Smartphone 3.1

Developer Guide

Developer Guide

Smartphone 3.1

for Windows NT/2000/XP

Contents

Preface

Welcome to Smartphone 3.1, the state-of-the-art and robust multi-purpose computer telephony software environment that integrates easy-to-use tools for creating custom interactive voice response applications and complete office solutions: Automated Attendant, Unified Messaging or Voicemail with Fax-mail. The Unified Messaging system includes voicemail and faxmail functionality that is integrated with the Microsoft Exchange Server messaging system. Users are able to view fax documents, listen to voice messages, and read emails all from a single mailbox.

About the documentation set

The documentation set for Smartphone 3.1 is a complete source of information and instructions for all messaging system users and system administrators and comprises the following documents:

What's New in Smartphone 3.1

This document highlights some of the Smartphone 3.1 features that are new or improved since version 3.0.

Smartphone 3.1. User's Guide

Written for users of Smartphone Unified Messaging or the voicemail system. A complete guide that makes it easy to understand basic Unified Messaging or voicemail concepts and install Smartphone Client software on a user PC.

Smartphone 3.1. Wallet Card

A small card for quick reference that explains basic Voicemail/Unified Messaging functionality. This card can be printed and distributed to users.

Smartphone 3.1. Unified Messaging and Voicemail Installation Roadmaps

General overall of installation and use ofUnified Messaging (UM) and voicemail solutions at the customer office.

Smartphone 3.1. Getting Started

Written for the person who will install Smartphone Server and UM / voicemail system. Includes a step-by-step description of the installation process.

Smartphone 3.1. PBX Configuration Guide

Written for PBX specialists who will prepare the telephony system for integration with Smartphone. Contains technical information about configuring the PBX to work with Smartphone Voicemail and Unified Messaging.

Smartphone 3.1. Administrator's Guide

Written for the administrator of the Smartphone Unified Messaging or Voicemail system. Includes everything that the system administrator needs to know to keep the system running.

Smartphone 3.1. Application Developer's Guide

A complete guide to developing custom interactive voice applications with Smartphone development tools.

Other sources of information

Active Help

Active help refers to the ability of Smartphone to display short explanations for a program interface element (for instance, menu item) by simply placing the mouse pointer over it. This type of help is referred to as “active” since Smartphone is “actively” watching the position of the mouse pointer. These short explanations are displayed in Smartphone status bar.

ToolTips

“ToolTips” are short labels of the icons in the program toolbar. They may help you to understand what happens if you click a particular icon in the toolbar.

Online Help

Smartphone supports the full functionality of Windows NT/2000/XP help. To access help simply:

- Press the F1 key at any time for general help.
- Press the Shift + F1 key at any time for context sensitive help for the active field.
- Select [Help | Contents] from the menu bar and search by:
    - Content tree with selectable books and topics.
    - Index of topics.
    - A word.
- Select [Help | Overview] from the menu bar for the Smartphone overview help topic.
- Select [Help | Glossary] from the menu bar for a glossary of terms.
- Click on the “Help” button in any dialog.
- Click on the “What’s This?” button  with the left mouse button. Then click on the item in a dialog for which an explanation is necessary. A pop-up explanation window will appear.

While in any online help window, simply click on any highlighted keywords for a more detailed explanation of the keyword topic.

ReadMe file

The last-minute information about Smartphone can be found in the “readme.txt” file.

Document conventions

All Smartphone technical documentation is presented in Adobe Acrobat (PDF) format and shipped on the Smartphone CD-ROM. All documents include a table of contents in the beginning of the file, PDF bookmarks, a keyword index at the end of the file, and a PDF Search Index. To use the PDF Search Index:

1. make sure you have installed the version of Adobe Acrobat Reader with Search that has been shipped with Smartphone.
2. in Acrobat Reader select the [Edit | Search | Select Indexes...] menu item. The “Index Selection” dialog appears.
3. click **Add...** and navigate to <CD-ROM letter>:\Server\Docs\English\Index.pdx.
4. click **OK**.

The server index is now connected. Use [Edit | Search | Query...] to instantly search all Smartphone Server documentation.

Typographical conventions

All NOVAVOX AG documents observe the following conventions:

1. Options selected from a menu and/or submenus are enclosed in double brackets (“[ ]”) with a vertical bar (“|”) between different menu levels. For example: “[Settings | License…]” indicates the dialog opened by clicking on “Settings” in the main menu, and selecting “License…” from the submenu.
2. The names of windows, dialogs, tabs, and fields are written with quotation marks.
3. The names of buttons are written in bold, e.g.: **OK**.

About this manual

This document provides complete information for voice application developers. By learning the fundamentals of what a voice application is in Smartphone Server, one can put his or her own intelligence into the art of interactive voice response applications; just use your imagination. With Smartphone Server development tools one can build any voice application, from a simple auto-answering machine to the complicated messaging system integrated with other software or even hardware. The document includes the following sections:

Functional Overview

This section provides a step-by-step tour, instructions for writing your own voice applications, a description of the Script language, automated speech recognition, databases, and fax exchange.

Program's Reference Guide.

This section describes the various programs, including the VAP editor and the Lexicon Toolkit.

Appendices.

The appendices provide technical information, useful while developing voice applications (about system variables, script language functions and statements, glossary, etc.).

Introduction

Chapter 1: Computer Telephony Overview

This section explains the fundamentals necessary for a basic understanding of computer telephony. Consult "Glossary" for any unfamiliar terms.

The integration of computers and telephones

The integration of computers and telephones presents certain problems, in that the developmental history of the telephone and computer are completely different. Whereas in the computer field changing standards and ever-increasing performance are considered to be the driving force behind increased productivity, in the telephone industry standardization was (at least for a long time) considered to be the characteristic that would best support the productivity of the customers. Although new technologies are changing this situation, the vast majority of installed telephones are the old technology devices that are referred to here.

Why integration is necessary

The major reasons why computers are a necessary component of modern telephone systems:

Computers work 24 hours a day, 7 days a week:

Computers, with all their obvious limitations when compared to a human worker, do have one advantage in that they work 'round the clock. Thus, a computer can take care of answering the telephone during those off hours when it would be financially prohibitive to have employees working.

The PBX can send the computer information about a call:

The PBX, using DID signaling, can send the computer information about the calling telephone.

Computers send commands to the PBX:

The computer can send the PBX commands for handling calls (rerouting, putting on hold, etc.) efficiently and predictably (assuming that the computer and the PBX have both been programmed to communicate with each other).

Computers can access databases:

Using information from the PBX about the origin of a call, the computer can access computer databases to obtain information about the caller.

Why integration is a demanding task

Public telephone systems and the actual telephone set was not designed with the computer in mind. A few examples:

Rotary dialing telephones:

Rotary dial telephones, though quite adequate for signaling to the local telephone company switch what number is requested, cannot transmit information to computers over the telephone line (a computer board could possibly recognize the rotary dialing pulse signals, except that these signals are not transmitted by the telephone switching station (because when the stations were designed this was never considered to be a necessary function for switching stations)).

Touch tone telephones:

Touch tone phones, though allowing the transfer of digital information to a computer relatively easily, are also limited by the number of standard keys on the keypad (trying to transfer commands or letters can present problems).

The serial analog voice line:

The telephone line is designed to carry analog voice signals. It was not designed to carry computer signals. Though it is possible to transmit digital data over the telephone line using voice-frequency signals (such as a fax machine), it is generally difficult to transmit voice signals

and computer signals at the same time (the voice signals can be interpreted as computer signals).

Lack of standardization:

The telephone companies in each country have developed, in general, as monopolies. Therefore, each country developed its own standards that, though they may closely match the standards in other countries, can cause some rather unpredictable problems for computer telephony systems attempting to work in different countries.

What functionality is necessary from a computer telephony system

The following major functions should be provided by a computer telephony system:

Graphical programming environment:

The graphical programming environment must provide the tools for fast and simplified programming of telephone line applications.

Telephone call statistics:

Telephone call statistics can be invaluable for analysis of telephone line costs or customer demographics.

Voicemail and Faxmail:

A voicemail system provides full telephone services for all employees at a site with a PBX. Every employee has an individual mailbox. The voicemail system must have both high functionality and flexibility to handle all incoming calls that the mailbox owner cannot answer.

The tools that provide the integration and functionality

The computer requires hardware and software support to work on a standard telephone line. The computer must also be able to work with PBX's, which are becoming increasingly indispensable for even smaller companies.

PBX

Modern PBX's (Private Automatic Branch Exchange) are designed with features supporting close integration with computers. These PBX's differ from the PSTN (Public Service Telephone Network) in that:

PBX's can be programmed:

PBX's can be programmed to provide the services necessary for the local site.

Telephony boards

The standard PC does not have a telephone line jack, and thus cannot connect directly to a telephone line. Dialogic (the industry leader) telephony boards provide the necessary hardware interface between a standard PC computer slot and the telephone line.

There are many different kinds of boards to support:

- Different types of telephone connections (standard twisted pair, ISDN, etc.).
- Fax transmission.
- Speech recognition.

Telephony board drivers

A computer telephony board requires the necessary software drivers that can fully control all functionality on the board while providing a simplified software interface to the host computer. Dialogic provides extensive software support (included with the Smartphone package) that provides:

A standard software interface:

Smartphone Server can call up standard software procedures that are provided by the board drivers for supported Dialogic boards.

System settings:

These system software settings can be easily changed to work in any telephone environment. Smartphone Server provides a user interface that makes changing these settings a simple task.

Recording of voice to a computer file:

Speech on the telephone line can be digitized and written to a .VOX file on your computer.

Smartphone Server

Smartphone Server brings all of the above tools together. Smartphone Server provides a full suite of tools that provides the maximum functionality of a complete computer telephony system, including:

Graphical programming environment:

The VAP editor makes it easy to write software programs that control how telephone calls are handled by Smartphone Server.

Telephone call statistics:

The Report Generator displays information about telephone line statistics in a graphical format. For more information see “Report Generator Overview” and “Functional Topics | Statistics”.

Voicemail, Faxmail and Unified Messaging:

Smartphone 3.1 is supplied with some ready-to-use solutions. With the appropriate license code, Smartphone activates one of the built-in solutions: voicemail (with integrated faxmail) or Unified Messaging system. Unified Messaging is a revolutionary new approach developed by NOVAVOX AG that extends the possibilities of your messaging systems to a new level of utility. These solutions do not require any additional programming. They need simply to be initialized and maintained with special tools provided with Smartphone Server.

Functional Topics

Chapter 2: Step-by-step tour

This chapter explains how to write VAP's by examining example VAP's in directory C:\Program Files\Smartphone Server\Examples\Step1, Step2, etc. Each example VAP is examined in detail along with various options for changing the VAP.

For more detailed information about a particular box (especially for those box options not included in these examples) see Part 3 of this guide, Program Reference Guide - Boxes Settings Dialog.

Step 1: A Simple playback service

Step 1 (C:\Program Files\Smartphone Server\Examples\Step1\step1.vap) implements a simple playback service:

Start Box "step1 - Start" answers the phone and starts VAP execution.

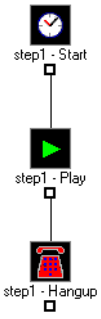
Play box "step1 - Play" plays a recorded voice file.

Hangup box "step1 - Hangup" hangs up the phone line and stops VAP execution.

Step1 could be useful for:

Answering the phone during company holidays.

This is step1.vap:



Step1.vap: explanation of each box

Start Box (step1 - Start):

A Start Box is required to start a VAP. Within the Start Box 1 of 2 possible events are defined as starting execution of the VAP with the Start Box:

The time on the host computer matches the time specified in the Start Box.

A certain number of rings has been detected on the phone line the VAP is running on.

Note: A VAP can contain more than 1 Start Box. The VAP will start executing at the first Start Box whose internal requirements for starting (a certain time has been reached or a certain number of rings has been detected) has been reached.

This is the Start Box in Step1.vap:

In the section "Event Settings" the field "Answer Call after … Rings" = 2: The Start Box will answer a call on the line after 2 rings.

Play Box (step1 - Play)

A Play box plays a recorded voice file.

This is the Play box in Step1.vap:

In the section “Greeting” the checkbox “Play Greeting” is checked: File “s1mess.vox” will be played on the phone line.

In the section “Greeting” the checkbox “Allow to stop Greeting” is checked: If the caller presses any key while the voice file is being played, the voice file immediately stops playing.

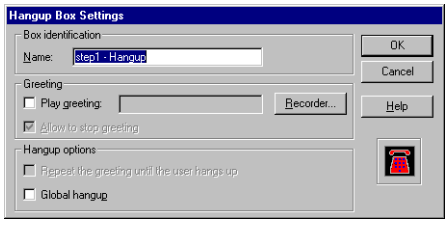
Hangup Box (step1 - Hangup)

A Hangup Box:

Instructs the Dialogic board to “hangup” on the line.

Ends the execution of the VAP.

This is the Hangup box in Step1.vap:



The above Hangup box will hangup the line immediately upon execution.

Step 2: A call transfer system

The application step2.vap is found in C:\Program Files\Smartphone Server\Examples\Step2

In Step 2 we present the Dialout Box. The box can connect the caller via PBX to any appropriate number. Often such systems are also named ACD (Automated Call Distribution) or Auto Attended Systems. In our example we show the very basic Version of such systems.

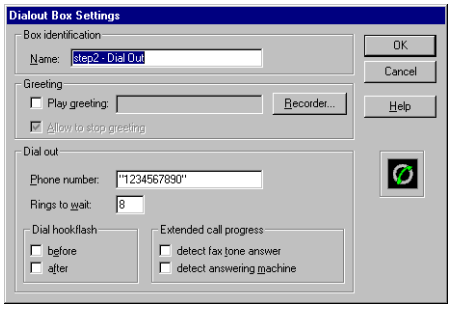
All incoming calls will be connected to fixed extension Numbers. This will work only if Smartphone Server is connected to a PBX system. The caller will hear the tones of the digits being dialed upon making the transfer.

Step2.vap

Compared to Step 1 we only added the Dial-Out Box to the Voice Application. After Hook off, the VAP announces the greeting-prompt, goes on hold and connects to the Number 1234567890.

The Dialout Box

The Dial-Out Box will look like this example.



Smartphone Server will ring up to 8 times before proceeding to the (N) Box-Exit. If the called party is occupied, Smartphone Server will proceed to the (B) Box-Exit.

If you use Smartphone Server connected to a PBX system, it is necessary to send a Hookflash to put the caller on Hold and get the free line. On error, please check your PBX Manual for the Hookflash time. It can be adjusted in the [Settings | Telephone Lines] menu of the Smartphone Server main window.

Step 3: A simple answering machine system

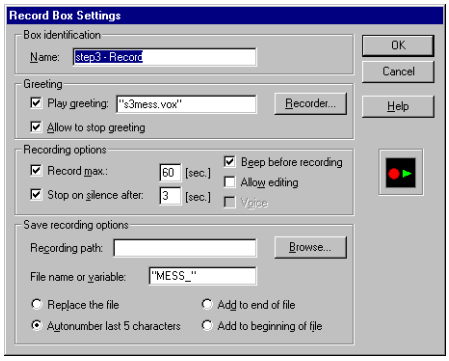
The application step3.vap is found in C:\Program Files\Smartphone Server\Examples\Step3

The Record box is introduced in step 2. This box is also one of the most important activity boxes, as is the Play box. The Record box is responsible for the recording of prompts and messages. It occurs in almost every application, be it a system connection, an information system, etc. A simple answering machine is prepared by using this box in step 2.

Step3.vap

The Record Box

The program plays a prompt, sends a signal tone and then takes in up to a 60 second message.



The Record box offers the standard options of all activity boxes, which are box identification and greeting. These have already been explained comprehensively in the introduction. The further options are divided into two sections: Those for the record settings and those to modify the files; following is more detail on these tw sections.

Record Settings:

Record, max. [Sec.]: Defines the maximum length of the recorded file. If the time (in seconds) is surpassed, the recording is stopped automatically.

Beep Before Recording: A short beep is played before recording starts.

Stop on Silence after (max.): Specify here after how many seconds of silence the recording should stop. If this option is not defined, the recording can only be stopped if the caller presses a key.

