes** function syntax has these parts:

Part	Description
<i>dDay</i>	Date from which the number of events is asked, or zero. See remarks.
<i>pId</i>	Output. Number of notes.

Remarks

If *dDay* is 0, number of all events in phone calendar will returned. See example below.

Example

```
Private Sub GetNumber_Click()  
  
Dim NumberOfNotes As Long  
Dim Day As Date  
  
Day = 0  
  
On Error GoTo ErrorTrap  
  
NumberOfNotes = puICalendarItemServ2.GetNumberOfNotes(Day)  
  
Notes.Text = CStr(NumberOfNotes)  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox "Error #" & puICalendarItemServ2.GetLastError  
  
End Sub
```

7.3.2.11 GetLastError

See section 7.3.1.5

7.3.2.12 StartListeningEvents

This method informs the component object that the client is ready to receive outgoing method calls from the object. See `IPhoneCalendarNotify2` interface for more details about notifications which are on hand.

Member of `CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2`

Syntax

`HRESULT StartListeningEvents();`

Remarks

To be able to receive outgoing method calls, the client must implement and register an outgoing interface sink.

A client thread that has no message pump or a thread that may be blocked in some Win32 wait function is not allowed to declare itself as an event sink by calling **StartListeningEvents**. Otherwise, the event calls queued by the component object become blocked and the component server is likely to deadlock.

Example

In the following example, the application indicates that it is ready to receive events by placing a **StartListeningEvents** call to the component.

```
Public puICalendarItemServ As CALADAPTERLib.CalendarItemServ2

Private Sub Form_Load()

Set puICalendarItemServ = New CALADAPTERLib.CalendarItemServ2
Call puICalendarItemServ.StartListeningEvents

End Sub
```

7.3.2.13 StopListeningEvents

This method informs the component object that the client is finishing a notification listening session. This function has to be called before releasing a component instance if StartListeningEvents was called at the beginning.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

HRESULT StopListeningEvents();

7.3.2.14 **get_ExpireTime**

This property returns the number of days when events will expire. The phone manages automatic removal of expired events.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

HRESULT ExpireTime([out, retval] BYTE *pVal);

The **get_ExpireTime** property syntax has these parts:

Part	Description
<i>pVal</i>	Output. Count of days when calendar event expired.

Remarks

There are a couple of fixed choices in the phone user interface for the expiration time. Correspondences are: 1 month equals 30 days, 3 months equals 90 days and 6 months equals 180 days.

7.3.2.15 put_ExpireTime

This property sets the new value for the number of days when events will expire.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

HRESULT ExpireTime([in] BYTE newVal);

The **put_ExpireTime** property syntax has these parts:

Part	Description
<i>newVal</i>	Input. Count of days when calendar event expired.

Remarks

There are couple of fixed choices in the phone user interface for the expiration time. Correspondences are: 1 month equals 30 days, 3 months equals 90 days and 6 months equals 180 days.

7.3.2.16 **get_DateFormat**

This property returns the currently used format for dates, which are shown on the phone user interface.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT DateFormat(  
[out, retval] CLDR_DATE_FORMAT * pVal  
);
```

The **get_DateFormat** property syntax has these parts:

Part	Description
<i>pVal</i>	Output. Currently used format for dates shown on phone user interface.

Example

```
Private Sub DateFormat_Click()  
  
Dim DateFormat As CLDR_DATE_FORMAT  
  
On Error GoTo ErrorTrap  
  
DateFormat = puICalendarItemServ2.DateFormat  
  
Select Case DateFormat  
Case FORMAT_DD_MM_YYYY  
    Format.Text = "DD_MM_YYYY"  
Case FORMAT_MM_DD_YYYY  
    Format.Text = "MM_DD_YYYY"  
Case FORMAT_YYYY_MM_DD  
    Format.Text = "YYYY_MM_DD"  
End Select  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox "Error #" & puICalendarItemServ2.GetLastError  
  
End Sub
```

7.3.2.17 **put_DateFormat**

This property sets the new value for dates, which are shown on the phone user interface.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT DateFormat(  
[in] CLDR_DATE_FORMAT newVal  
);
```

The **put_DateFormat** property syntax has these parts:

Part	Description
<i>newVal</i>	Input. Currently used format for dates shown on phone user interface.

7.3.2.18 **get_DateSeparator**

This property returns the currently used separator character, which is used in the phone's user interface when date values are shown.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT DateSeparator(  
[out, retval] CLDR_SEPARATOR * pVal  
);
```

The **get_DateSeparator** property syntax has these parts:

Part	Description
<i>pVal</i>	Output. Defined values representing the separator character.

Remarks

See section 7.3.2.1 for different possible values.

7.3.2.19 put_DateSeparator

This property sets the new value for the currently used separator character, which will be used in the phone's user interface when date values are shown.

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT DateSeparator(  
[in] CLDR_SEPARATOR newVal  
);
```

The **put_DateSeparator** property syntax has these parts:

Part	Description
<i>newVal</i>	Input. Defined values representing the separator character.

Remarks

See section 7.3.2.1 for different possible values.

7.3.2.20 **get_StartOfWeek**

This property returns to the currently used day when the week starts. The day might be Monday or Sunday

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT StartOfWeek(  
[out, retval] CLDR_WEEKSTART * pVal  
);
```

The **get_StartOfWeek** property syntax has these parts:

Part	Description
<i>pVal</i>	Output. Defined values representing the day that starts the week.

Remarks

See section 7.3.2.1 for different possible values.

7.3.2.21 **put_StartOfWeek**

This property set a new value for the currently used day that starts a week. The day might be Monday or Sunday

Member of CALADAPTERLib.CalendarItemServEx.ICalendarItemServ2

Syntax

```
HRESULT StartOfWeek(  
[in] CLDR_WEEKSTART newVal  
);
```

The **put_StartOfWeek** property syntax has these parts:

Part	Description
<i>newVal</i>	Input. Defined values representing the separator character.

Remarks

See section 7.3.2.1 for different possible values.

7.3.3 IPhoneCalendarNotify2 Interface

Outgoing "source" interface contains a method for receiving calendar alarm notifications. An index of the calendar item that has raised an alarm is returned.

Event	Description
CalendarAlarm	Indicates when phone calendar has raised an alarm.

7.3.3.1 CalendarAlarm Event

Occurs when some of the calendar events in the phone have raised an alarm.

Member of CALADAPTERLib.CalendarItemServEx.IPhoneCalendarNotify2

Syntax

```
HRESULT CalendarAlarm(  
[in] long pId  
);
```

The **CalendarAlarm** event syntax has these parts:

Part	Description
<i>pId</i>	Input. Index of calendar event that has raised the alarm.

Remarks

See section 7.3.2.1 for different possible values.

Example

This event handler writes the ID of the alarming event on a label in the form.

```
Private WithEvents puICalendarItemServEx_  
    CALADAPTERLib.CalendarItemServEx  
  
Private Sub puICalendarItemServEx_CalendarAlarm(ByVal pId As Long)  
    alarm_id.Text = CStr(pId)  
End Sub
```

7.4 CalendarItemAttrEx Component

This component contains the data of one calendar event.

7.4.1 Type Definitions

Contains calendar event types.

Member of CALADAPTERLib

The following table summarizes the possible calendar event types. Types with suffix "2" are used with interfaces that have the suffix "2" in their name, e.g., IMessage2 or IBirthday2. Methods for reading and writing these kinds of events can be found from interface ICalendarItemServ2.

Events created using type-definition without any suffix, e.g., itBirthday or itMeeting, are used with interfaces that have the suffix "Ex", e.g., IMessageEx, IcallToEx. These can be handled via methods found from ICalendarItemServEx.

Property	Description
itBirthday	The calendar event is a birthday reminder.
itCallTo	The calendar event is a call reminder.
itMeeting	The calendar event is a meeting reminder.
itMessage	The calendar event is general reminder.
itBirthday2	The calendar event is a birthday reminder.
itCallTo2	The calendar event is a call reminder.
itMeeting2	The calendar event is a meeting reminder.
itMessage2	The calendar event is general reminder.

Remarks

Note that the phone does not use seconds when DATE type of data is stored; the seconds will be stripped.

7.4.2 ICalendarItemAttrEx Interface

The ICalendarItemAttrEx interface is the default interface of the CalendarItemAttrEx component.

Property	Description
Type	Type of the calendar event; either birthday, call, meeting or message reminder.
TypeProperties	Reference to the calendar event object.

7.4.2.1 Type (rw)

This property contains the type of the calendar event.

Member of CALADAPTERLib.CalendarItemAttrEx.ICalendarItemAttrEx

Default value: itMeeting

Syntax

```
HRESULT Type([out, retval] iType *pVal  
);
```

```
HRESULT Type([in] iType newVal  
);
```

Property	Description
pVal/newVal	In/Out. Value can be some of the values found from table (see section 3.1).

7.4.2.2 TypeProperties (r)

This property contains a reference to the interface object specified by the **Type** property.

Member of CALADAPTERLib.CalendarItemAttrEx.ICalendarItemAttrEx

Syntax

```
HRESULT TypeProperties([out, retval] IUnknown **pVal  
);
```

Part	Description
<i>pVal</i>	Output. An object expression that evaluates a CalendarItemAttrEx object.

7.4.3 IBirthdayEx Interface

The IBirthdayEx interface is a hidden data interface that contains information about one birthday reminder calendar event.

The IBirthdayEx objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter **7.4.11 (Event component properties)**.

Property	Description
alarmTime	The date and time at which the event will raise an alarm.
birthday	The date of a birthday.
name	The name of the person who has the birthday.

7.4.4 ICallToEx Interface

The ICallToEx interface is a hidden data interface that contains information about one call reminder calendar event.

The ICallToEx objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter 7.4.11 (Event component properties).

Property	Description
alarmTime	The date and time at which the event will raise an alarm.
callDate	The date and time at which the call should be made.
name	The name of the person who should be called.
number	The number of the person who should be called.

7.4.5 IMeetingEx Interface

The IMeetingEx interface is a hidden data interface that contains information about one meeting reminder calendar event.

The IMeetingEx objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter 7.4.11 (Event component properties).

Property	Description
alarmTime	The date and time at which the event will raise an alarm.
meetingTime	The date and time at which the meeting begins.
message	Additional message, e.g., the name of the meeting.

7.4.6 IMessageEx Interface

The IMessageEx interface is a hidden data interface that contains information about one general reminder calendar event.

The IMessageEx objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter 7.4.11 (Event component properties).

Property	Description
alarmTime	The date and time at which the event will raise an alarm.
messageDate	The date and time relating to the event.
message	Information about the event which the user is to be reminded.

7.4.7 IBirthday2 Interface

The IBirthday2 interface is a hidden data interface that contains information about one birthday reminder calendar event.

The IBirthday2 objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter 7.4.11 (Event component properties).

Property	Description
alarmTime	The date and time at which the event will raise an alarm.
birthday	The date of the birthday.
name	The name of the person who has the birthday.
audio	Audio on/off. If is set to True the alarm indication is sent to the client as set in the <i>alarmTime</i> property. If bAudio is set to False, the alarm indication is sent at 9:00 AM on the set birthday.
ID	ID number of the item in the phone. This is valid only after Read-procedure. It is not possible to choose an ID to write because this is handled by the phone calendar server.

7.4.8 ICallTo2 Interface

The ICallTo2 interface is a hidden data interface that contains information about one call reminder calendar event.

The ICallTo2 objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter 7.4.11 (Event component properties).

Property	Description
alarmTime	The date and time at which the event will raise an alarm.
callDate	The date and time at which the call should be made.
name	The name of the person who should be called.
number	The number of the person who should be called.
recurrence	Repeat interval.
ID	ID number of the item in the phone. This is valid only after Read-procedure. It is not possible to choose ID to write because this is handled by the phone calendar server.

7.4.9 IMeeting2 Interface

The IMeeting2 interface is a hidden data interface that contains information about one meeting reminder calendar event.

The IMeeting2 objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter 7.4.11 (**Event component properties**).

Property	Description
alarmTime	The date and time at which the event will raise an alarm.
meetingDate	The date and time at which the meeting begins.
message	Additional message, e.g., the name of the meeting.
recurrence	Repeat interval
ID	ID number of item in the phone. This is valid only after Read-procedure. It is not possible to choose an ID write because this is handled by the phone calendar server.

7.4.10 IMessage2 Interface

The IMessage2 interface is a hidden data interface that contains information about one general reminder calendar event.

The IMessage2 objects should not be instantiated explicitly. They are referenced through the **TypeProperties** property contained in the ICalendarItemAttrEx interface.

See description of properties from Chapter 7.4.11 (Event component properties).

Property	Description
messageDate	The date and time relating to the event.
message	Information about the event that the user is to be reminded.
recurrence	Repeat interval
ID	ID number of item in the phone. This is valid only after Read-procedure. It is not possible to choose ID to write because this is handled by the phone calendar server.

7.4.11 Event Component Properties

7.4.11.1 **alarmTime (rw)**

This property contains the time at which the calendar event raises an alarm.

Member of CALADAPTERLib.CalendarItemAttrEx.IBirthdayEx,
CALADAPTERLib.CalendarItemAttrEx.ICallToEx,
CALADAPTERLib.CalendarItemAttrEx.IMeetingEx,
CALADAPTERLib.CalendarItemAttrEx.IMessageEx,
CALADAPTERLib.CalendarItemAttrEx.IBirthday2,
CALADAPTERLib.CalendarItemAttrEx.ICallTo2,
CALADAPTERLib.CalendarItemAttrEx.IMeeting2

Type: **Date**

Syntax