Record File:

Recording Path and File name: The file name must be specified according to Smartphone Server standards. In addition, the path can be explicitly selected. Without such an explicit directory selection, the directory in which the VAP is located will be used.

Autonumber last 5 characters: Firstly, it is this option that makes a real answering machine possible. Every new recording would normally be overwritten; this option, however, counts the incoming messages automatically, adding the number at the end of the file name.

Note: in the case of automatic numbering, only the first four characters of the file name are considered.

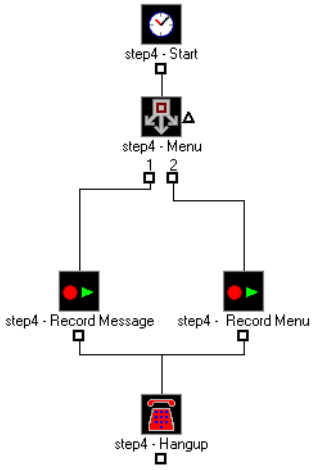
Note: It is always possible for the caller to interrupt a recording within a record box by pressing a key on the telephone pad.

Step 4: Interactive answering machine system

The application step4.vap is found in C:\Program Files\Smartphone Server\Examples\Step4

The Branch box, also called the menu box, is introduced in this step. It allows a reaction to the caller's telephone keypad input, thus determining the continuation of the program run. It also allows your VAP to choose a particular branch according to variable values and so on. Step 4 also shows how prompts can be changed over the telephone.

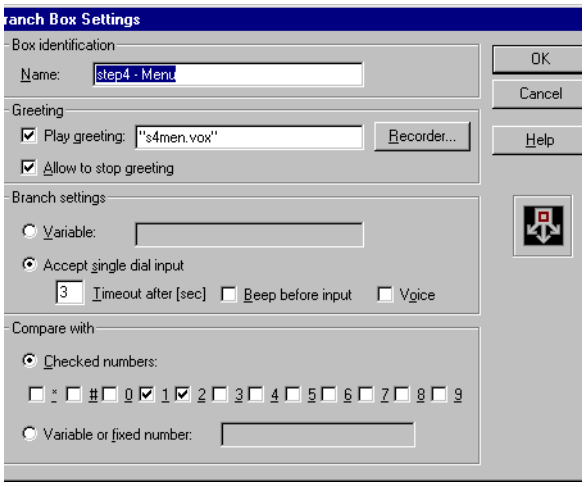
step4.vap



When the application is started, a menu is heard: "Press the number 1 if you would like to leave a message, or number 2 if you want to change your recorded message". Then telephone keypad input is required. Press the number 1 on your phone pad, and "Please leave your message" is heard, at which time the message can be given. Press 2, and "Please enter the menu text", at which time the caller can alter the greeting or record a new one.

The Branch Box

The Branch box is where choices are made. The caller hears a pre-recorded prompt, decides which menu to follow, enters the appropriate number on his/her telephone keypad, and thus determines the program run. The error exit is chosen should the caller not give any input or gives a choice which has not been defined. (In the example below, such choices would be numbers: 3,4,5,6,7,8,9,0,*, #).



The options, which were chosen in this case, mean the following:

Greeting:

Play greeting: s4men.vox is the prompt which contains the menu selections (“Press 1 if you...”).

Allow to stop greeting: Gives the caller the possibility to interrupt a pre-recorded prompt in order to give his/her input. In step4.vap, the caller can make in input immediately while the menu is still being played.

Branch: Settings

Variables: If you want to branch according to the value of a variable, enter the name of the variable here. Smartphone Server then uses the exit defined by the value of the variable.

Accept single dial input: This option must be chosen if a Branch box is used as a menu, and one character from the telephone keypad is expected as input. As a result of this input (either 0 or 1 in this case) the program run will then branch accordingly.

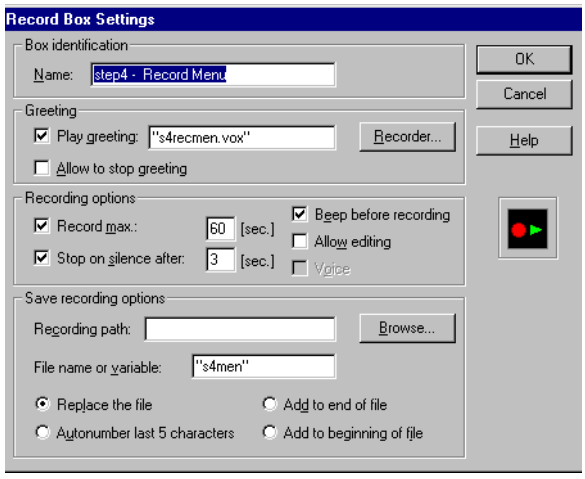
Beep before input: If marked, the caller will hear a beep before giving input.

Compare With

Checked Numbers: By marking the character boxes (*, 1, 2, 3, ...) the allowable input to which the box should react, is defined. If input is made other than those numbers marked, the application will take the error exit.

When a single number causes the application run to branch accordingly, this is what is referred to as a menu box. A branch box is referred to when a variable value causes the applicaiton to branch.

Note: *The Save recording options of the record box, where prompts can be changed, look as follows:*



As seen here, the stack-up messages option was not chosen since a particular prompt should be changed. The new recording overwrites the old one.

Remarks

As is easily recognized, such an answering machine is not usable in practice since each caller would have the possibility to amend the prompts Naturally, it is possible to change the menu text so that the caller does not realize that the “2” text can be changed. The voice prompt of the message box would then say : “I am not at home at the moment; if you wish to leave a message, press the number “1”.

Indeed, the danger still exists then, that somebody presses “2” by chance, thus amending the announcement. There-

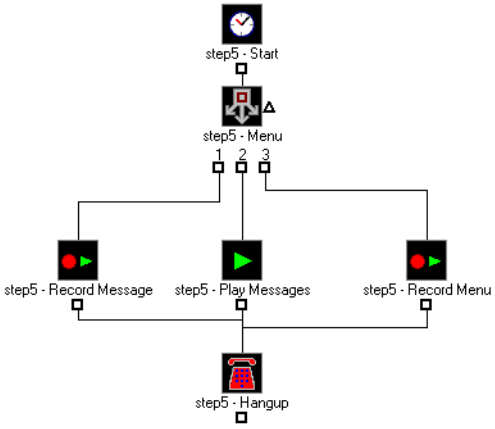
fore, this routine must be protected from unauthorized access in any case. The insertion of a PIN (personal identification number) for this reason is shown in step 6.

Step 5: Answering machine with remote access

The application step5.vap is found in C:\Program Files\Smartphone Server\Examples\Step5

The messages that have been recorded in the Record box, can be played back not only on a computer but also by means of a Play box. Thus “remote access”, the possibility to listen to recorded messages via an external telephone is available.

Step5.vap

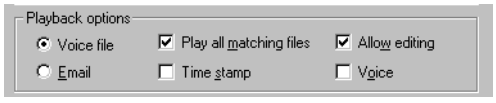


The caller hears the following menu: “No one is here at the moment. If you want to leave a message, press 1” (Neither of the other menu options are mentioned). Then telephone keypad input is requested. If the caller preses “1”, he/she hears : “Please leave your message now” and the caller then speaks. If the caller chooses “2”, all recorded messages can be heard. Pressing “3”allows prompts to be altered.

The Play Box Part II

In contrast to step 1, where a pre-recorded text is simply played back, just as one would push “play” on a cassette recorder, the play box is inserted here in order to hear all previously recorded messages. An integrated play menu is also inserted (“would you like to hear the next message, would you like the message repeated, would you like to erase the message, or, would you like to exit”, system prompt). If “next message” is chosen but there is not another message, the box plays the system prompt and the program run continues.

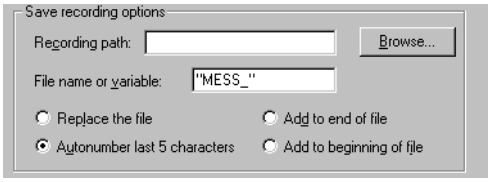
Multiple playback and the integrated menu are switched on simply by choosing the “Play all matching files” and “Allow editing” options. Should only one file with this name exist, the play box treats it as a normal “play”, the file is played, and the VAP then continues.



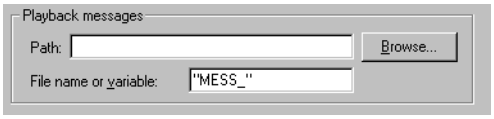
The coordination of recording the messages and subsequently playing the same ones back is based on a four character file name. The name specified in the record box recording the messages has to be identical to the one specified in the play box playing the messages back. The options in the Play and the Record boxes clarify this fact.

If the Time Stamp feature is enabled in a Play box, every message played in that Play box will include the time and date the message was recorded. The meaning of the option “Allow editing” is explained below :

Record Box



Play Box



This, of course, is by no means a complete Voicemail system; for one hundred people, each with his own mailbox, one hundred Play boxes and one hundred Record boxes would be needed, each with a different file name. In such a case, variables would be inserted, which is described later.

Allow editing

"Allow editing" is an important function for message systems. If it is selected together with "Play all matching files", the system will play a menu after every message, allowing the caller to repeat the playback of a message, go to the next message, delete a message, or exit the menu.

In principle, all incoming messages are stored until the addressee explicitly deletes them. The messages are played back in sorted order; that is, the first message recorded will be the first message played back.

Note:

The agreement of both file names is extremely important. If a recorded message is to be played back, both file names (record file and play back file) must be identical.

Remarks

This step lacks the ability to recognize unauthorized users. Any caller may still listen to all of the messages and change voice prompts. Otherwise, this is an already advanced Voicemail application, which even allows greetings to be changed via the telephone.

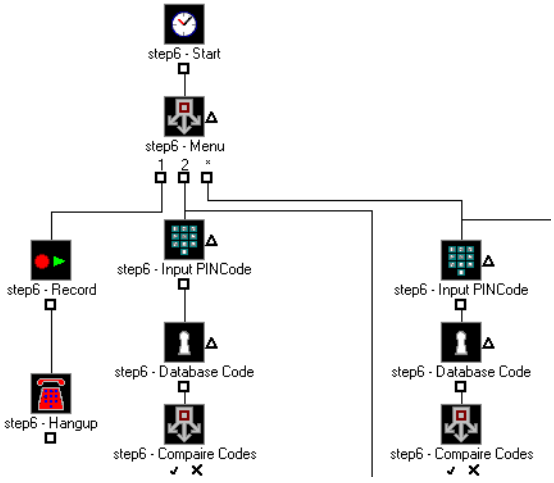
Step 6: Password protected system

The Application step6.vap is found in C:\Program Files\Smartphone Server\Examples\Step6

This is an overview of the basic PIN principle, which enables authorization within an application.

Changing your PIN by phone is also explained.

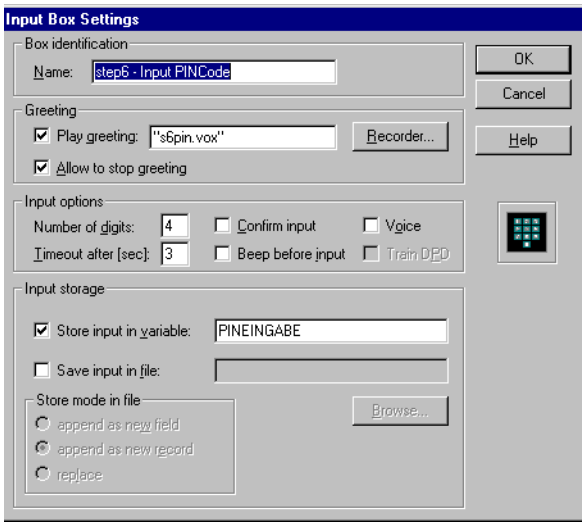
Step6.vap Part 1

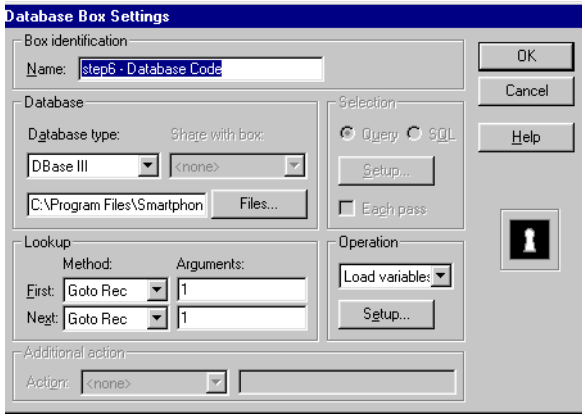


The first part of the example shows a menu box (Branch box) with different selections depending on the value introduced. By pressing "1" the caller can leave a message, pressing "2" allows any recorded messages to be heard (after the correct PIN is provided). By pressing the star key the caller can change his PIN.

The Input Box

To introduce a PIN the Input Box is used. For step 6 – InputPINCode – the configuration is the following:





As can be seen, the password consists of three numbers. The input will be stored in the variable PINEINGABE.

To store the input as a text file, select the option “Save Input in File”. This function will not be used in our example.

The Database Box

The PIN is stored in the database file s6pin.dbf in the PIN field. Here the PIN is defined as 1234. The PIN will be loaded into the PIN variable.

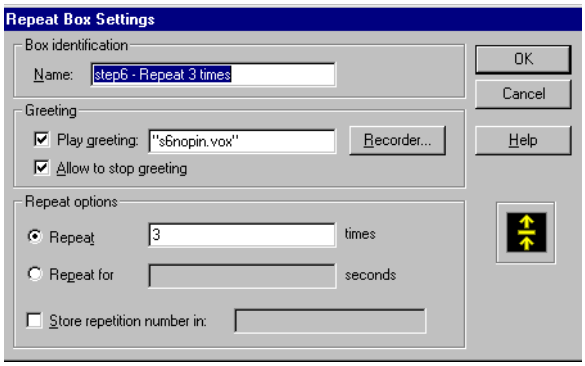
Since the current PIN can be changed in this application, the PIN has to be saved in a database, from which it can be loaded, changed and finally replaced by the new code.

The Branch Box

In the Branch Box the password introduced will be checked against the PIN information in the database. If they coincide, received messages can be played out using the Play Box. If the wrong password is entered, the caller has the possibility of re-enteringthe input. For security reasons, the caller can only repeat input of the PIN three times, after which Smartphone Server will disconnect.

The Repeat Box

Settings for the Repeat Box are as follows:

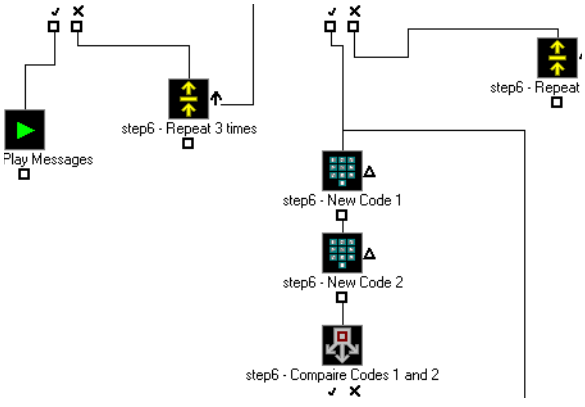


When the Repeat Box is activated, a prompt will be played notifying the caller that the input is incorrect, and it then asks caller to repeat input. After running three times, the application aborts any further repetition and terminates (regular exit).

PIN per Telephone

If the caller presses the star key in the menu, he can change the PIN telephone keypad input. Prior to any changes, a caller must correctly enter the current PIN (just as when “2” is pressed).

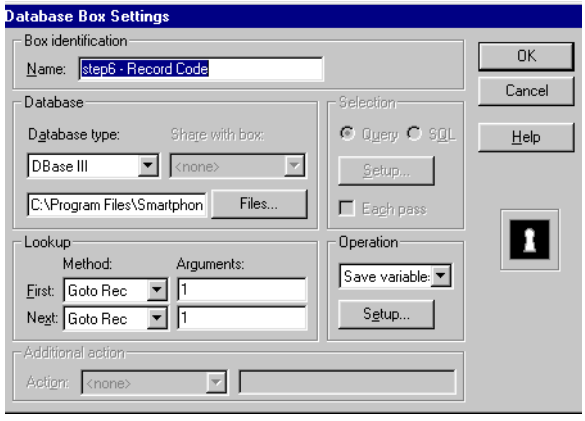
step6.vap Part 2

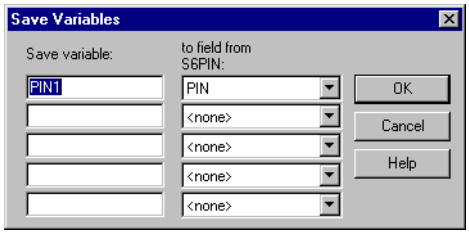


Next, the caller will be asked to input the new PINCODE, which has to be entered two times. Both inputs will be compared inside the Branch box. If they do not coincide, the procedure can be repeated up to three times.