```
HRESULT alarmTime([out, retval] DATE *pVal);  
HRESULT alarmTime([in] DATE newVal);
```

Property	Description
pVal/newVal	Out/in. DATE value indicates when alarm will happen.

Remarks

If the alarm is set to happen later than the meeting (IMeeting2::meetingDate), an E_INVALIDARG-error is returned when trying to save the event. In the case of ICallTo2 and IMeeting2 interfaces, only time information is shown on phone user interface. When using with IBirthday2, the date is also shown.

Example

See 7.4.11.2

7.4.11.2 birthday (rw)

This property contains the date, reminded by the calendar alarm, of the birthday.

Member of CALADAPTERLib.CalendarItemAttrEx.IBirthdayEx,
CALADAPTERLib.CalendarItemAttrEx.IBirthday2

Type: **Date**

Syntax

HRESULT birthday([out, retval] DATE *pVal);
HRESULT birthday([in] DATE newVal);

Property	Description
pVal/newVal	Out/in. DATE value indicates date of birth.

Example

The following code snippet shows how to set default attributes for the birthday calendar event.

```
Public puICalendarItemAttr As CALADAPTERLib.CalendarItemAttrEx
Public puIBirthday As CALADAPTERLib.IBirthdayEx

...
puICalendarItemAttr.Type = itBirthday
Set puIBirthday = puICalendarItemAttr.TypeProperties
puIBirthday.birthday = Now
puIBirthday.alarmTime = Now
puIBirthday.name = ""
...
```


7.4.11.3 **callDate (rw)**

This property contains the date and time, reminded by the calendar alarm, when the phone call needs to be made.

Member of CALADAPTERLib.CalendarItemAttrEx.ICallToEx,
CALADAPTERLib.CalendarItemAttrEx.ICallTo2

Type: **Date**

Syntax

```
HRESULT callDate([out, retval] DATE *pVal);  
HRESULT callDate([in] DATE newVal);
```

Property	Description
pVal/newVal	Out/in. DATE value indicates when the phone call needs to be made.

Example

The following code snippet shows how to set default attributes for the call reminder calendar event.

```
Public puICalendarItemAttr As CALADAPTERLib.CalendarItemAttrEx  
Public puICallTo As CALADAPTERLib.ICallToEx  
  
...  
puICalendarItemAttr.Type = itCallTo  
Set puICallTo = puICalendarItemAttr.TypeProperties  
puICallTo.callDate = Now  
puICallTo.alarmTime = Now  
puICallTo.name = ""  
puICallTo.number = ""  
...
```

7.4.11.4 meetingDate (rw)

This property contains the date and time, reminded by the calendar alarm, of the meeting.

Member of CALADAPTERLib.CalendarItemAttrEx.ImeetingEx.,
CALADAPTERLib.CalendarItemAttrEx.IMeeting2

Type: **Date**

Syntax

```
HRESULT meetingDate([out, retval] DATE *pVal);  
HRESULT meetingDate([in] DATE newVal);
```

Property	Description
pVal/newVal	Out/in. DATE value indicates the beginning time of the meeting.

Example

The following code snippet shows how to set default attributes for the meeting reminder calendar event.

```
Public puICalendarItemAttr As CALADAPTERLib.CalendarItemAttrEx  
Public puIMeeting As CALADAPTERLib.IMeetingEx  
  
...  
puICalendarItemAttr.Type = itMeeting  
Set puIMeeting = puICalendarItemAttr.TypeProperties  
puIMeeting.meetingDate = Now  
puIMeeting.alarmTime = Now  
puIMeeting.message = ""
```

7.4.11.5 **message (rw)**

This property contains the message to be shown when the meeting or message calendar event sends an alert.

Member of. CALADAPTERLib.CalendarItemAttrEx.ImeetingEx,
CALADAPTERLib.CalendarItemAttrEx.ImessageEx, CALADAPTER-
Lib.CalendarItemAttrEx.IMeeting2,
CALADAPTERLib.CalendarItemAttrEx.IMessage2

Type: **String**

Syntax

```
HRESULT message([out, retval] BSTR *pVal);  
HRESULT message([in] BSTR newVal);
```

Property	Description
pVal/newVal	Out/in. String value for message to be shown.

Example

The following code snippet shows how to set default attributes for the message calendar event.