If the two inputs coincide, the old PIN will be replaced by the new one in the database.

The settings in the Database Box are as follows:





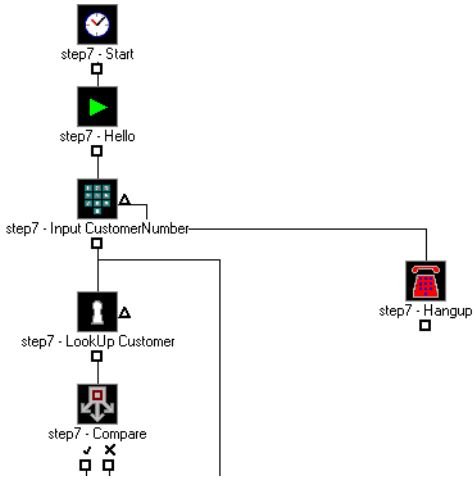
Step 7: A mail order telephone system

The Application step7.vap is found in C:\Program Files\Smartphone Server\Examples\Step7

Step 7 shows how the application can be used to place phone orders. A Client can order different items with his Client Number and corresponding PIN by simply press-

ing the phone keys. For this application two databases are used: one for the specific Client Data of the client and one to store the items ordered. With the help of this example the functionality of the Textout and Assignment boxes will be shown. A text file will also be created in which the ordering information will be specified.

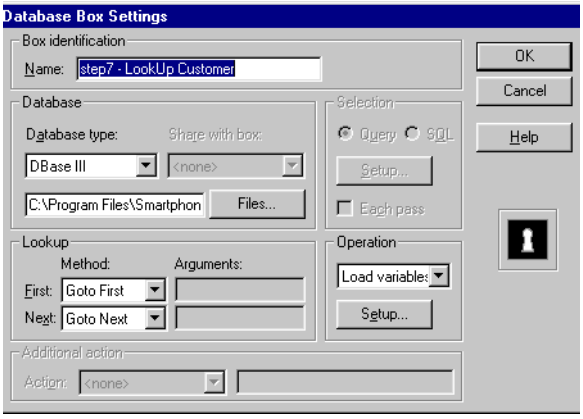
Step7.vap Part 1



When the application starts up, the caller is asked to input his Client Number (4 characters). Once the code is entered, the specific client record in the database s7kundnr.dbf will be searched. If the caller introduces the wrong number, or the Client Number does not exist in the database, the application will proceed to the Repeat box, which allows three repetitions. After three wrong inputs or three unsuccessful searches, Smartphone Server cuts off the call.

The Database Box (I)

The Database Box step 7 – client data are configured as follows:



The database file s7kunden.dbf

The delivery database appears as follows:

Table 1:

	Record 1	Record 2
K_Nummer	0001	0002
PIN	1111	2222
Name	Hans Hansen	Fritz Fritzen
Anrede	Herr	Herr

Table 2:

	Record 3	Record 4
K_Nummer	0003	0004
PIN	3333	4444
Name	Cornelia Cornelius	Petra Peter
Anrede	Frau	Frau

It consists of four records each containing four fields with the Client number, PIN, name and title.

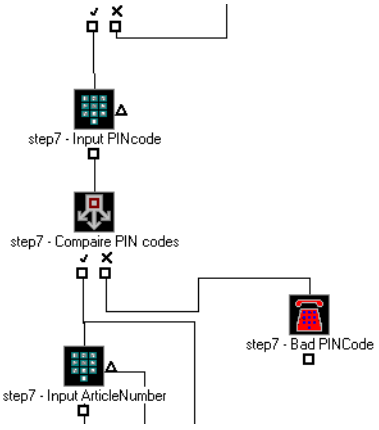
In the Database box the contents of the fields will be loaded into the Smartphone Server variables in the following pattern:

Table 3:

K_Nummer	in	KNUMMER
PIN	in	KPIN
Name	in	KNAME
Anrede	in	KANREDE

Next steps in running the application:

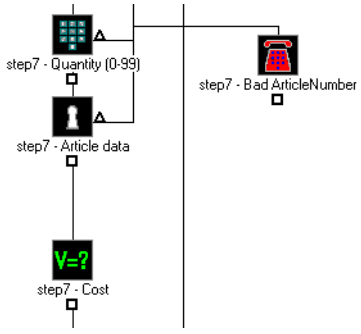
Step7.vap Part 2



The client is asked for his PIN (order code). Smartphone Server compares the input of four characters with the variable PIN that has been loaded previously from the client database.

If the input and the variable PIN do not coincide, or the input is wrong, Smartphone Server cuts off the call. In the third part the actual order and billing information will be formulated.

Step7.vap Part 3



The client is asked for the item number (1 character) and the quantity of items he wants to order. After this the corresponding description and price of this item will be read from the Items database.

The Database Box (II)

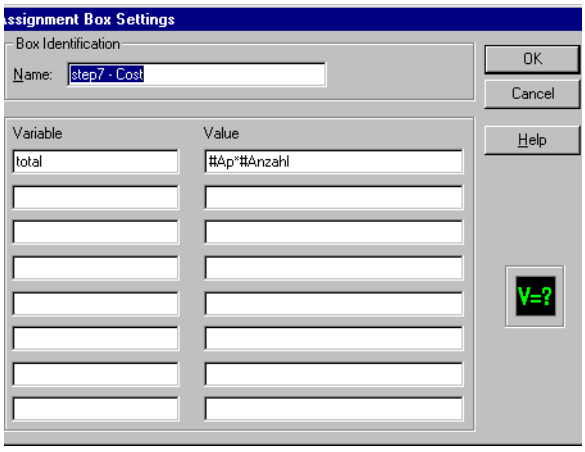
The Items database s7artik.dbf appear as follows:

Table 4:

Record number	Item type	Item price
1	Umbrella	30
2	Flowerhat	14
3	Chair	46
4	Computer game	8
5	Pen	1

Smartphone Server reads from the corresponding database (item number) the description and the price per item and stores this information in the Smartphone Server variables An and Ap respectively.

The Assignment Box



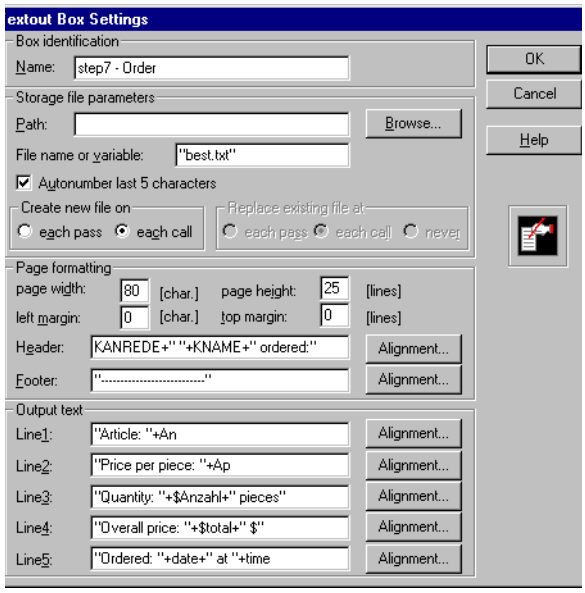
Here a calculation is made: price per item multiplied by number of items. The number of items has been inputted by the client and the price per item is read by Smartphone Server from the item database.

The result is stored in the variable “total”.

Please note that Smartphone Server only processes integers.

The Textout Box

In this example a text file is created in the Textout Box. This text file stores the name of the client, the item, the number of items, the date and the total amount of the purchase order. Step 7 appears as follows:



To connect the sequences of characters and the Smartphone Server variables, the Smartphone Server Alignment command is used.

Before the application exits the client is asked whether or not he would like to place another order.

The example

For a better understanding of the different functions, a simulated example is shown. Let us suppose Miss Petra Peter wants to order an umbrella and five chairs.

She calls and Smartphone Server welcomes her with the message: s7mess.vox. After she has been asked for her Client Number, she presses “0004”.

Smartphone Server now searches the client database for the Client Number. If Smartphone Server finds the Client Number, the values for the Client Number, the PIN and the name and title will be loaded into the Smartphone Server variables Client Number, PIN, name and title.

Now Miss Peter is asked to input her PIN. She fails with the first attempt and introduces “1444”. This input does not coincide with the stored code (PIN). Consequently,

the application asks again to input the (PIN). The second time Miss Peter presses the right sequence “4444”.

Now she is asked for the item number and number of items. She presses item number “1” for the umbrella and “1” for the number of items. This will be written into text file c:\best0001.txt.

Now Miss Peter is asked if she wants to place another order. This is, in fact, the case so she presses “1”.

Then she orders 5 chairs, so she will press key “3” (item number) and key “5” (number of items).

This order will also be written into text file c:\best0001.txt.

If we open the text file, the following entries can be seen:

Table 5:

Miss Peters has ordered:
Article: Umbrella
Item Price: $30
Number of items: 1
Total price: $30
Order taken on 4.10.95 21.30
Miss Peters has ordered:
Article: Chair
Item Price: $46
Number of items: 5
Total price: $230
Order taken on 4.10.95 21.31

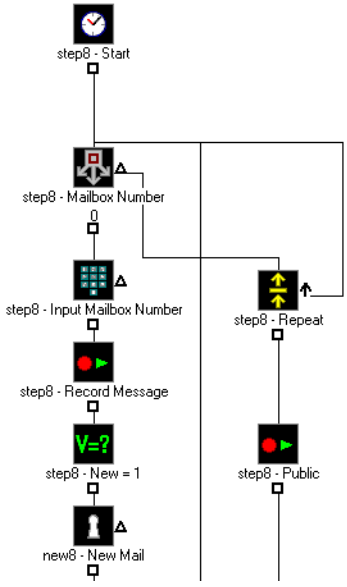
Also shown here are the system variables “date” and “time” used by the application to introduce the date and the time of the order. Further information on system variables can be found in the “System Variables” appendix.

Step 8: The Intelligent Voicemail System

The Application step8.vap is found in C:\Program Files\Smartphone Server\Examples\Step8

Step 8 shows an example of the Voicemail system. Every morning the system calls all the persons who have a message in their mailboxes. When the system recognizes a message as having been recorded for one of the clients, the system creates a field in the database, which is either marked “0” or “1”. Such a field is called a “flag”.

step8.vap Part 1



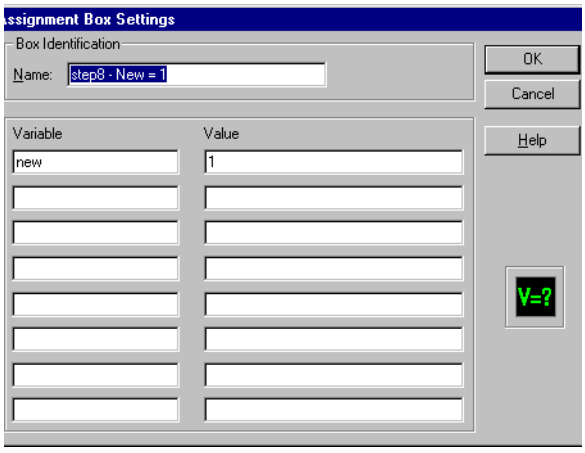
The caller is asked whether he knows the Mailbox Number of the desired talk partner. If not, a menu is played, providing with information about the numbers. Now the caller has the option of inputting the Mailbox Number or of talking to a Public Mailbox.

If the number of the mailbox is known, after pressing “0” the system asks for the Mailbox Number.

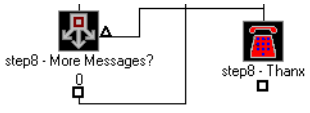
Then the message will be recorded. Now Smartphone Server marks the “New Mail flag” by writing a “1” in the “new mail” field record.

The Assignment Box

In the Assignment Box the Smartphone Server- variable will be set to “1”. The settings now appear as follows:

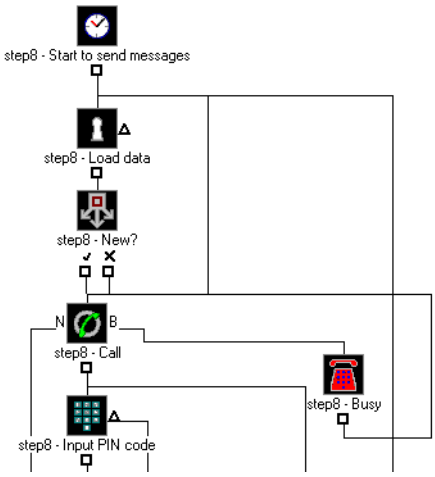


The caller is then asked if he wants to leave any other messages. If not, Smartphone Server hangs up.



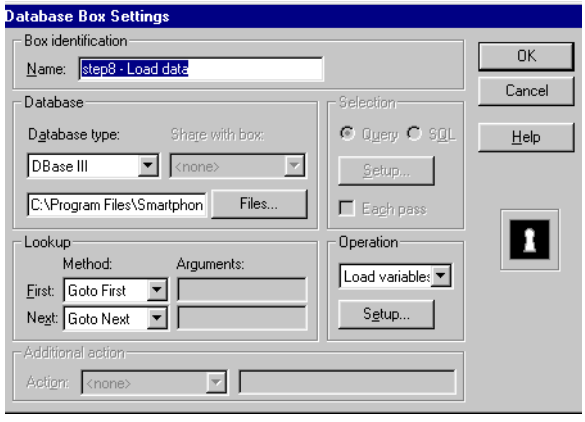
step8.vap Part 2

In the second part of the application, the postman principle is used. Smartphone Server calls at 7:30 a.m. every mailbox user who has received a message.



The Database Box loads all data necessary for Smartphone Server to proceed to execute the application. These are the phone number, the New Mail flag and the PIN. The PIN has to be used, because Smartphone Server cannot detect who picks up the phone.

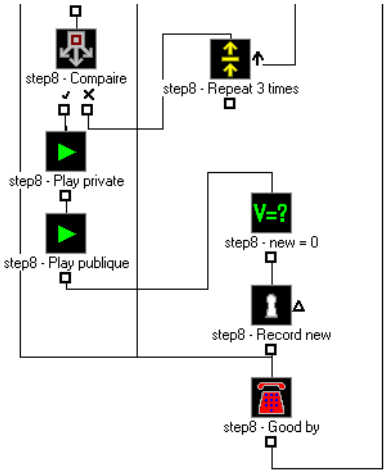
The settings of the Database Box look as follows:



Smartphone Server checks the following Branch Box to see if the mailbox user has received any new messages. If not, Smartphone Server loads the data of the next record.

However, if the New Mail flag marks “1”, the mailbox user will be called and asked for his PIN Code.

step8.vap Part 3



Should the PIN Code introduced coincide with the one in the database, the system plays out all personal messages. Here the system variable “lastdbrecord” is used, where the number in the current database is recorded. This number is identical to the mailbox number where the messages have been stored.

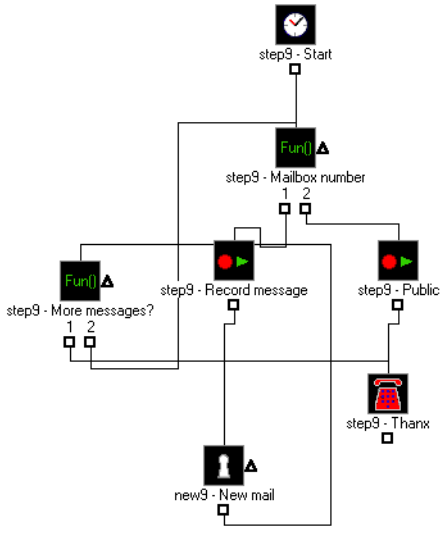
When this process is completed the system plays all public messages as well. Finally Smartphone Server puts the New Mail flag back to “0” and saves it in the database. Smartphone Server now jumps to the next record and so on, until it runs through the entire database.