```
Public puICalendarItemAttr As CALADAPTERLib.CalendarItemAttrEx  
Public puIMessage As CALADAPTERLib.IMessageEx  
  
...  
puICalendarItemAttr.Type = itMessage  
Set puIMessage = puICalendarItemAttr.TypeProperties  
puIMessage.messageDate = Now  
puIMessage.alarmTime = Now  
puIMessage.message = ""  
...
```

7.4.11.6 name (rw)

This property contains the person's name in the birthday reminder calendar event and the person's name to call in the call reminder calendar event.

Member of CALADAPTERLib.CalendarItemAttrEx.IbirthdayEx,
CALADAPTERLib.CalendarItemAttrEx.ICallToEx.,
CALADAPTERLib.CalendarItemAttrEx.IBirthday2,
CALADAPTERLib.CalendarItemAttrEx.ICallTo2

Type: **String**

Syntax

HRESULT name([out, retval] BSTR *pVal);
HRESULT name([in] BSTR newVal);

Property	Description
pVal/newVal	Out/in. String value for name of the person.

7.4.11.7 number (rw)

This property contains a phone number to call in the call reminder calendar event.

Member of CALADAPTERLib.CalendarItemAttrEx.ICallToEx.,
CALADAPTERLib.CalendarItemAttrEx.ICallTo2

Type: **String**

Syntax

HRESULT number([out, retval] BSTR *pVal);
HRESULT number([in] BSTR newVal);

Property	Description
pVal/newVal	Out/in. String value for the phone number.

7.4.11.8 ID (rw)

The property contains the ID of a calendar event. This value is managed by phone and cannot be changed. The client can use this for identifying events when searching and updating. In the case of a new event, this is ignored.

Member of CALADAPTERLib.CalendarItemAttrEx.IMeeting2,
CALADAPTERLib.CalendarItemAttrEx.IMessage2,
CALADAPTERLib.CalendarItemAttrEx.IBirthday2,
CALADAPTERLib.CalendarItemAttrEx.ICallTo2

Type: **Long**

Syntax

```
HRESULT Id([out, retval] long *pVal);  
HRESULT Id([in] long newVal);
```

Property	Description
pVal/newVal	Out/in. A unique ID of a calendar event.

7.4.11.9 recurrence (rw)

This property contains a recurrence of a calendar event.

Member of CALADAPTERLib.CalendarItemAttrEx.IMeeting2,
CALADAPTERLib.CalendarItemAttrEx.IMessage2,
CALADAPTERLib.CalendarItemAttrEx.IBirthday2,
CALADAPTERLib.CalendarItemAttrEx.ICallTo2

Type: CLDR_RECURRENCE

Syntax

```
HRESULT recurrence([out, retval] CLDR_RECURRENCE *pVal);  
HRESULT recurrence([in] CLDR_RECURRENCE newVal);
```

Property	Description
pVal/newVal	Out/in. CLDR_RECURRENCE type value indicating recurrence pattern of the event.

7.4.11.10 audio (rw)

This property contains current state of the audio alarm.

Member of CALADAPTERLib.CalendarItemAttrEx.IBirthday2

Type: **Boolean**

Syntax

```
HRESULT audio([out, retval] VARIANT_BOOL *pVal);  
HRESULT audio([in] VARIANT_BOOL newVal);
```

Property	Description
pVal/newVal	Sets Audio on/off. If this is set to True, the alarm indication is sent to the client as set in the alarmTime property. If audio is set to False, the alarm indication is sent at 9:00 AM on the set birthday.

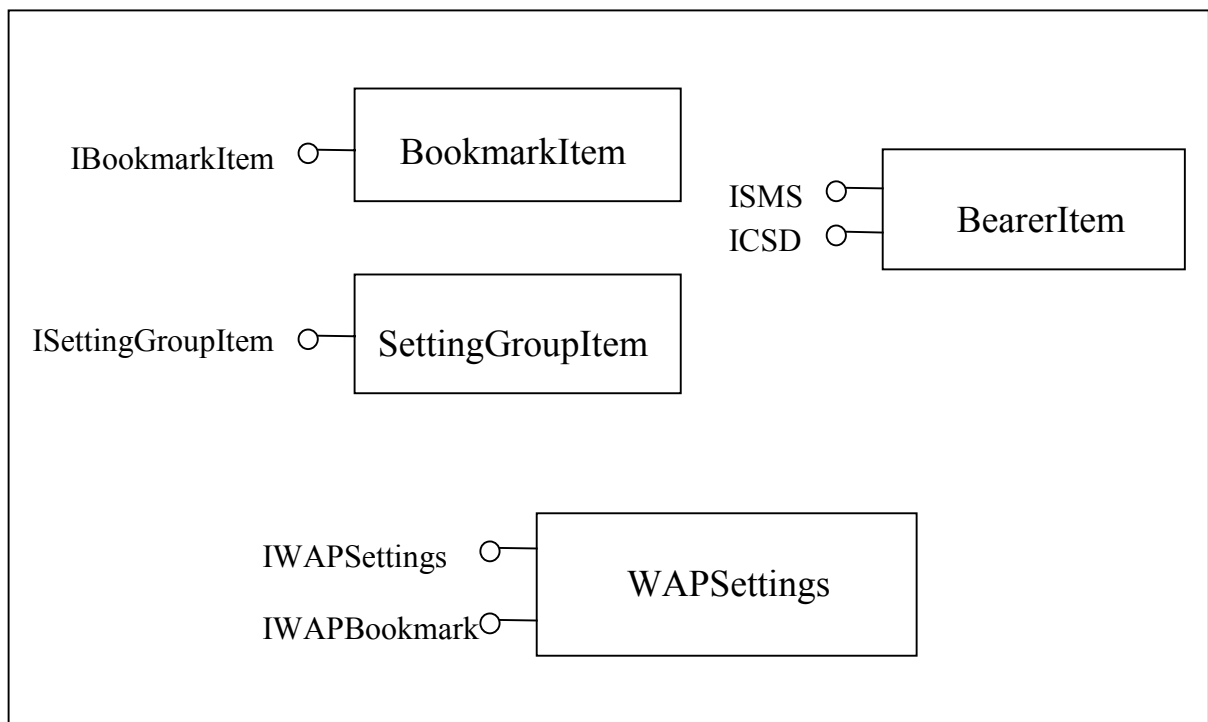
8. WAP ADAPTER LIBRARY (NOKIACLWAP)

8.1 Overview

The WAP Settings Library (NokiaCLWAP) contains components for handling WAP settings and bookmarks. With this library, it is possible to create applications that can handle WAP settings and bookmarks on the Nokia 62xx and 71xx series phones.

8.1.1 Object Model

The picture below represents the NokiaNCLWAP.dll module. The module contains the implementation for four objects: WAPSetting, SettingGroupItem, BearerItem and BookmarkItem. SettingGroupItem and BearerItem objects can be used with the IWAPSettings interface. The BookmarkItem object can be used with IWAPBookmark interface. The objects are described in more detail in the next chapters.



Picture 5. Components and interfaces of the WAP module.

8.2 WAPSettings Component

This component contains two interfaces: IWAPSettings and IWAPBookmark. IWAPSettings interface is for handling WAP setting group settings and IWAPBookmark is for handling WAP bookmarks.

8.2.1 IWAPSettings Interface

The IWAPSettings interface contains methods for handling WAP settings. The interface also contains the method for asking for supported features of the phone.

Method	Description
SettingGroupRead	Reads common WAP settings.
SettingGroupWrite	Writes common WAP settings.
ActiveGroupIndexRead	Reads currently active setting group index.
ActiveGroupIndexWrite	Sets active setting group.
BearerRead	Reads a bearer setting of the last read or written setting group.
BearerWrite	Writes a bearer setting of the last written or read setting group.
EnableService	Registers WAP service. Reading or writing is faster when using this method.
DisableService	Unregisters WAP service. This method has to be called if EnableService method is used.

8.2.1.1 SettingGroupRead

This method reads common WAP settings.

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

```
HRESULT SettingGroupRead(  
    [in] BYTE Index,  
    [out, retval] ISettingGroupItem **ppSettingGroup  
);
```

The **SettingGroupRead** method syntax has these parts:

Part	Description
<i>Index</i>	Input. A byte expression that evaluates a setting group index.
<i>ppSettingGroup</i>	Output. An object that evaluates a SettingGroupItem object (8.3.1).

Remarks

Allowed indexes are values 0–4. See also ISettingGroupItem.

Example

```
'Reads common WAP settings and shows them in text boxes

Private WAPGroupObj As NokiaCLWAP.WAPSettings

Private Sub ReadSettingGroup(ByVal SetIndex As Byte)

On Error GoTo ErrorTrap

    Dim GroupEntry As NokiaCLWAP.SettingGroupItem

    Set GroupEntry = WAPGroupObj.SettingGroupRead(SetIndex)

    SetName.Text = GroupEntry.Name
    SetHomepage.Text = GroupEntry.HomePage
    ConnectionSecurity.Text = GroupEntry.SessionSecurity
    ConnenctionType.Text = GroupEntry.SessionType
    SetBearerType.Text = GroupEntry.BearerType

    Set GroupEntry = Nothing

Exit Sub

ErrorTrap:
```

```
        MsgBox Err.Source & Chr$(10) & Err.Description  
End Sub
```

8.2.1.2 SettingGroupWrite

This method writes common WAP settings.

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

```
HRESULT SettingGroupWrite(  
    [in] BYTE Index,  
    [in] WAPBearerType BearerType,  
    [in] ISettingGroupItem * pSettingGroup  
);
```

The **SettingGroupWrite** method syntax has these parts:

Part	Description
<i>Index</i>	Input. A byte expression that evaluates a setting group index.
<i>BearerType</i>	Input. Value or constant of type WAPBearerType (8.6.1) indicating the bearer type to be modified.
<i>pSettingGroup</i>	Input. An object expression that evaluates a SettingGroupItem object (8.3.1).

Remarks

Allowed indexes are values 0-4.

Example

```
'Sets common WAP settings  
  
Private WAPGroupObj As NokiaCLWAP.WAPSettings  
  
Private Sub WriteSettingGroup(ByVal SetNumber As Byte, SetBearerType As _  
    NokiaCLWAP.WAPBearerType)  
  
    On Error GoTo ErrorTrap  
  
        Dim GroupEntry As NokiaCLWAP.SettingGroupItem  
  
        Set GroupEntry = New NokiaCLWAP.SettingGroupItem  
  
        GroupEntry.Name = SetName.Text  
        GroupEntry.HomePage = SetHomepage.Text  
        GroupEntry.SessionType = ConnectionSecurity.Text  
        GroupEntry.SessionSecurity = ConnenctionType.Text  
  
        Call WAPGroupObj.SettingGroupWrite(SetNumber, SetBearerType, GroupEntry)
```

```
        Set GroupEntry = Nothing

    Exit Sub

ErrorTrap:

        MsgBox Err.Source & Chr$(10) & Err.Description

End Sub
```

8.2.1.3 ActiveSettingIndexRead

This method reads active WAP setting group index.

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

```
HRESULT ActiveGroupIndexRead(  
    [out, retval] BYTE *Index  
);
```

The **ActiveGroupIndexRead** method syntax has these parts:

Part	Description
<i>Index</i>	Output. A byte expression that evaluates a setting group index.

Remarks

Example

```
'Reads active setting set  
  
Private WAPGroupObj As NokiaCLWAP.WAPSettings  
  
Private Sub ReadActiveSetIndex_Click()  
  
On Error GoTo ErrorTrap  
  
    ActiveSet.Text = WAPGroupObj.ActiveGroupIndexRead  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox Err.Source & Chr$(10) & Err.Description  
  
End Sub
```

8.2.1.4 ActiveGroupIndexWrite

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

```
HRESULT ActiveGroupIndexWrite(  
    [in] BYTE Index  
);
```

The **ActiveGroupIndexWrite** method syntax has these parts:

Part	Description
<i>Index</i>	Input. A byte expression that evaluates a setting group index.

Remarks

Allowed indexes are values 0-4.

Example

```
'Sets active setting set  
  
Private WAPGroupObj As NokiaCLWAP.WAPSettings  
  
Private Sub WriteActiveSetIndex_Click()  
  
On Error GoTo ErrorTrap  
  
    Call WAPGroupObj.ActiveGroupIndexWrite(ActiveSet.Text)  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox Err.Source & Chr$(10) & Err.Description  
  
End Sub
```


8.2.1.5 BearerRead

This method reads bearer settings of the setting group.

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

```
HRESULT BearerRead(  
    [in] WAPSessionType SessionType,  
    [in] WAPBearerType BearerType,  
    [out, retval] IUnknown **ppBearerItem  
);
```

The **BearerRead** method syntax has these parts:

Part	Description
<i>SessionType</i>	Input. Value or constant of type WAPSessionType (8.6.3) indicating the connection type of the setting group to be read.
<i>BearerType</i>	Input. Value or constant of type WAPBearerType (8.6.1) indicating the bearer type of the setting group to be read.
<i>ppBearerItem</i>	Output. An object that evaluates a BearerItem object that can be SMS (8.4.1) or CSD (8.4.2).

Remarks

This method reads the bearer settings of the last read setting group, which is read by the SettingGroupRead method.

Example