Step 9: Using Script Language

The Application step9.vap is found in C:\Program Files\Smartphone Server\Examples\Step9

Step 9 shows an example of a Voicemail system equal to the one illustrated in step 8, but written using the Script Box. The system recognizes when a message has been received for one of the users and flags the Mailbox accordingly. The system then calls the users whose Mailboxes have been flagged with new messages to notify them and play the messages.

Step9.vap Part 1



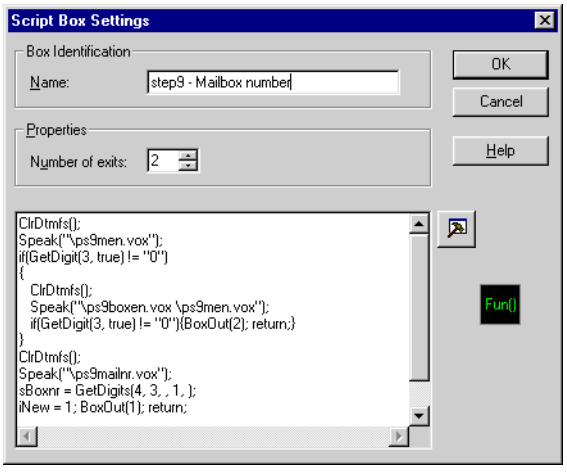
In this application the caller is asked whether he knows the Mailbox Number of the desired talk partner. If not, a menu is played, providing information about the numbers. Now the caller has the option of inputting the Mailbox Number or of talking to a Public Mailbox.

If the number of the mailbox is known, after pressing “0” the system asks for the Mailbox Number.

Then the message will be recorded. Now Smartphone Server marks the “New Mail flag” by writing a “1” in the “new mail” field.

The Script Box

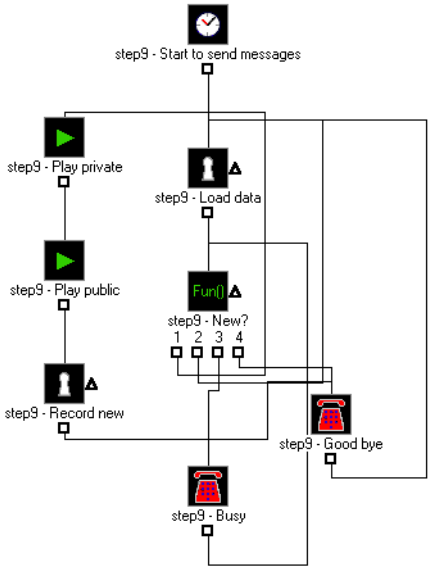
The above first steps are all accomplished by one Script Box. Thus, in this case, the functionality of the Branch, Repeat, Input and Assignment Boxes is combined into one box using Script Language. The box is programmed to play the initial recording, accept input, compare the input, and assign a variable to the field indicating a new message has been received. The necessary number of exits can be assigned and the actual Script Language is entered into the Editor Window. Next to the window where SL is entered is the Hammer button; by clicking on it you can open the Expression Builder. The Expression Builder can help you with your SL expressions as it provides folders containing all available functions, constants, variables and operators . Following is an example of the settings for the first Script Box in step 9:



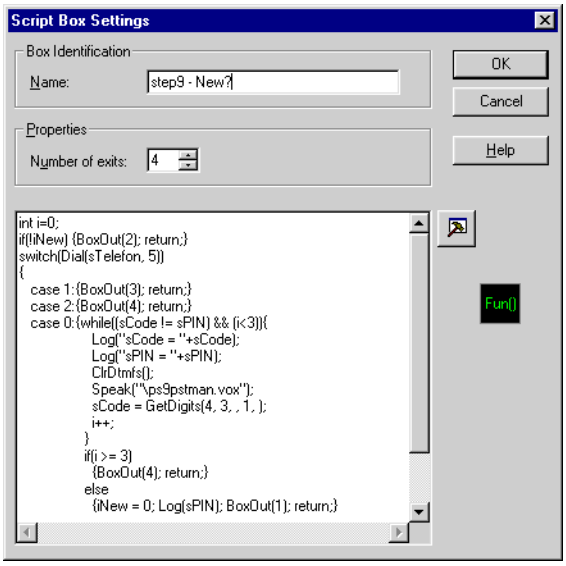
After recording a message the caller is then asked if he would like to leave any additional messages (again a Script Box is used for this function). If not, Smartphone Server then hangs up after storing the messages to the appropriate mailbox directories.

Step9.vap Part 2

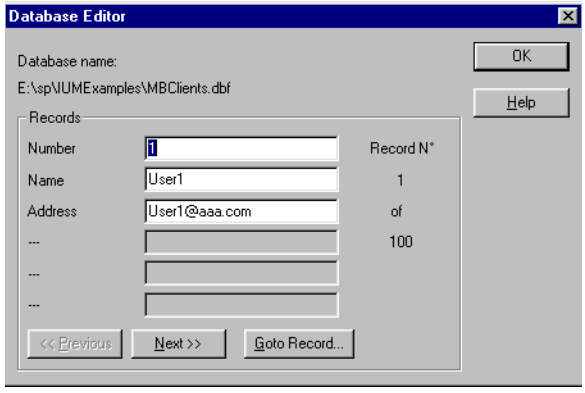
Again, similar to Step 8, the second part of the application works on the postman principle. The Database Box loads the necessary data (phone number, new mail flag, and PIN) for the Application to proceed with Smartphone Server calling every mailbox user who has a new message in his mailbox.



Following, in one box, Script Language is used to execute the functions of checking if there are new messages for the user, making a call to the user, asking for and identifying the PIN, and setting the new message flag back to 0 after the messages are played. Four exits have been set for this Script Box. The first exit is taken after the called user identifies himself by his PIN; his new messages, followed by public messages will be played. The second exit is taken when the system identifies that the record (a certain user) had no new messages and the application continues checking the next record without making a call. The third exit is taken if the called phone number is busy; and the fourth if the phone was not answered. Below are the settings for the Script Box in this part of the application:



First, you must fill in the real address names of the persons to whom voice messages may be sent. To do this, start “Database Editor” and open the file “MBClients.dbf” located in the folder “C:\Program Files\Smartphone Server\Examples\Step10”:



After messages have been played for a user, the application goes on to the next record and so on until Smartphone Server runs through the entire database of users.

Step 10: Sending or reading emails via phone

The Application step10.vap is found in C:\Program Files\Smartphone Server\Examples\Step10

Step 10 shows an example of how to link your telephone network and the electronic mail system. users can send voice messages to the email addresses that are stored in the database and listen to their own emails via phone using the text-to-speech technique.

Note: *Before this step, you need to adapt your environment to be able to perform the related functions. See the directions below.*

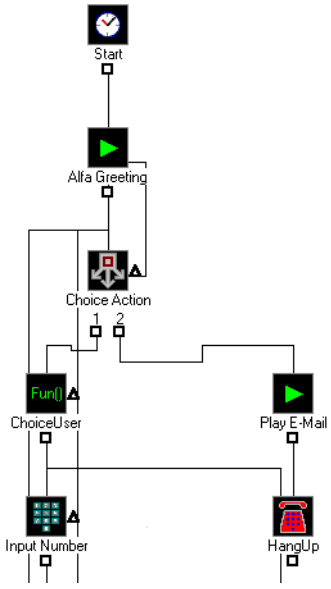
In the field “Number“ enter the user ID of the person in the mail system.

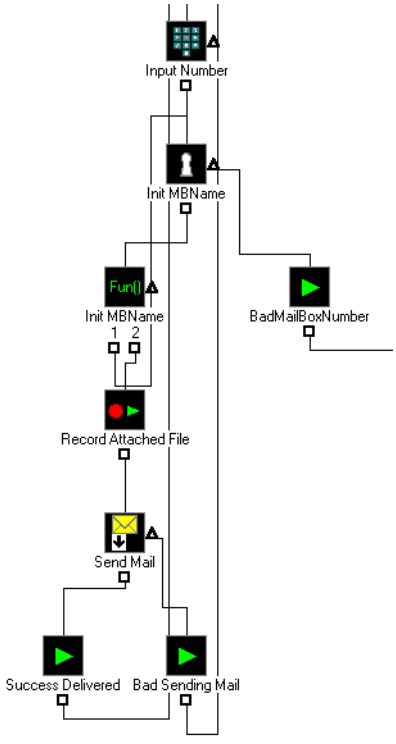
In the field “PIN” enter the user’s PIN number.

In the field “Address” type in the full addressee name of the person as it is identified in your mail system (the mail system must be able to determine the email address from the name entered; e.g. Smith, John or John Smith depending on how the name is defined in your mail system). Alternatively, the full email address of the user may be entered.

Please note that before the functions in step 10 can be successfully executed, an appropriate file from which to read email messages must be specified in the Play Box designated for playing email messages. See Step 10 Part 3 below.

Step10.vap Part 1



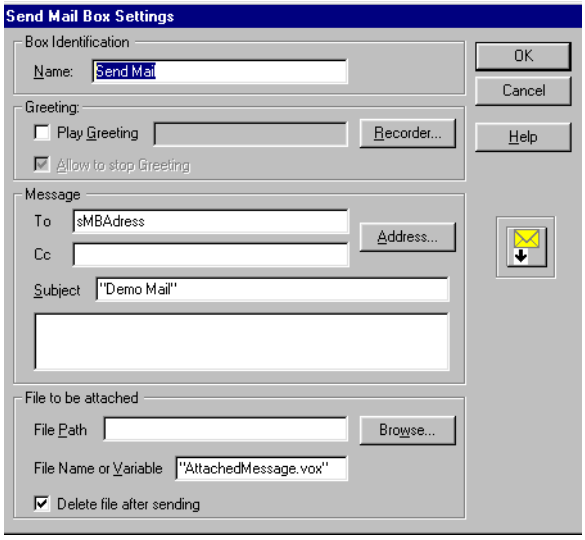


After calling Smartphone Server, the user hears a greeting and is prompted with a choice of sending a voice message via email or listening to his email messages.

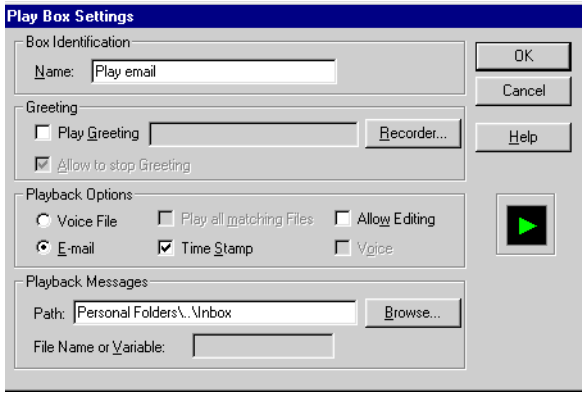
Step10.vap Part 2

If the caller presses 1 he chooses to send a voice message, and is prompted to enter the user ID of the recipient. After the caller has entered the user ID of the intended recipient, Smartphone Server searches the database for the email address of the specified user. If the user ID was not found, a Play Box is used to inform the caller that an incorrect number was entered and the caller will be asked to re-enter a number If the search was successful, Smartphone Server records the message and uses the Sendmail Box to send it to the recipient.

Following is an example of the Settings for the Sendmail Box. In this example, the variable for the user address to be taken from the database is entered in the “To” field. Thus, if a user ID was entered and a match found in the database, the voice message will be sent to the user’s email address. By pressing the “Address” button, an address can be chosen from an Address Book. The name of the attached voice file should be entered in the “Name or Variable” field. If the “Delete file after sending” checkbox is selected, the voice file will be deleted after having been successfully sent.



Within the Play Box Settings, Email is selected under "Playback Options."

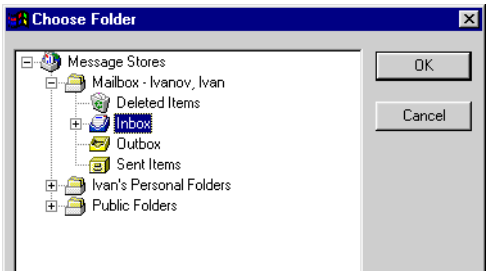


Once the message was successfully sent, the caller hears an announcement indicating that the message was delivered. If the message could not be sent successfully, the caller will hear a message to that effect.

Step10.vap Part 3

Upon completion of sending the message, the caller may return to the menu and choose to send another message by pressing 1 or listen to his own emails by pressing 2. If the user chooses to read his emails, the Play Box will play back all messages from his Inbox:

In the "Path" field the folder from which the email messages will be read is specified. Select the Inbox or any other available folder with messages to be played by pressing the "Browse" button. This brings up the "Choose Folder" dialog from which you can select any of the available folders.



After playing all email messages in the specified folder, the user can end the call.

Chapter 3: Writing Voice Applications

Designing a VAP

Operation of the VAP Editor

VAPs Management

VAP Editor is a visual programming desk for creation and editing Smartphone Server Voice Applications (VAPs). Each VAP Editor window (see a picture above) allows to edit one VAP. There may be none, one or many editing windows opened.

To create a new (blank) VAP Editor window, choose [File | New].

To open a VAP for editting in VAP Editor, choose [File | Open].

To save a VAP you are currently editting (its window must be active), choose [File | Save].

To close a VAP Editor window, click on icon at the top of the window.

VAP Editor window

VAP Editor window (see a picture above) consists of

- **window title**, which contains the filename of the VAP,
- Boxbar, which contains all box templates available in this version of Smartphone Server,
- Program design area, where VAP body is located.

Editting a VAP

VAP creation procedure includes:

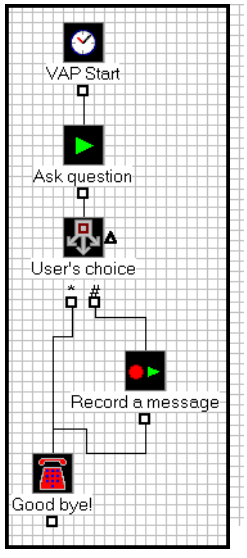
1. adding boxes: the boxes can be moved by means of the Drag&Drop technique on the program design area from the Boxbar,
2. **editing box settings:** after adding a box to the VAP Editor window, you must edit the box settings,
3. **making connecting lines:** you can create lines by clicking (with the mouse) on the exit of a box and dragging the line to the next box (still keeping the mouse pressed). The lines can be deleted by clicking the exit of the box where the connection starts.

Each VAP must contain at least one Start Box.

It's easy to change VAP structure after all boxes are already in the program design area. You can use the clipboard to copy, cut, paste, and delete one or more boxes with connecting lines. Select multiple boxes by holding the **<SHIFT>** key and clicking on the boxes. This allows connected groups of boxes and their connecting lines to be moved within the program design area.

To finalize VAP design, you can run "Auto Arrange" tool.

Voice Application (VAP)



Arbitary task of building telephone system can be successfuly resolved in Smartphone Server VAP Editor. VAP Editor is a visual programming desk for creation and editting Smartphone Server Voice Applications (VAPs). A VAP is a graphic program (flow chart) which controls PC telephone boards.

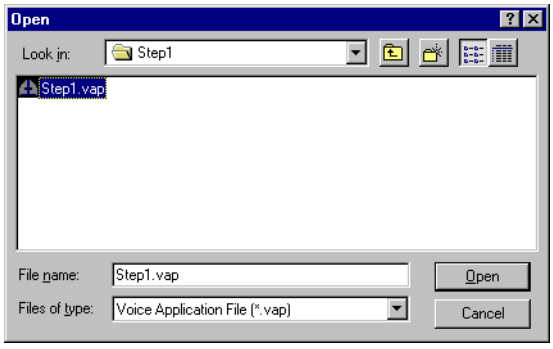
The process of VAPs creation is fast and easy because of visual methods provided with VAP Editor. The process of executing VAPs is also simple because Smartphone Server interprets Voice Applications in run-time, without compiling them.

Voice Applications are stored in files with extension ".VAP". You can create new VAP, open and edit or save an existing VAP in VAP Editor.

Voice Applications can be started or stopped in Setup Lines dialog. One VAP operates one telephone line. You can cause several lines to be operated by the same VAP.