```
'Reads SMS bearer settings and shows them in text boxes  
  
Private WAPBearerObj As NokiaCLWAP.WAPSettings  
  
Private Sub SMSBearerRead_Click()  
  
On Error GoTo ErrorTrap  
  
    Dim SMSEntry As NokiaCLWAP.ISMS  
  
    Set SMSEntry = WAPBearerObj.BearerRead(WAP_SESSION_TYPE_CO, _  
WAP_BEARER_SMS)  
  
    DestAdd.Text = SMSEntry.DestAddress  
    GWAdd.Text = SMSEntry.GWAddress
```

```
        Set SMSEntry = Nothing  
  
Exit Sub  
  
ErrorTrap:  
        MsgBox Err.Source & Chr$(10) & Err.Description  
  
End Sub
```

8.2.1.6 BearerWrite

This method writes bearer specific settings of the last read or written setting group.

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

```
HRESULT BearerWrite(  
    [in] IUnknown *pBearerItem  
);
```

The **BearerWrite** method syntax has these parts:

Part	Description
<i>pBearerItem</i>	Input. An object variable of type SMS (8.4.1) or CSD (8.4.2).

Remarks

SettingGroupRead or SettingGroupWrite method has to be called before this method. Otherwise, the correct setting group is not known.

Example

```
'Sets SMS bearer settings  
  
Private WAPBearerObj As NokiaCLWAP.WAPSettings  
  
Private Sub SMSBearerWrite_Click()  
  
On Error GoTo ErrorTrap  
  
    Dim SMSEntry As NokiaCLWAP.ISMS  
  
    Set SMSEntry = New NokiaCLWAP.Bearer  
  
    SMSEntry.DestAddress = DestAdd.Text  
    SMSEntry.GWAddress = GWAdd.Text  
  
    Call WAPBearerObj.BearerWrite(SMSEntry)  
  
    Set SMSEntry = Nothing  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox Err.Source & Chr$(10) & Err.Description  
  
End Sub
```

8.2.1.7 EnableService

This method registers WAP service. The method's aim is to provide faster reading and writing from the phone.

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

HRESULT EnableService();

Remarks

Method has to be called before other methods. See also DisableService (8.2.1.8).

Example

'Reads all setting set names and shows them in list box

```
Private Sub ReadSettingGroups_Click()
```

```
On Error GoTo ErrorTrap
```

```
    Dim GroupEntry As NokiaCLWAP.SettingGroupItem
```

```
    WAPGroupObj.EnableService
```

```
    Dim i As Integer
```

```
    For i = 0 To 4
```

```
        Set GroupEntry = WAPGroupObj.SettingGroupRead(i)
```

```
        SettingGroupList.AddItem GroupEntry.Name
```

```
    Next i
```

```
    WAPGroupObj.DisableService
```

```
    Set BookmarkEntry = Nothing
```

```
Exit Sub
```

```
ErrorTrap:
```

```
    MsgBox Err.Source & Chr$(10) & Err.Description
```

```
End Sub
```

8.2.1.8 DisableService

This method unregisters WAP service. Method must be called when EnableService method is called.

Member of NokiaCLWAP.WAPSettings.IWAPSettings

Syntax

```
HRESULT DisableService( );
```

Remarks

Method has to be called before other method calls. See also EnableService (8.2.1.7).

Example

See 8.2.1.7

8.2.2 IWAPBookmark Interface

The IWAPBookmark interface contains methods for handling WAP bookmarks.

Method	Description
BookmarkRead	Reads bookmark from given memory index.
BookmarkWrite	Writes bookmark to given memory index.
BookmarkErase	Deletes bookmark, which is equivalent for given unique identification number of the bookmark.
EnableService	Registers WAP service. Reading or writing is faster when using this method.
DisableService	Unregisters WAP service. This method has to be called if EnableService method is used.

8.2.2.1 BookmarkRead

This method reads the bookmark properties.

Member of NokiaCLWAP.WAPSettings.IWAPBookmark

Syntax

```
HRESULT BookmarkRead(  
    [in] short Index,  
    [out, retval] IBookmarkItem ** ppBookmark  
);
```

The **BookmarkRead** method syntax has these parts:

Part	Description
<i>Index</i>	Input. A short integer expression that evaluates bookmark entry index.
<i>ppBookmark</i>	Output. An object that evaluates a BookmarkItem object (8.5.1)

Remarks

Allowed indexes are values 0–14.

Example

```
'Reads bookmark and shows them in text boxes  
  
Private BookmarkObj As NokiaCLWAP.IWAPBookmark  
  
Private Sub ReadBookmark(ByVal BookmarkIndex As Integer)  
  
On Error GoTo ErrorTrap  
  
    Dim BookmarkEntry As NokiaCLWAP.BookmarkItem  
  
    Set BookmarkEntry = BookmarkObj.BookmarkRead(BookmarkIndex)  
  
    UniqueID.Text = BookmarkEntry.UniqueID  
    Title.Text = BookmarkEntry.Title  
    URL.Text = BookmarkEntry.URL  
  
    Set BookmarkEntry = Nothing  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox Err.Source & Chr$(10) & Err.Description  
  
End Sub
```

8.2.2.2 BookmarkWrite

This method reads the bookmark properties.

Member of NokiaCLWAP.WAPSettings.IWAPBookmark

Syntax

```
HRESULT BookmarkWrite(  
    [in] IBookmarkItem * pBookmark  
);
```

The **BookmarkWrite** method syntax has these parts:

Part	Description
<i>pBookmark</i>	Input. An object expression that evaluates a BookmarkItem object (8.5.1).

Remarks

See UniqueID property (8.5.1.3) for more details.

Example

```
'Sets new bookmark  
  
Private BookmarkObj As NokiaCLWAP.IWAPBookmark  
  
Private Sub WriteBookmark_Click()  
  
On Error GoTo ErrorTrap  
  
    Dim BookmarkEntry As NokiaCLWAP.BookmarkItem  
  
    Set BookmarkEntry = New NokiaCLWAP.BookmarkItem  
  
    BookmarkEntry.UniqueID = UniqueID.Text  
    BookmarkEntry.Title = Title.Text  
    BookmarkEntry.URL = URL.Text  
  
    Call BookmarkObj.BookmarkWrite(BookmarkEntry)  
  
    Set BookmarkEntry = Nothing  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox Err.Source & Chr$(10) & Err.Description  
  