The structure of Voice Applications is similar to graph which consists of functional blocks called “boxes” united with connecting lines. Every box has a special purpose, some settings which allow you to meet your needs, one entry point and one or more exits.
Connecting lines start from box exits and finish at boxes entry points. Several connecting lines can finish at the same box.
Each box is represented by the icon. Each type of a box has a special icon. You can look at all box types of this version of Smartphone Server by viewing the Boxbar. Each box can be made separate by assigning the unique identification name which is displayed under the box icon.
VAPs have also some advanced settings which can be viewed and changed in [Settings | Voice Application Settings] dialog.
Each VAP must contain at least one Start Box.

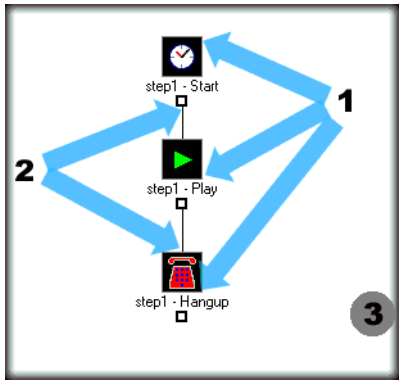
Loading an Example Voice-Application



Select [File | Open]. Select Step1.vap in the subfolder ‘C:\Program Files\Smartphone Server\Examples\Step1’. Click **Open**. The VAP is loaded and displayed in the VAP Editor. The VAP can now be started in the Setup Lines dialog.

The use of the VAP Editor

Smartphone Server is really a graphic programming language with which programs controlling a PC phone board can be written. Such a program is called voice application or VAP. A VAP bears a strong resemblance to a flow chart because Smartphone Server provides a visual programming desk called the VAP Editor.

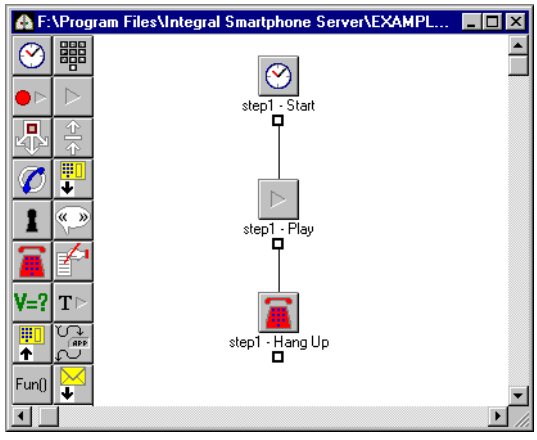


How to load an existing application into the VAP Editor:

Select “Open” in the file menu.

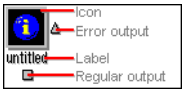
Choose the file step1.Vap from the Smartphone Server directory.

The following window opens:



We program by composing the programming elements, the so called boxes on the visual programming desk (the graphic user interface).). All the boxes inside the programming desk form the voice application (VAP). The program illustrated above consists of three boxes which have been titled. The terminology of boxes is illustrated in the following picture:

Terminology of the boxes



The icon is a symbol which represents a box in the programming of Smartphone Server. The Box list contains all boxes which are available to the user (different icons stand for different functions). Each box has a designation (a name) and one or several exits.

Connecting lines

These exits join a box with the box immediately following in the program run. So, in 1step.vap, the boxes are executed one after the other from top to bottom.

The user creates links by clicking (with the mouse) on the exit of a box and dragging the line to the next box (still keeping the mouse pressed). The lines can be deleted by clicking the exit box where the connection starts.

The lines, which lead away from the selected box, are better indicated in color.

Using the boxes

The following section explains, how you can work with the different boxes.

Drag&Drop

The boxes can be moved by means of the Drag&Drop technique on the graphic user interface (the VAP editor window). It allows for to “drag”, place the mouse on the symbol, press the mouse, and, while still pressing the mouse down, move it to the desired position. To “drop”, simply release the mouse.. Use the same method to add new boxes—drag it from the box list and release it onto the programming window or desk.

Selecting more than one box at a time:

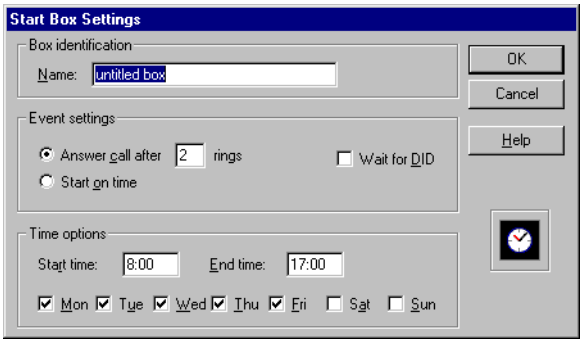
A group of boxes can be moved at the same time by pressing the Shift key. This allows connected groups of boxes and their linking lines to be moved within the programming desk.

Cut and paste

Both single boxes and groups of boxes can be cut out and copied. This allows the arrangement of the box or boxes to be repeated easily within the same application , or, in a new application.

Settings

Although the program has now been visually designed, it is not yet a complete program. A box is only an element which now needs to have its parameters set. In order to set the parameters of a box, double click on the box, and the setting window will appear. The example below shows the ‘Start Box’ setting window.



Now the individual functions, meanings and uses can be entered.

Editing a VAP

This chapter illustrates how to change existing VAPs, how to deal with them and how to handle Smartphone Server. How to create VAPs will be dealt with in Part IV.

Security Passwords

Each VAP can be protected by a password. This prevents any unwanted changes in the applications. To do this, simply mark the function “Open as read only” when saving an application..

Prompts and System prompts

In computer telephony jargon, a “prompt” is a digitally recorded file which contains instructions to an computer telephony application user. It is basically a short dictation consisting of greetings, instructions, tunes, and so on. With this definition, any recorded voice is a prompt.

Smartphone Server discerns two kinds of prompts. The first, the so-called greetings, are recordings made by the user, stored by Smartphone Server in a *.vox or *.wav file each.

The second kind of prompts are the so-called System prompts. They are pre-defined by the system and are used for internal announcements.

An example of an internal announcement is a request to confirm an input. If the option confirm input of the input box is on, a caller will be played back his or her own input. In other words. If someone types in “3-4-1”, he or she will hear: “Your input is three-four-one” The single system prompts played back to the caller are, in this case:

- “Your input is”
- “Three”
- “Four“
- “One”

These system prompts are available already upon installation of Smartphone Server. They are included in several languages, and can be found in the directory:

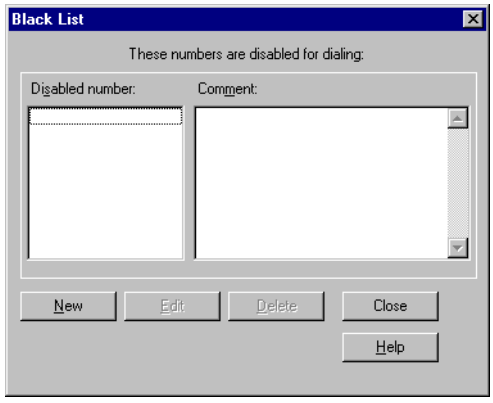
C:\Program Files\Smartphone Server\sysprmts\L1; L2; etc.

Every language has their own prompts: L1=German, L2=English, etc. Further information about the available System prompts is found in the appendix, aptly under System prompts.

Disabling Phone numbers

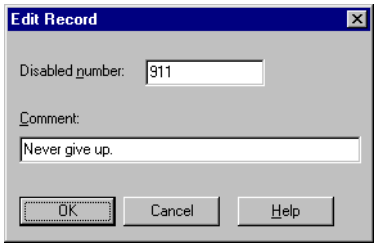
With the “Blacklist” it is possible to disable dialing certain numbers and groups of numbers in Smartphone Server. If such a number is tried, Smartphone Server will react as if the call is not answered. The Blacklist can be configured for each VAP individually.

Choose the command “Blacklist” from the menu settings and the following window will appear:



Now enter the number, or numbers, to be disabled. Please note that Smartphone Server disables all numbers which begin with the given codes. For example, if all calls from the Switzerland to other foreign countries are to be disabled, enter 00. All calls which begin with 00 are then treated by Smartphone Server as if the phone does not pick up. Any remarks may be entered in the space to the right.

To make a new entries in the Blacklist, click on the button “New”. It opens the following window:



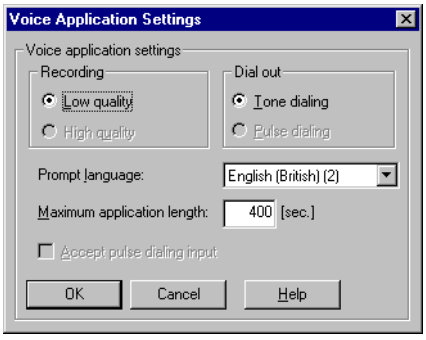
Enter the new number to be disabled and the remark, and click on the button “OK”.

Voice Application Settings

The Voice Application settings always refer to the VAP currently being worked on.

Choose the command “Voice Application Settings” from the menu. Settings. If no VAP was loaded in the VAP Editor, this command can not be carried out. Since Smartphone Server supports different hardware, only the set-

tings relevant to your voice board can be established. The following Dialogue window will appear:



The settings of this dialogue window are:

Recording Quality: Here the recording quality is set by ticking in the desired setting.

Dial Out: Choose which dialing Smartphone Server should use within the VAP.

Language of System prompts: Select which language is to be used for the system prompts.

Maximum Length of Application [sec] This field specifies the maximum time a call should be given; the example of 400 seconds would mean that after 400 seconds, the call is terminate should there be no user input.

Accept Pulse Dialing: This determines whether or not Smartphone Server should detect pulse dialing input (proper hardware needed).

For each application, the recording quality, the language of the prompts, the dialing method, etc. can be set.

How to amend the language of the System prompts:

1 Choose the command “Voice Application Settings” from the menu settings.

2 Select the desired language.

3 Close the window by pressing the “Ok” button.

4 Save the VAP.

5 Activate the VAP again (if desired).

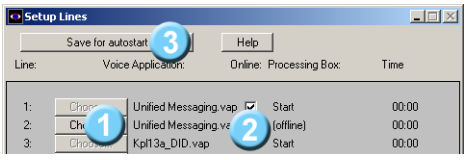
Activating a VAP

The command [File | Setup Lines] in the menu opens the “Setup Lines” window. Since Smartphone Server can manage numerous lines simultaneously, each line must be assigned a specific VAP. After the application has been selected, Smartphone Server loads the VAP file in the memory in mere seconds, checks it and assigns it to the line. From this moment, the VAP is ready to work and handle incoming or outgoing calls.

To activate a VAP:

Choose the command [File | Setup Lines] from the menu.

Depending on the number of licensed lines, a dialog similar to the following will be displayed:



Click **Choose**. Hold the <Alt> key to apply the selection to all lines.

Select the desired VAP in the “Open” dialogue that now appears on the screen.

Repeat the previous two steps for each line to be assigned a VAP.

Click **Online** to activate the desired VAP for each telephone line. Hold the <Alt> key to activate VAPs on all lines.

The System Windows

After the VAPs are activated, the system windows can be seen. These are important elements in the Smartphone Server system, the so called “management team” of Smartphone Server. They show which lines are active, what is happening on each line, and much more.

The system windows can not be closed. If the displays are not wanted, they can be minimized. The system windows render the invisible background activities of the current VAPs visible, thus offering information so a user can see what Smartphone Server is doing at any given time.

The Status Window

The Status Window shows the present state of the lines; it displays whether a line is online, how long it has been online, and what box a caller is currently carrying out. The main task of the status window is to survey the temporary status of the entire system.

Autostart

Smartphone Server offers an autostart option. When the program is started, it automatically loads the saved configuration of the telephone lines and goes online with them.

How to use the Autostart option:

1. Set up the exact configuration of telephone lines that is wanted in autostart.
2. Click the button “Save for autostart”.

When restarted, the selected configuration will be restored.

The main application of this option is to cope with external interference and errors. After a system crash due to power failures or other reasons, the computer can start up again, load, and carry out the same functions as before.

For further information about setting up a full autostart configuration see “Smartphone Server 3.1. Getting Started”.

The Log Window

The Log Window is a monitor. All important activities which Smartphone Server carries out are protocolled here. The Log Window also indicates mistakes in the use of a VAP.

Troubleshooting

Although an application is activated (online), Smartphone Server fails to answer calls. Possible sources of error are:

The time period specified in the start box does not match the current time. Check the defined time range and correct if necessary, as well as the time showing on the PC.

Activating a line takes a few seconds. If a call comes in during this short time, Smartphone Server is unable to answer, and this may cause problems to arise later. In such cases, deactivating the line concerned and then reactivating it is recommended.

An amended and stored application was not activated again. Smartphone Server does not re-load the VAP until you take it offline and put it online again. Check to make sure that any changes which have been made have been entered and that the VAP has been correctly activated again.

Chapter 4: Expressions and variables

Expressions

Expressions contain operators and operands. An expression is evaluated to a single value that is assigned to a variable.

The allowed operands in an expression are:

- variables constants
- function calls

* other expressions

Parentheses can be used to explicitly define the order of precedence of an expression within another expression.

Smartphone Server expressions can be used in most fields in Box settings dialogs and in Script Language routines.

Note: *You can use Expression Builder to create any Smartphone Server expression.*

Reserved Words

Reserved words are used by Smartphone Server to name built-in objects. A user-defined variable cannot have the same name as a reserved word.

There are some types of reserved words:

- Script Language statements
- Smartphone Server functions
- Smartphone Server system variables

Expression Builder topics:

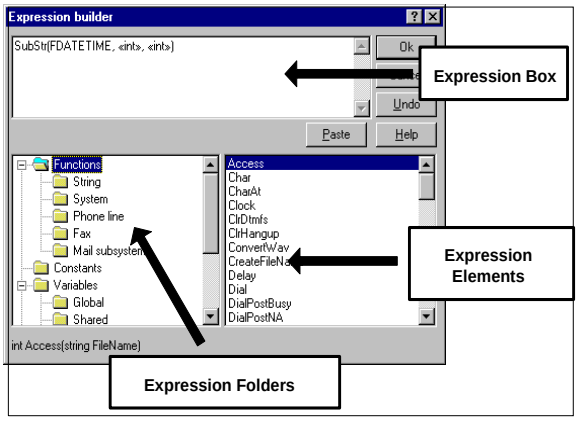
Expression Builder

- Read about Expression Builder
- Start Expression Builder
- Insert a new expression using Expression Builder
- Read more about expressions

About Expression Builder

In order to make it easier for you to create expressions in Script Language, Smartphone Server includes the Expression Builder – a dialog that puts all of the expression elements at your fingertips.

Expression builder is divided into three sections:



Expression Box The expression itself is shown in the expression box, located at the top of the window. You can type the expression directly in this window, but for your convenience, Expression Builder categorizes all of the expression elements in the lower left box and lists the elements associated with each category in the lower right box.

Expression Folders All expression elements can be categorized into functions, constants, variables, or operators. These categories are further broken down. The lower left box contains the various folders into which the expression elements have been categorized, allowing you to quickly determine which element you are looking for.

Expression Elements Once you have opened the necessary file, choose the particular element you want to add to your expression in the lower right box. Double-click on the element or highlight it and press **Paste.** The element

will be added to your expression in the place where the cursor was located. Hit **Undo** at any time to undo the last action you made.

Note: *If an element or whole folder isn't listed on Expression Builder, that is because it is not valid in the context where you started the Builder. For example, if you open Expression Builder in a field called Variable, only the Variables folder will be listed in the Expression folder box, since this is the only valid type of element for this field.*

To learn more about Expression Builder: Click on the ? active help button within Expression Builder.

Expression Box

Displays the expression you are building or modifying.

Expression Folders

To find the expression elements you need, click on the corresponding folder.

Expression Elements

The full list of all expression elements in the particular category.

Inserting an Expression With Expression Builder

To construct a new Expression and insert it into your Script Language routine:

1. Open Expression Builder. How? See Start Expression Builder.
2. In the Expression Folder window, choose the category and then the subcategory containing the element that you want to insert.
3. In the Expression Element window, double-click on the particular element you want to insert or highlight the element and press **Paste.** The element appears in the Expression Box.