End Sub
```


8.2.2.3 BookmarkErase

This method deletes the bookmark.

Member of NokiaCLWAP.WAPSettings.IWAPBookmark

Syntax

```
HRESULT BookmarkErase(  
    [in] short UniqueID  
);
```

The **Delete** method syntax has these parts:

Part	Description
<i>UniqueID</i>	Input. A short integer expression that evaluates a unique identifier number of the bookmark.

Remarks

UniqueID is equivalent to a deleted bookmark.

Example

```
'Deletes a bookmark  
  
Private BookmarkObj As NokiaCLWAP.IWAPBookmark  
  
Private Sub EraseBookmark(ByVal BookmarkUniqueID As Integer)  
  
On Error GoTo ErrorTrap  
  
    BookmarkObj.BookmarkErase (BookmarkUniqueID)  
  
Exit Sub  
  
ErrorTrap:  
  
    MsgBox Err.Source & Chr$(10) & Err.Description  
  
End Sub
```

8.2.2.4 EnableService

This method registers WAP service. The method's aim is to provide faster reading and writing from the phone.

Member of NokiaCLWAP.WAPSettings.IWAPBookmark

Syntax

```
HRESULT EnableService( );
```

Remarks

Method has to be called before other methods. See also DisableService (8.2.1.8).

Example

See section 8.2.1.7

8.2.2.5 DisableService

This method unregisters WAP service. Method must be called when EnableService method is called.

Member of NokiaCLWAP.WAPSettings.IWAPBookmark

Syntax

```
HRESULT DisableService( );
```

Remarks

Method has to be called before other method calls. See also EnableService (8.2.1.7).

Example

See section 8.2.1.7

8.3 SettingGroupItem Component

This component contains ISettingGroup interface for handling bearer settings of WAP setting group settings.

8.3.1 ISettingGroup Interface

The ISettingGroup interface contains methods for handling specific bearer type settings and properties for WAP setting group settings. The interface also contains tree enumerated data types.

Property	Description
Name	Name of the setting group.
HomePage	Starting page of the service.
WAPSessionType	Connection type of the service.
WAPSessionSecurity	Connection security of the service.
BearerType	Bearer type of the setting group.

8.3.1.1 Name (rw)

This property contains the name of the setting group.

Member of NokiaCLWAP.SettingGroupItem.ISettingGroupItem

Syntax

```
HRESULT Name(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT Name(  
    [in] BSTR newVal  
);
```

Example

See section 8.2.1.1

8.3.1.2 HomePage (rw)

This property contains the name of the bookmark.

Member of NokiaCLWAP.SettingGroupItem.ISettingGroupItem

Syntax

```
HRESULT HomePage(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT HomePage(  
    [in] BSTR newVal  
);
```

Example

See section 8.2.1.1

8.3.1.3 WAPSessionType (rw)

This property contains the name of the bookmark.

Member of NokiaCLWAP.SettingGroupItem.ISettingGroupItem

Syntax

```
HRESULT SessionType(  
    [out, retval] WAPSessionType *pVal  
);
```

```
HRESULT SessionType(  
    [in] WAPSessionType newVal  
);
```

Example

See section 8.2.1.1

8.3.1.4 WAPSessionSecurity (rw)

This property contains the name of the bookmark.

Member of NokiaCLWAP.SettingGroupItem.ISettingGroupItem

Syntax

```
HRESULT SessionSecurity(  
    [out, retval] WAPSessionSecurity *pVal  
);
```

```
HRESULT SessionSecurity(  
    [in] WAPSessionSecurity newVal  
);
```

Example

See section 8.2.1.1

8.3.1.5 BearerType (rw)

This property contains the bearer type of the setting group.

Member of NokiaCLWAP.SettingGroupItem.ISettingGroupItem

Syntax

```
HRESULT BearerType(  
    [out, retval] WAPBearerType *pVal  
);
```

```
HRESULT BearerType(  
    [in] WAPBearerType newVal  
);
```

Example

See section 8.2.1.1

8.4 Beareritem Component

This component contains tree interfaces ISMS and ICSD. ISMS contains properties for SMS (Short Message Service) specific bearer type setting. ICSD contains properties for CSD (Circuit Switched Data) specific bearer type setting.

8.4.1 ISMS Interface

The ISMS interface contains properties for SMS (Short Message Service) bearer type settings.

Property	Description
DestAddress	Service number of the service provider.
GWAddress	Service number of the WAP server.

8.4.1.1 DestAddress (rw)

This property contains the service number of the service provider.

Member of NokiaCLWAP.BearerItem.ISMS

Syntax

```
HRESULT DestAddress(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT DestAddress(  
    [in] BSTR newVal  
);
```

Example

See 8.2.1.5

8.4.1.2 GWAddress (rw)

This property contains the service number.

Member of NokiaCLWAP.BearerItem.ISMS

Syntax

```
HRESULT GWAddress(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT GWAddress(  
    [in] BSTR newVal  
);
```

Example

See section 8.2.1.5

8.4.2 ICSD Interface

The ICSD interface contains properties for CSD (Circuit Switched Data) bearer type settings. Interface also contains four enumerated data types.

Property	Description
DestAddress	Dial-up number of the service provider.
GWAddress	IP address of the WAP server.
AuthType	Authentication type of the service.
DatacallType	Data call type of the service.
AnalogCallSpeed	Analogue data call speed.
ISDNCallSpeed	ISDN data call speed.
UserName	Username of the service.
Password	Password of the service.

8.4.2.1 DestAddress (rw)

This property contains the dial-up number of the service provider.

Member of NokiaCLWAP.BearerItem.ICSD

Syntax

```
HRESULT DestAddress(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT DestAddress(  
    [in] BSTR newVal  
);
```

8.4.2.2 GWAddress (rw)

This property contains the IP address of the WAP server.

Member of NokiaCLWAP.BearerItem.ICSD

Syntax

```
HRESULT GWAddress(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT GWAddress(  
    [in] BSTR newVal  
);
```

Remarks

GWAddress format is: xxxx.xxxx.xxxx.xxxx
where x is a number

8.4.2.3 AuthType (rw)

This property contains the authentication type of the service.

Member of NokiaCLWAP.ICSD

Syntax