Note: *You can also type any element directly into the Expression Box.*

4. When your expression is complete, click **OK.** Your new expression will appear in the Script Box Settings dialog.

Starting Expression Builder

You can start Expression Builder from the Settings dialog of any box when you want to enter an expression in a field. The Script Box is a box especially designed for writing Script Language.

To open Expression Builder from any box:

1. Double-click on the box to open its Settings dialog.
2. Place the cursor on the field in which you would like to enter an expression and click on the right mouse button.
3. Choose Expression Builder from the menu.

To open Expression Builder from Script Box:

1. Double-click on Script Box.
2. In the Script Box Settings dialog, place the cursor in the exact place in the script window where you want to insert an expression and click on the Build button.

Expression operators

Operators are special symbols that represent an operation to be performed on the operands in an expression.

In the following example:

```
2 + 3;
```

The operator is '+'

The operands are the numeric constants 2 and 3

The expression is "2+3"

The following is the full list of available operators:

Arithmetic operators:

+	addition (numeric or string operands)
-	subtraction
*	multiplication
/	division
%	mod

Unary arithmetic operators:

+	unary plus
-	unary minus

Relational operators:

==	equal to

!= not equal to

> greater than

< less

>= greater than or equal to

<= less than or equal to

Bitwise operators:

& bitwise AND

| bitwise inclusive OR

^ bitwise XOR (exclusive OR)

~ bitwise unary negation

>> shift right

<< shift left

Logical operators:

&& logical AND

|| logical OR

! unary logical negation

Assignment operators:

= assignment

+= assign sum

-= assign difference

*= assign product

/= assign quotient

%= assign remainder (modulus)

>>= assign right shift

<<= assign left shift

++ increment

-- decrement

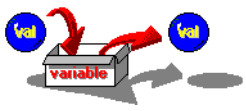
Other operators:

<exp> ? <st1> : <st2>

ternary operator (if <exp> is true then <st1>, else <st2>)

, (comma) evaluate

Variables



A variable is an identifier which represents a value that can change. For example, variables are useful for working with file names that are not constant (that are assigned according to a caller's name, etc.)

The following rules should be observed when creating a variable name:

- The name should be descriptive and unique.
- Variable names are not case-sensitive (abc1 and ABC1 are equivalent variable names).
- Reserved words cannot be used as variable names.
- The variable name contains only letters and digits (32 maximum; first character is a letter).

There are 3 levels of variable scope:

- **Shared:** Shared variables can be accessed by several VAPs. They are used to exchange data between VAPs. Shared variables are defined in the VAP variables / Shared variables dialog.
- A shared variable is stored in a file with the extension ".spv". Each VAP which has access to the necessary .spv file has shared-level access to all variables in that file.
- **Global:** Global variables can be accessed by any box in the VAP. Global variables are defined in the VAP variables / Global variables dialog.
- **Local:** Local variables can be accessed only in the routine in which the variable is defined. The value of a local variable is only defined while the Script Box is executing.

There are four types of variables:

- **Float:** Floating-point numbers (for example: 3.142, 0.001). Equivalent in C, C++, Pascal, Delphi: *double.*
- **Int:** 32-bit integers (-2147483648 to +2147483647). Equivalent in C, C++: *long*; in Pascal, Delphi: *longint.*
- **String:** Strings of **any** length.

- **Date:** Full-format date and time records (for example: 1997-10-22 19:30:19). System variable FDATETIME provides the current date and time in the appropriate format. Date variables cannot be local.

Variables can be assigned the following types of constants:

- **Float:** Decimal value.
- **Int:** The following types:
    - **Decimal**.
    - **Octal** (with the prefix "0" (zero); for example: 015 (13 decimal)).
    - **Hexidecimal** (with the prefix "0x" (zero + "x"); for example: 0x15 (21 decimal)).
- **String:** Any characters enclosed in quotation marks (“); for example: "stringstring".

Certain prefixes are used to represent the value of a variable converted into another type:

$: Conversion from Int / Float /Date to String: $123 = "123".

#: Conversion from String to Int: #"24" = 24.

@: Conversion from String to Date: @fdatetime.

Note: *Use the Float function to convert from String to Float: Float("3.1415") = "3.1415".*

Note: *Conversion from Float to Int, and from Int to Float, is automatic.*

VAP variables: Global

This dialog displays a list that contains:

- Global variables for the current VAP.
- System variables (cannot be edited).

Now global variables can be inherited from the parent Voice Application when the Call Application Box is used to call a sub-VAP. If the checkbox "Inheritance" is checked in the "New/Edit Variable" dialog, this variable will be inherited from the parent VAP to this application and/or the child VAP.

The following commands are available in the dialog:

Add a Global variable:

1. Click on ***New***.
2. The ***"New variable"*** dialog appears.
3. Enter the name of the new variable in the *'Name'* field
4. Select a type (*String, Int[eger], Float, Date*).
5. Click on ***OK***.

Change the name or type of a variable:

Click on ***Edit***. Click on ***OK***.

Delete a variable:

Click on ***Delete***. Click on ***OK***.

VAP variables: Shared

The following functions are available from this dialog:

- Create an .spv file.
- Link the current VAP to any ".spv" file.
- View or modify the variables in an .spv file.

To create a new ".spv" file:

1. Click ***Browse...***.
2. Select a directory for the ".spv" file
3. Enter the file name in the field ***File name***.
4. Click ***OK***.

To link an ".spv" file:

1. Click **Browse...**
2. Select the .spv file. A list of variables is displayed.
3. Click **OK**.

To view or modify the variables in an .spv file:

1. Click ***Browse...***
2. Select the .spv file. A list of variables is displayed.

To add a new Shared variable

1 Click on ***New.***

2 The ***"New variable"*** dialog appears.

3 Enter the name of the new variable in the *'Name'* field

4 Select a type (*String, Int[eger], Float, Date*).

5 Click ***OK***.

To change the name or the type of a variable

1 Click on ***Edit*** button.

2 Change the name or type.

3 Click ***OK***.

To delete a variable:

1 Click on ***Delete***.

2 Click ***OK***.

System Variables

System variables are defined by Smartphone Server. System variable names cannot be used to name other variables.

System variables can be:

- Added to any expressions.
- Set within certain boxes in a VAP.

There are 2 types of system variables:

- *Read Only* (*r/o*) (cannot be changed by a VAP)
- *Read/Write* (*r/w*) (can be changed by a VAP)

List of all system variables:

Table 1:

ANI	r/o
CALLNAME	r/w
CALLNUMBER	r/o
DATE	r/o
DATETIME	r/o
DID0, DID1, DID2, DID3	r/w
DNIS	r/o
FDATE	r/o
FDATETIME	r/o
FTIME	r/o
LANGUAGE	r/w
LASTDBRECORD	r/o
LASTERRORINT	r/o
LASTERRORSTR	r/o
LASTWARNSTR	r/o
LASTRECORDFILE	r/o
LASTTEXTOUTFILE	r/o
LINENUMBER	r/o
NODIALTONE	r/o
RECORDCOUNT	r/o
REDIR	r/o
REDIRTYPE	r/o
TIME	r/o
TRANSFERCALLDURATION	r/o
__Event	r/o
_FaxHeader	r/w
_FaxCallerId	r/w
_FaxRemoteId	r/o

NODIALTONE(r/o system variable)

Indication whether or not a dial tone was detected on the line.

0 - a dial tone was detected

1 - no dial tone was detected

TRANSFERCALLDURATION(r/o system variable)

The duration in seconds of the call that was transferred.

LASTERRORINT(r/o system variable)

Contains the number of the last internal application error (due to hardware or software related problems). This integer is written to the log file only when its value changes, i.e. the error pin of a box was taken as the exit.

0 - no error occurred

See also LASTERRORSTR

LASTERRORSTR(r/o system variable)

Contains a description of the last internal application error that occurred (due to hardware or software related problems). This string is written to the log file only when the string changes, i.e. the error pin of a box was taken as the exit.

See also LASTERRORINT

LASTWARNSTR(r/o system variable)

Contains a description of the last warning which occurred due to incorrect user input or insufficient data to complete the box functions.

ANI(r/o system variable)

Variable ANI contains the received Automatic Number Identification digits which specify the number of the calling party.

See also DNIIS, REDIR

CALLNAME(r/w system variable)

The name of the Voice Application that is currently running.

The value of the CALLNAME variable is NOT changed when another VAP is started with the Call Application box within the “parent” VAP. However, this system variable is writable and the “sub” VAP can change the CALLNAME value depending on the flow of the VAP.

The CALLNAME system variable is used to identify the VAP in the statistics database. In this database the name of the VAP, on-line time and the phone line number are stored.

The statistic information is saved at the moment the operation flow is switched from a Hangup box (with normal or global hangup) to the next box in the sequence of the original “parent” VAP. If there is no box attached to the Hangup box, statistic information is not saved.

If there is a chain of 2 or more VAPs that called one another: the value of the CALLNAME variable is taken from the last active VAP that executed the Hangup box. If the VAP finished without execution of the Hangup box, the value of the CALLNAME variable set in this VAP will not be saved.

See also CALLNUMBER

CALLNUMBER(r/o system variable)

The number of the call on this line.

See also LINENUMBER

DATE(r/o system variable)

The current date (month, day, year). The exact format of this string variable is set within Windows NT/2000/XP.

Example: 10/22/2001.

See also FDATE

DATETIME(r/o system variable)

The current date and time (the combined variables **DATE** and **TIME**).

Example: 10/22/2001 07:30:19 PM.

See also FDATETIME

DNIS(r/o system variable)

Variable DNIS contains the received Dialed Number Identification Service digits which specify the number of the called party.

This variable is currently supported on ISDN boards only.

See also ANI, REDIR

DID0, DID1, DID2, DID3(r/w system variable)

The components of the last DID received by Smartphone Server from the PABX. The names of these components can be changed by changing the definitions in the Smartphone Server “smphone.ini” file, section [DIDSettings]).

FDATE(r/o system variable)

The current date (year-month-day) in a fixed, standard format, used for working with databases and comparing dates (e.g., checking whether two dates taken from a database record are equal).

Example: 2001-10-22.

See also DATE

FDATETIME(r/o system variable)

The current date and time (the combined variables **FDATE** and **FTIME**).

Example: 2001-10-22 19:30:19.

See also DATETIME

FTIME(r/o system variable)

The current time (hours, minutes, seconds) in a fixed, standard format, used for working with databases and comparing the time stamps.

Example: 19:30:19.

See also TIME

LANGUAGE (r/w Systemvariable)

Set the language of all system prompts in a VAP:

0 - user defined language

1 - German

2 - English

3 - French

4 - Italian

6 - Russian

LASTDBRECORD(r/o system variable)

Number of the last accessed record within the last accessed DBaseIII database.

See also RECORDCOUNT

LASTRECORDFILE(r/o system variable)

Path/file name of the last file recorded in a Record Box.

See also LASTTEXTOUTFILE

LASTTEXTOUTFILE(r/o system variable)

Path/file name of the last text file used for output in a Textout Box or Input Box.

See also LASTRECORDFILE

LINENUMBER(r/o system variable)

Number of the line the current VAP is operating on.

See also CALLNUMBER

RECORDCOUNT(r/o system variable)

Number of records in the record set.

See also LASTDBRECORD

REDIR(r/o system variable)

Variable REDIR contains the received Redirecting number information digits which specify the number of the party that transferred the call.

This variable is currently supported on ISDN boards only.

Note: *Some ISDN service providers do not provide REDIR information.*

See also ANI, DNIIS

REDIRTYPE(r/o system variable)

Variable REDIRTYPE contains the type of redirection of the current call:

0 - no redirection;

1 - redirection on busy;

2 - redirection on no answer;

3 - unconditional (permanent) redirection.

This variable is supported on ISDN boards only.

Note: *Some ISDN service providers do not provide REDIRTYPE information.*

See also REDIR

TIME(r/o system variable)

The current time (hours, minutes, seconds). The exact format of this string variable is set within Windows NT/ 2000/XP.

Example: 07:30:19 PM.

See also FTIIME

__Event(r/o system variable)

Allows VAP to monitor critical events for the connection on the telephone line.

FINAL - the remote party has disconnected.

PhysicalLineDown - the connection to the PBX has been physically interrupted (for ISDN boards only).

PhysicalLineUp - the physical connection to the PBX has been restored (for ISDN boards only).

_FaxHeader(r/w system variable)

String for the fax header to send. Assign the value before fax send functions (in Script Language only).

See also _FaxCallerId, _FaxRemoteId.

_FaxCallerId(r/w system variable)

Caller ID string for the fax to send. Assign the value before fax send functions (in Script Language only).

See also _FaxHeader, _FaxRemoteId.

_FaxRemoteId(r/o system variable)

Stores a remote ID string (opposite fax machine identifier) after outgoing or incoming fax session has been established (in Script Language only).

See also _FaxCallerId, _FaxHeader.

Chapter 5: Script Language Basics

Script Language (SL) is a high level interpreted language (similar to HAL) supported by Smartphone Server. The complete functionality of any Smartphone Server box can be generated with SL. The syntax of **SL** statements is very similar to the syntax of C statements. However, user-defined functions, which are supported in C, are not supported by SL.

A single SL routine consists of one or more statements in a single Script Box.

All Smartphone Server expressions can be used in an SL routine. For example:

```
a = b+32;
counter++;
```

To make your SL programs more readable, comments can be added. There are two ways to add comment text:

Enclose text on any number of lines with braces and star symbols (“ /* */ “). For example:

```
/* This is
a comment.*/
```

Insert double braces “//” before any comment text on a single line. For example:

```
int var1;// This is a comment.
```

Script routine execution starts with the first statement in the Script Box.

Script routine execution stops when one of the following occurs:

- A ‘return’ statement is executed.
- The last statement in the Script Box is executed.

Shared or global variables are accessible from any Script Box.

Local variables are accessible only from the **Script Box** in which they are defined.

For more detailed information about the Script Language see:

- Expressions
- Expression operators
- Variables
- System variables
- Functions
- Statements

Script Language Statements

A **statement** represents an action to be executed by Smartphone Server. A Script Language routine consists of one or more statements.

There are two types of statements:

Singular: Singular statements end with a semicolon. There are two types of singular statements:

Reserved words: Some reserved words can be used to define loop, variable definition and branch statements. For example:

```
return;
```

Expressions: Expressions can also be used as singular statements. For example:

```
a += b*c;
```

Compound: Compound statements contain one or more statements enclosed in brackets (‘**{ }**’). For example:

```
{
   BoxOut(decision);
   if( decision>3 ) break;
}
```

The following is the full list of Smartphone Server Script Language **statements**:

Table 1:

Loop statements:	Variable definition statements:
break continue for while	float int string
Branch statements: if...else return switch...case...default	

Functions

Smartphone Server expressions and variables can include the following functions:

Table 2:

File management functions:	Branching functions:
Access Fcopy FMerge Frename CreateFileName Unlink	BoxOut WinExec
String functions: Char CharAt StrLen StrStr StrUpr SubStr	**Date and time functions:** Clock Day WeekDay
Dialing functions: CanDial ClrDtmfs Dial DialPostBusy DialPostNA	**Fax functions:** FaxAdd FaxSend FaxClear FaxGetCount FaxGetFile FaxGetPhone FaxCheck FaxPrint FaxPrintOK FaxReceive
Input functions: Record GetDigit GetDigits	**Output functions:** Playback PlayPrompt PlaySys Speak

Table 2:

Off hook/on hook status functions:	Initial files functions:
ClrHangup IsHangup IsOffHook Hangup SetHangup	GetProfileInt GetVMIInt GetVMIString WriteVMIInt WriteVMIString Write
Prompts management functions: GetPromptFile GetSysFile Wait	**Variables functions:** GetVar SetVarNum SetVarStr
Tracking functions: Log MessageBox MsgBox	**Other functions:** GetDefLang Delay Sleep Random

Table 2:

Database access functions:	**OLE functions:**
Connect Disconnect Execute Drop Fetch Get Set Error	Create PropGet PropSet Invoke AddRef Release
List management functions: Detach	**Date and time management functions:** Get function Add function Sub function ToStr function
SMS functions: SMS_ send	**Call VAP functions:** load_ VAP call_ VAP

TransferCall

TransferSetParam

Old functions

The following three functions are left for backward compatibility. It is not recommended to use them in new applications. Their functionality is fully supported by these new functions: Random, StrLen, SubStr.

Table 3:

rnd	Random integer number.	rndN (N=0..9)	a = rnd9 = 3
strlen	Returns the length of a string variable	strlen(str_var)	a = strlen(line) = 4
substr	Extracts a sub-string from string variable.	Substr(source, start_pos, not_more_chars)	mid = sub-str(whole, 5, 4) = e

Namespaces

Custom Script language functions may be added to the standard Script language by simply adding a reference in the Windows registry to a DLL file that defines and implements the custom functions. These custom functions can be displayed in the Smartphone Server Expression Builder.

Expression Builder groups the available components for Script language expressions into standard groups (Functions, Constants, Variables, and Operators) and the custom Namespaces group. In Expression Builder, the custom functions are contained within the Namespaces group or in a subgroup of the Namespaces group.

Any group or subgroup in Namespaces that does not contain any subgroups represents a unique **name space**. A **name space** refers to a group of variables and functions that have the same scope. All names contained within a single name space must be unique. Therefore, the same function name may appear in different function groups, but only once in any one functional group.

Creating a namespace .dll file.

Installing a namespace file on your computer.

Namespaces: Creating namespace .dll files

Consult C++ programming software documentation on how to create DLL files with the software you have.

To make Smartphone Server detect your DLL, the following function must be exported:

```
// Copyright(c) 2000, NOVAVOX AG

extern "C" __declspec(dllexport) NAMESPACE* __stdcall GetNameSpaces(void);
```

The following text is C++ header file that must be included in the program project of the developer's DLL:

```
#ifndef __nmspace_h
#define __nmspace_h

#if !defined(_INC_WINDOWS)
```

```
  #define STRICT
  #include <windows.h>
#endif

#include <pshpack1.h>

//
//
//
HKEY_LOCAL_MACHINE\SOFTWARE\NOVAVO
X\Smartphone Server\Script\extensions
//
//

struct VALUE{
  int type;
  union{
    long       l;
    double     f;
    char       *pStr;
    void       *pv;
    SYSTEMTIME dt;
  };
};

class IScriptCallback{
public:
  virtual void* __cdecl Alloc(unsigned
long ulSize)=0;
  virtual void  __cdecl Free(void*)=0;
  virtual void  __cdecl
FreeValue(VALUE*)=0;

  virtual VALUE __cdecl Pop(void)=0;
  virtual void  __cdecl Push(VALUE)=0;

  virtual VALUE __cdecl
GetVarValue(VALUE* Var)=0;
  virtual void  __cdecl
SetVarValue(VALUE* Var, VALUE Value)=0;

  virtual HWND  __cdecl
GetWindow(void)=0;

  virtual int   __cdecl
GetCurrentLine(void)=0;
  virtual void  __cdecl GetVAPName(char
*Buf, int SizeBuf)=0;
};

typedef void (__cdecl
*PFSCRIPTFUNC)(IScriptCallback*);
typedef int  (__cdecl *PFNSENTRY)(int
Line, int Reason, IScriptCallback*);

struct FUNCTION{
  const char   *szName;
  PFSCRIPTFUNC  pfExec;
  int          iRetType;
  const char   *szParams;
  const char   *szDefine;
  unsigned int uiHelpId;
};

struct PROPERTY{
  const char *szName;
  int type;
  union{
    long value;
    char *pc;
    long *pl;
     PFSCRIPTFUNC pfRead;
  };
  PFSCRIPTFUNC pfWrite;
  long data;
  unsigned int uiHelpId;

  PROPERTY(const char *_szName, long
_value, unsigned int _uiHelpId=0, long
_data=0)
  :szName(_szName), type('n'),
value(_value), uiHelpId(_uiHelpId),
data(_data)
  {
  }
  PROPERTY(const char *_szName, char
*_pc, unsigned int _uiHelpId=0, long
_data=0)
  :szName(_szName), type('s'), pc(_pc),
uiHelpId(_uiHelpId), data(_data)
  {
  }
  PROPERTY(const char *_szName, long
*_pl, unsigned int _uiHelpId=0, long
_data=0)
  :szName(_szName), type('r'), pl(_pl),
uiHelpId(_uiHelpId), data(_data)
  {
  }
  PROPERTY(const char *_szName, int
_type, PFSCRIPTFUNC _pfRead,
PFSCRIPTFUNC _pfWrite, unsigned int
_uiHelpId=0, long _data=0)
  :szName(_szName),
type(_type | 0x8000), pfRead(_pfRead),
pfWrite(_pfWrite), uiHelpId(_uiHelpId),
data(_data)
  {
  }
```

```
};

#define DEFINE_FUNCTION_TABLE(name)
FUNCTION name[]={
#define END_FUNCTION_TABLE  {NULL}}

#define DEFINE_PROPERTY_TABLE(name)
PROPERTY name[]={
#define END_PROPERTY_TABLE
PROPERTY(NULL, 0L)}

#define DEFINE_NAMESPACE_TABLE(name)
NAMESPACE name[]={
#define END_NAMESPACE_TABLE {NULL}}

struct TFunction;
struct TConstant;

enum{
  NS_INIT,     //load dll
  NS_UNINIT,   //free dll

/*
  NS_STARTVAP, //start VAP on line
  NS_STOPVAP,  //stop VAP

  NS_CALL,     //new call
  NS_HANGUP,   //hangup
*/
};



struct NAMESPACE{
  const char  *szName;
  union{
    FUNCTION  *pFunctions;
    TFunction *pFunc;
  };
  union{
    PROPERTY  *pProperties;
    TConstant *pConst;
  };
  const char  *szHelpFile;
  PFNSENTRY pfNSEntry;
};

typedef NAMESPACE* (__stdcall
*GETNAMESPACES)(void);

#include <poppack.h>

#endif
```

Contact NOVAVOX AG for up-to-date information about creating your own custom DLL files for custom namespaces.

Namespaces: Installing on your computer

See also

To install a custom namespace file on your computer:

1. Exit Smartphone Server.
2. Copy the namespace .dll file to any directory on your computer (the recommended directory is the Smartphone Server directory (by default C:\Program Files\Smartphone Server)).
3. Start the Windows registry editor (From the Windows Start menu: Select [Start / Run] and enter “regedit”).
4. Go to “My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\NOVAVOX\Smartphone Server\script\extensions”.
5. Click on “extensions”.
6. Right-click on “extensions”. From the menu select: [New | String Value].
7. In the “Name” box: A new key appears. Right-click on the key and select “Modify”.
8. In the “Value data:” field: Enter the the path and filename of the namespaces .dll file.
9. Click OK.
10. Close the registry editor.
11. When you start Smartphone Server and open the Expression Builder dialog, the new namespace(s) will appear.

Chapter 6: Script Language Examples

Examples for SL statements

This example shows how to use while, break, Dial, Clock

This SL routine attempts to dial a number. If the phone number is successfully dialed or 1 minute (60,000 msec) expires, the routine ends.

The variable "dialed" contains the result of the dialing.

```
int dialed = 0;
int TimeStart;

TimeStart = Clock();

while( Clock()-TimeStart < 60000 )
{
   dialed = Dial( "514000", 5 );
   if( dialed == 0 )
   break;
}
```

This example shows how to use for, continue, FaxCheck, Speak

This SL routine checks for the fax option on all (64 maximum) phone lines. If the fax option exists on a line, a message is played. The number of phone lines supporting fax is also computed.

```
stringinp;
intfaxlines = 0;

for( int line=1; line <= 64; line++ )
{
   if( FaxCheck( line ) == false ) continue;
\\ THE_LINE.vox text = "The line number"
\\ MAY_BE.vox text = " may be used for sending and receiving fax documents."
   Speak( "\pTHE_LINE.vox \n"+$line+" \pMAY_BE.vox" );
   faxlines++;
}
```

This example shows how to use int, float, IsOffHook

This SL routine calculates the percentage of the first 8 telephones lines that are busy.

```
intbusylines = 0, totallines = 8, ASCII_code = ‘C’;
```

```
floatbusypart, busypercent=0;

for( int line=1; line <= totallines; line++ )
   if( IsOffHook( line ) == true) busylines++;

busypart = busylines / totallines;
busypercent = busypart * 100;
```

This example shows how to use if…else, MessageBox, Write

This SL routine uses a Message Box to ask whether to write a string to the end of a text file in the current Windows NT/2000/XP directory (using the Write function) and announces the results in a Message box.

```
// Generate a message box with the question mark icon, the OK & Cancel buttons.
if( MessageBox( "Press 'OK' to attempt to write a string to the text file, 'Cancel'
to quit.", "Script example", 0x00000021 ) == 1 )
{

// Attempt to append a string to the text file:
   if( Write( "Example.txt", "This sample illustrates how the Write function in
Smartphone Server Script language works." + Char(13) + "This test was performed on
"+$fdatetime+"."+ Char(13) ) != 0 )

// If the attempt was successful: Generate message box with the OK button.
   MessageBox( "The string was successfully appended to the file 'Example.txt'.",
"Script example", 0x00000000);
   else

// If the attempt was unsuccessful: Generate message box with the hand icon and the
OK button.
   MessageBox( "An error occurred while writing to the text file 'Example.txt'. The
string was not appended.", "Script example", 0x00000010 );
}
```

This example shows how to use return, ClrDtmfs, BoxOut, GetDigit, Record

This SL routine allows the user to press a telephone key to record a voice file. The Script box exit taken depends on the user input.

Note: *The Script Box must have a minimum of 2 exits.*

```
intExitYes = 1, ExitNo = 2;
intUserInput = 0;

// Play prompt: 'Press 1 to leave a message, 2 to quit.'
while( UserInput != ExitYes && UserInput != ExitNo )
{
   ClrDtmfs();
   Playback( "Question.vox" );
   UserInput = #GetDigit( 10, 1 );
```

```
}
BoxOut( UserInput);

// If 2 pressed: Exit immediately:
If( UserInput == ExitNo )
   return;

// If 1 pressed: Record message:
ClrDtmfs();
Record( "FromUser.vox", 1, 60, 5 );
```

This example shows how to use switch, MsgBox, FaxAdd, FaxClear, FaxSend

This SL routine dials the phone number "813000" and states the call status. If the call is:

answered by a person (voice): a greeting is played.

busy: a busy prompt is played.

not answered: a no answer prompt is played.

answered by a fax machine: a text file is sent to the fax machine (any errors are reported).

answered by any answering machine: an appropriate prompt is played.

If any error occurs: an appropriate prompt is played.

```
switch( Dial( "813000", 10 ) )
{
// If call is answered: Play greeting:
   case 0:
   Playback( "Greeting.vox" );

// If phone is busy: Play message:
   case 1:
   MsgBox( "The phone is busy." );

// If phone not answered: Play message:
   case 2:
   MsgBox( "The phone was not answered." );

// If phone answered by fax machine: attempt to add fax to queue and transmit queue.
   case 3:
   {
   FaxClear();
   if( FaxAdd( "ToSend.txt" ) == 0 )
   MsgBox( "Cannot add 'ToSend.txt' file to the fax queue." );
   if( FaxSend( ) == 0)
   MsgBox( "Cannot send fax document." );
   }

// If phone answered by auto-answering machince: Play message:
   case 4:
   MsgBox( "The call was answered by auto-answering machine." );
```

```
// If none of the above occurred: Error occurred:
   default:
   MsgBox( "Error has occurred" );
}
```

This example shows how to use string, Char, CharAt, StrLen, SubStr

This SL routine demonstrates how to work with strings. In this example, all carriage returns (ASCII 13) in a string are replaced with tab symbols (ASCII 9). The string is then displayed in a message box.

```
string SourceText = "Sample text."+Char(13)+"This example shows how some string
functions work in Smartphone Server."+Char(13)+"NOVAVOX AG", DestText;

MsgBox( "Source string is:"+Char(13)+SourceText );

intSLength = StrLen( SourceText );
DestText = "";
for( int symb=0; symb <= SLength; symb++ )
{
   if( CharAt( SourceText, symb ) == 13 )
   DestText += Char(9);
   else
   DestText += SubStr( SourceText, symb, 1 );
}

MsgBox( "Destination string is:"+Char(13)+DestText );
```

This example shows how to use Access, Fcopy, Fmerge, Unllink

This SL routine concatenates the contents of file "Text1.txt" to the end of the same file if the file exists in the same directory as the Script Box VAP.

```
if( Access( "Text1.txt" ) != 0 )
{
   FCopy( "Text2.txt", "Text1.txt" );
   FMerge( "Text1.txt", "Text2.txt" );
   Unlink( "Text2.txt" );
}
```

This example shows how to use ClrHangup, Hangup, IsHangup, IsOffHook, SetHangup

This SL routine demonstrates how to use SL to "pick up" or "hang up" a phone line. The status of the phone line can be checked in 2 different ways:

Check IsHangup: A quick method that may sometimes give an incorrect value.

Check IsOffHook: A slower method that checks hardware directly and always give a correct value.

The following example demonstrates how the Hangup attribute is cleared and how the functions IsHangup and IsOff-Hook can possibly return different values.

```
Hangup();
MsgBox( "User disconnected. Hangup attribute set." );

ClrHangup();
MsgBox( "Hangup attribute cleared. No hangup is registered in Smartphone Server. This will
now be verified…" );

if( IsHangup( ) != 0 )
   MsgBox( "Function IsHangup (checks variable) indicates that the call is
disconnected (on hook)." );
else
   MsgBox( "Function IsHangup (checks variable) indicates that the call is still active
(off hook)." );

if( IsOffHook( 0 ) == 0 )
   MsgBox( "Function IsOffHook (checks hardware) indicates that the call is
disconnected (on hook)." );
else
   MsgBox( "Function IsOffHook (checks hardware) indicates that the call is still
active (off hook)." );

// Set 'hang up' attribute:
SetHangup();
```

This example shows how to use CreateFileName, Log

This SL routine creates a new file. The filename is displayed in the LOG window. Then a voice message is recorded and saved to this file.

```
stringNewMessage;

NewMessage = CreateFileName( ".\", "MSG" );
Log( "New filename is created: " + NewMessage );

Record( NewMessage, 1, 45, 5 );
```

This example shows how to use Day, WeekDay

This SL routine writes:

the number of the current day (integer) to variable "CurDayInt" (integer).

the number of the current day (integer) to variable "CurDayStr" (string).

the contents of the "date" system variable (string) to variable "CurDate".

the number of the current day of the week (integer) to variable "DayOfWeek".

```
intCurDayInt, DayOfWeek;
stringCurDayStr, CurDate;
```

```
CurDayInt = Day( );
CurDayStr = $CurDayInt;
CurDate = date;
DayOfWeek = WeekDay();
```

This example shows how to use Delay, Sleep

This SL routine attempts to call a phone number. Up to 5 attempts are made to complete the call, with a 2-minute pause between each attempt ("Delay(300)" creates a 1-minute delay, "Sleep(60)" creates a second 1-minute delay).

```
for( int cc = 0; cc < 5; cc++ )
{
   if( Dial( "514 000", 8 ) == 0 )
   break;
   Delay( 300 );
   Sleep( 60 );
}
```

This example shows how to use Playback, DialPostBusy, DialPostNA

This SL routine demonstrates how to redirect calls. It first plays a waiting message. Then it uses a flash hook (with the "&" character) to dial the phone number 666-000. If the called number is:

Answered by voice: A hello prompt is played. The SL routine hangs up. At this point, the PBX connects the caller to the called party and the VAP stops execution immediately.

Busy: Smartphone Server reconnects to the calling party and states that the phone is busy.