```
HRESULT AuthType(  
    [out, retval] WAPAuthType *pVal  
);
```

```
HRESULT AuthType(  
    [in] WAPAuthType newVal  
);
```


8.4.2.4 DatacallType (rw)

This property contains data call type of the service.

Member of NokiaCLWAP.BearerItem.ICSD

Syntax

```
HRESULT DatacallType(  
    [out, retval] WAPDataCallType *pVal  
);
```

```
HRESULT DatacallType(  
    [in] WAPDataCallType newVal  
);
```

8.4.2.5 AnalogCallSpeed (rw)

This property contains the analogue data call speed.

Member of NokiaCLWAP.BearerItem.ICSD

Syntax

```
HRESULT AnalogCallSpeed(  
    [out, retval] WAPAnalogDatacallSpeed *pVal  
);
```

```
HRESULT AnalogCallSpeed(  
    [in] WAPAnalogDatacallSpeed newVal  
);
```

8.4.2.6 ISDNCallSpeed (rw)

This property contains the ISDN data call speed.

Member of NokiaCLWAP.BearerItem.ICSD

Syntax

```
HRESULT ISDNCallSpeed(  
    [out, retval] WAPISDNDatacallSpeed *pVal  
);
```

```
HRESULT ISDNCallSpeed(  
    [in] WAPISDNDatacallSpeed newVal  
);
```

8.4.2.7 UserName (rw)

This property contains the username of the service.

Member of NokiaCLWAP.BearerItem.ICSD

Syntax

```
HRESULT UserName(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT UserName(  
    [in] BSTR newVal  
);
```

8.4.2.8 Password (rw)

This property contains the password of the service.

Member of NokiaCLWAP.BearerItem.ICSD

Syntax

```
HRESULT Password(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT Password(  
    [in] BSTR newVal  
);
```

8.5 Bookmarkitem Component

This component contains IBookmarkItem interface that is for WAP bookmark properties.

8.5.1 BookmarkItem Interface

The BookmarkItem interface contains properties for bookmark.

Property	Description
Title	Name of the bookmark.
URL	URL address of the bookmark.
UniqueID	Unique identification number of the bookmark.

8.5.1.1 Title (rw)

This property contains the name of the bookmark.

Member of NokiaCLWAP.BookmarkItem.IBookmarkItem

Syntax

```
HRESULT Title(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT Title(  
    [in] BSTR newVal  
);
```

Example

See 8.2.2.1

8.5.1.2 URL (rw)

This property contains the URL address of the bookmark.

Member of NokiaCLWAP.BookmarkItem.IBookmarkItem

Syntax

```
HRESULT URL(  
    [out, retval] BSTR *pVal  
);
```

```
HRESULT URL(  
    [in] BSTR newVal  
);
```

Remarks

URL format is: <http://www.nokia.com>

Example

See 8.2.2.1

8.5.1.3 UniqueID (rw)

This property contains the unique identification number of the bookmark.

Member of NokiaCLWAP.BookmarkItem.IBookmarkItem

Syntax

```
HRESULT UniqueID(  
    [out, retval] short *pVal  
);
```

```
HRESULT UniqueID(  
    [in] short newVal  
);
```

Remarks

UniqueID property has to be –1 when new bookmark is created. For modifying a bookmark, it has to be equivalent to the modified bookmark's UniqueID.

Example

See section 8.2.2.1

8.6 Enumerated DATA Types

Type	Description
WAPBearerType	Bearer types of the WAP setting group.
WAPSessionSecurity	Connection security types of the WAP setting group.
WAPSessionType	Connection types of the WAP setting group.
WAPAuthType	Authentication types of the CSD bearer type.
WAPDataCallType	Data call types of the WAP setting group.
WAPAnalogDatacallSpeed	Analogue data call speeds of the WAP setting group.
WAPISDNDatacallSpeed	ISDN data call speeds of the WAP setting group.

8.6.1 WAPBearerType

Contains bearer types of the WAP setting group.

Member of NokiaCLWAP

The following table summarizes the possible bearer types.

Type	Description
WAP_BEARER_SMS	SMS bearer type.
WAP_BEARER_CSD	CSD bearer type.

8.6.2 WAPSessionSecurity

Contains connection security types of the WAP setting group.

Member of NokiaCLWAP

The following table summarizes the possible connection security types.

Type	Description
WAP_SESSION_SECURITY_UNSECURE	Secure connection security.
WAP_SESSION_SECURITY_SECURE	Unsecure connection security.

8.6.3 WAPSessionType

Contains connection types of the WAP setting group.

Member of NokiaCLWAP

The following table summarizes the possible connection types.

Type	Description
WAP_SESSION_TYPE_CL	Connectionless connection type.
WAP_SESSION_TYPE_CO	Connection oriented connection type.

8.6.4 WAPAuthType

Contains authentication types of the CSD bearer type.

Member of NokiaCLWAP

The following table summarizes the possible authentication types.

Type	Description
WAP_AUTH_TYPE_NORMAL	Normal authentication type, which uses Password Authentication Protocol; meaning clear-text passwords.
WAP_AUTH_TYPE_SECURE	Secure authentication type, which uses Challenge Handshake Authentication protocol; meaning that the most secure form of encrypted authentication supported by server and phone is used.

8.6.5 WAPDataCallType

Contains data call types of the WAP setting group.

Member of NokiaCLWAP

The following table summarizes the possible data call types.

Type	Description
WAP_DATA_CALL_TYPE_ANALOG	Analogue data call type.
WAP_DATA_CALL_TYPE_ISDN	ISDN data call type.

8.6.6 WAPAnalogDatacallSpeed

Contains analogue data call speeds of the WAP setting group.

Member of NokiaCLWAP

The following table summarizes the possible analogue data call speeds.

Type	Description
WAP_ANALOG_DATACALL_SPEED_AUTOBAUD	Automatic data call speed.
WAP_ANALOG_DATACALL_SPEED_9600	9600 data call speed.
WAP_ANALOG_DATACALL_SPEED_14400	14400 data call speed.

8.6.7 WAPISDNDatacallSpeed

Contains ISDN data call speeds of the WAP setting group.

Member of NokiaCLWAP

The following table summarizes the possible ISDN data call speeds.

Type	Description
WAP_ISDN_DATACALL_SPEED_9600	9600 data call speed.
WAP_ISDN_DATACALL_SPEED_14400	14400 data call speed.

Remarks

Values start from 1.