Not answered: Smartphone Server reconnects to the calling party and states that the phone was not answered.

```
Playback( "WaitNow.vox" );
switch( Dial( "&666 000", 10 ) )
{
   case 0:
   {
   Playback( "Hello.vox" );
   Hangup( );
   }
   case 1:
   {
   Dial( DialPostBusy(), 3 );
   Playback( "IsBusy.vox" );
   }
   case 2:
   {
   Dial( DialPostNA(), 3 );
   Playback( "NoAnswer.vox" );
   }
   default:
   {
```

```
   Dial( "&,", 3 );
   Playback( "NotVoice.vox" );
   }
}
```

This example shows how to use FaxGetCount, FaxGetFile, FaxGetPhone

This SL routine displays the number of faxes in the fax queue and the filename for each fax in the queue.

```
intfaxdocs = FaxGetCount( );

if( faxdocs == 0 )
   MsgBox( "No fax documents in the queue." );
else
{
   MsgBox( "There are "+$faxdocs+" fax document(s) to be sent to fax number
"+FaxGetPhone() );
   for( int faxc = 0; faxc < faxdocs; faxc++ )
   MsgBox( "Document #" + $(faxc+1) + ": file `" + FaxGetFile(faxc) + "'." );
}
```

This example shows how to use FaxPrint, FaxPrintOK, FaxReceive

This SL routine will receive a fax document and save it as a TIFF/F file. The fax file will then be printed.

```
stringFaxFile = "FaxTest.tif";

Playback( "NowStart.vox" );
if( FaxReceive( FaxFile ) != 0 )
{
   if( FaxPrintOK() != 0 )
   FaxPrint( FaxFile );
}
```

This example shows how to use Frename

This SL routine renames a prompt file with the ".vox" extension as a backup file with the ".bak" extension. It then records a new file with the same filename as the original prompt file.

```
stringPromptFile = "Message";

if( Access( PromptFile+".vox" ) != 0 )
   FRename( PromptFile+".bak", PromptFile+".vox" );

Record( PromptFile+".vox", 0, 30, 5 );
```

This example shows how to use GetDefLang

This SL routine displays the following system language settings in the LOG window:

- Voicemail/Unified Messaging language.
- VAP language (from system variable GetDefLang(1))
- VAP language (from system variable LANGUAGE)

```
Log( "Voicemail/Unified Messaging language: " + $GetDefLang(0) + "; VAP language: " +
$GetDefLang(1) + "; LANGUAGE system variable: " + $LANGUAGE );
```

This example shows how to use GetDigits, Random

This SL routine plays the names of 2 single digits (selected randomly). The routine then asks for the caller to press the same digits on the telephone keypad. The routine then states if the entered digits match the random digits.

```
stringsOK = "*", sCANCEL = "#", test, UInput;

test = $Random( 10 ) + $Random( 10 );

ClrDtmfs( );
\\ GREET1.VOX = "The numbers are:",
\\ GREET2.VOX = "Press the same numbers on your phone keypad after the beep."
\\ RES1.VOX = "The original numbers and the numbers you entered matched."
\\ RES2.VOX = "The original numbers and the numbers you entered did not match."
Speak( "\pGREET1.vox \n" + test + "\pGREET2.vox" );

UInput = GetDigits( 2, 10, sOK + sCANCEL, 1, sCANCEL );
if( test == Uinput )
   Playback( "RES1.VOX" );
else
   Playback( "RES2.VOX" );
```

This example shows how to use GetProfileInt

This SL routine checks the entry "EnableConnectDigit" in the section "Speech" in the file smphone.ini. If 1 (Continuous Speech Recognition enabled): an error message is played (this feature is not supported by Smartphone Server).

```
int
ContRec;

ContRec = GetProfileInt( "smphone.ini", "Speech", "EnableConnectDigit", 0 );
if( ContRec != 0 )
```

MsgBox("'EnableConnectDigit' entry is set to 1 in smphone.ini file. This enables Continuos Speech Recognition usage in Input Box. Note that this option is NOT supported by Smartphone Server. ");

This example shows how to use GetPromptFile, PlayPrompt, Wait

This SL routine plays all Voicemail/Unified Messaging English daytime prompts.

```
for( int PromptN=0; Access( GetPromptFile( PromptN, 2, 0 ) ) != 0; PromptN++ )
{
   Wait();
   PlayPrompt( PromptN, 1 );
}
```

This example shows how to use GetSysFile, PlaySys

This SL routine displays the filename for each English single-digit numeric prompt (prompts for digits 0 - 9) and plays the prompts.

```
intOldLang;

OldLang = language;
language = 2;
for( int SysP = 0; SysP < 10; SysP++ )
{
   Log( "Playing back the following system prompt: `"+GetSysFile( $SysP, 2 )+"'" );
   PlaySys( $SysP );
}
language = OldLang;
```

This example shows how to use GetVar, SetVarNum, SetVarStr

This SL routine increments the variable "CallsLineX" (where "X" is the number of the current phone line) and sets the variable "NameLineX" to "John Brown".

Note: *The variables "CallsLineX" must be created before running this example.*

To run this example:

- Create a VAP for line 2 that contains:
- Start Box with with "Start on time" option selected.
- Script Box with the following text:

```
while( true )
   Log( "Line #1. Calls: "+$CallsLine1+", user name: "+NameLine1 );
```

- Start the VAP on line 2.
- Create a VAP for line 1 that contains:
    - Start Box with with "Start on time" option selected.
    - Script Box with the following text:

```
intCurCalls;
stringCallerName = "John Brown";

CurCalls = #GetVar( "CallsLine"+$linenumber );
SetVarNum( "CallsLine"+$linenumber, CurCalls+1 );

SetVarStr( "NameLine"+$linenumber, CallerName );
```

- Start the vap on line 1.

This example shows how to use GetVMIInt, GetVMIString, WriteVMIInt, WriteVMIString

This SL routine reads and writes the following entries in section "Voicemail/Unified Messaging" in the "config.vmi" file:

- Entry "LangEnable".
- Entry "DataSource".

Note: *This example does not change any settings in "config.vmi" file.*

Be extremely careful when changing entries in configuration files.

```
intvLangEnable;
stringvDataSource;

vLangEnable = GetVMIInt( "Voicemail/Unified Messaging", "LangEnable" );
WriteVMIInt( "Voicemail/Unified Messaging", "LangEnable", vLangEnable );

vDataSource = GetVMIString( "Voicemail/Unified Messaging", "DataSource" );
WriteVMIString( "Voicemail/Unified Messaging", "DataSource", vDataSource );
```

This example shows how to use StrStr, StrUpr

This SL routine checks for the existence of a sub-string in a string and displays the results in the log window.

```
stringSentence = "To the faithful departed.", Word = "DEPARTED";

if( StrStr(StrUpr(Sentence), StrUpr(Word) ) == -1 )
   Log( "The word was not found in the sentence." );
else
   Log( "The word was found in the sentence." );
```

This example shows how to use WinExec

This SL routine:

Starts the file C:\Windows\Notepad.exe in a maximized (3) window.

Executes the command line "notepad C:\Windows\Readme.txt" (opens the file C:\Windows\Readme.txt in Notepad) in a normal (1) window.

Note: *If necessary, change the directory to the directory where Notepad.exe is located on your computer.*

```
WinExec( "C:\Windows\Notepad.exe", "", 3 );

WinExec( "", "notepad C:\Windows\Readme.txt", 1 );
```

Example for Database functions

This example shows how to use Database access functions (Connect, Disconnect, Execute, Drop, Fetch, Get, Set, Error)

This SL routine uses the sample MS Access 7.0 database included with the Smartphone Server package. The Connect function refers to this database by full path and file name, so the standard "C:\Program Files\Smartphone Server" sub-string in the second parameter passed to the function should be changed to the actual path where your Smartphone Server is installed in order to run this example routine in a Script Box.

On successful connection, this routine counts the records in a call statistics database table. If connection to the database fails or the table contains no records, the routine terminates its execution and outputs an error diagnostic message to the Smartphone Server Log Window.

If records are found for 1 or more calls, the calls summary data is collected for each telephone line and transferred to a new summary table (this table should not be present in the database; if found at this point it has to be deleted, and this routine must be restarted). Once the summary table has been successfully created and filled in, a new column is added to it. Then the calls duration data is collected from the main calls statistics table and summarized for each telephone line separately, providing data to fill in the new column in the summary table. When complete, the summary data from the new table is output to the Smartphone Server Log Window.

Finally, the summary table created for the purposes of this example is deleted from the database, and the connection to the database is closed.

```
string EOL=char('\r')+char('\n'), TAB=char('\t');
string RecCount="", serr="";

//Establishing a connection to a database
//(change the path according to your Smartphone Server installation location)
if(!db::connect("C:\Program Files\Smartphone Server\Examples\RepDemo\sp_demo.mdb"))
  {
   Log(EOL+"Error connecting to SP_DEMO.MDB..."+EOL);
   return;
  }
else
  {
   Log(EOL+"Connection to SP_DEMO.MDB successful..."+EOL);
  }
//Counting records in the 'Calls' table
db::execute(0,"SELECT Count(*) AS RecCount FROM Calls;");
db::fetch(0,db::Go_First);
```

```
Log("Your demo database contains records of "+RecCount=db::get(0,"RecCount")+"
calls"+EOL);
db::Drop(0);
if(RecCount==0)
  {
   Log("Database is empty! Exiting..."+EOL);
   return;
  }
//Summarizing line usage statistics in a new table 'Summary'
if(!db::execute(0,"SELECT Count(*) AS CallsCount, Line, Min(Time) AS First, Max(Time)
AS Last INTO Summary FROM Calls GROUP BY Line;"))
  // If the test table was not deleted in a previous execution of this VAP due to an
error:
  {
   Log("Error "+$db::error(serr)+": "+serr);
   db::execute(-1, "DROP TABLE Summary;");
   Log("'Summary'table was unexpectedly found in the database and deleted.");
   Log("*** Please restart this VAP. ***");
   return;
  }
db::drop(0);
//Adding a column for total calls duration and filling it in
db::execute(-1,"ALTER TABLE Summary ADD COLUMN Duration NUMBER");
db::execute(0,"SELECT Duration, Line FROM Summary;");
while(db::fetch(0,db::go_next))
  {
   db::execute(1,"SELECT Sum(Duration) AS Total FROM Calls HAVING
Calls.Line="+db::get(0,"Line")+";");
   db::fetch(1,db::go_first);
   db::set(0,"Duration",db::get(1,"Total"));
   db::Drop(1);
  }
db::Drop(0);
//Outputing data from the 'Summary' table
db::execute(0,"SELECT * FROM Summary ORDER BY Line ASC");
Log("Line usage statistics:"+EOL);
Log(TAB+"Line"+TAB+"Calls"+TAB+"Duration"+TAB+TAB+"First"+TAB+TAB+"Last");
Log(TAB+"-------------------------------------------------------------------------
--------------------------");
while(db::fetch(0,db::go_next))
  {

Log(TAB+db::get(0,"Line")+TAB+db::get(0,"CallsCount")+TAB+db::get(0,"Duration")+TAB+
db::get(0,"First")+TAB+db::get(0,"Last"));
  }
db::Drop(0);
//Deleting the 'Summary' table from the database
db::Execute(-1,"DROP TABLE Summary;");
db::Disconnect();
return;
```

Example for OLE functions

```
string EOL=char('\r')+char('\n');


Log("Step 1: Creating MS Outlook Application");

int hObj=auto::Create("Outlook.Application");
if(hObj == 0)
{
  LOG("Error Creating Outlook App."+EOL+"Exiting…"+EOL);
  return;
}

Log("Step 2: Creating a Journal Item in MS Outlook");

int OurItem = auto::Invoke(hObj, "CreateItem", 4);
if(OurItem != 0)
{
  Log("Step 2: success");
  auto::PropSet(OurItem, "Body", "Hello, it's SL example on line "+$LINENUMBER+"
"+DATETIME);
  auto::PropSet(OurItem, "Subject", DATETIME);
  auto::PropSet(OurItem, "Companies", "NOVAVOX AG");
  auto::PropSet(OurItem, "Type", "Smartphone Server Entry");
  auto::Invoke(OurItem, "Save");
  auto::Release(OurItem);
}

Log("Step 3: Creating a Journal Item in MS Outlook");

OurItem=auto::Invoke(hObj, "CreateItem",  6);
if(OurItem != 0)
{
  Log("Step 3: success");
  auto::PropSet(OurItem, "Body", "Hello, POST ITEM from line "+$LINENUMBER+"
"+DATETIME);
  auto::PropSet(OurItem, "Subject", DATETIME);
  auto::PropSet(OurItem, "Companies", "NOVAVOX");
  auto::Invoke(OurItem, "Post");
  auto::Invoke(OurItem, "Save");
  auto::Release(OurItem);
}

Log("Step 4: Creating MS Outlook Mail message");

OurItem=auto::Invoke(hObj, "CreateItem",  0);
if(OurItem != 0)
{
  Log("Step 4: success");
  auto::PropSet(OurItem, "Body", "from Smartphone Server on line "+$LINENUMBER+"
"+DATETIME+eol);
```

```
  auto::PropSet(OurItem, "Subject", "go
home");
  auto::PropSet(OurItem, "Companies",
"NOVAVOX");
  auto::PropSet(OurItem, "To",
"[smtp:sp_testV@novavox.ru]");
  int nAtt=auto::PropGet(OurItem,
"Attachments");
  auto::Invoke(nAtt, "Add", _Prompt(1),
1, -1, "prompt.vox");
  auto::Release(nAtt);
  auto::PropSet(OurItem, "MessageClass",
"IPM.Note.VVVoicemail.Voice");
  auto::Invoke(OurItem, "Send");
  auto::Release(OurItem);
}

auto::Release(hObj);
```

Chapter 7: Automatic Speech Recognition (ASR)

How automatic speech recognition functions

The Speech recognition option allows the caller to give information by voice instead of using the telephone keypad.

Smartphone Server recognizes both words from a defined vocabulary, as well as self-defined words. To create a new vocabulary, use the program LexTool included in Smartphone Server software package. The integrated speech recognition is based on phonetic recognition, which means that the system recognizes given words without training. It compares the incoming voice picture to the automatically generated voice picture from the vocabulary. Smartphone Server can support up to 1000 words.

Smartphone Server has powerful speech recognition capabilities, based on the matching of speech with vocabularies (word sets with pronunciation information).

Speech recognition is supported for the following languages:

1 - German

2 - English (UK)

3 - French

4 - Italian

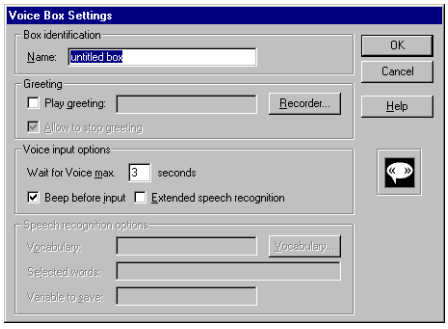
10 - English (US)

The **Lexicon Toolkit** (included with Smartphone Server) can be used to modify existing vocabularies or create new vocabularies for software-based recognition.

Usage

Speech recognition can be used in two ways. First, in the Voice box to save the input as a variable; second, in the standard boxes to give direct tasks.

The Voice box



Voice input options

Wait for Voice max. ... seconds: The amount of time entered here determines how long Smartphone Server should wait before setting forth the application by taking the exit.

Beep before input: Should you want a signal tone to sound before the caller gives in any input, mark this field.

Extended Speech Recognition: This field lets you know if the speech recognition has been activated or not.

Speech Recognition options

Vocabulary: Choose the vocabulary required. The path can be chosen by clicking on Vocabulary.

Selected Words: Choose the required words from the vocabulary.

Variable to Save: Voice input can be saved in a Smartphone Server-Variable and used in other applications.

Words Selector

The **Words Selector** supports custom speech recognition vocabularies that are a subset of system-provided contexts.

For each system language there are 2 vocabularies: Cx and Lx (where x is the number of the language; see system variables for more information about available languages). Cx contains only digits, while Lx contains digits and single words. Multiple digits can be recognized if Cx is specified; only single digits or words can be recognized if Lx is specified.

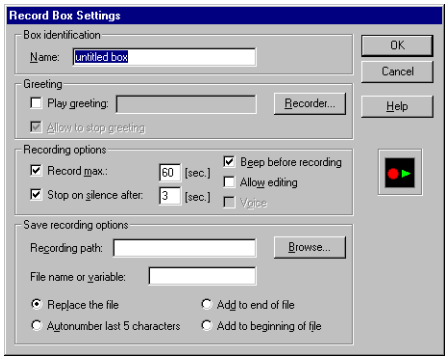
Speech recognition in other boxes

Using the following boxes enables the caller to choose the form of input—either with the keypad or by voice:

ASR in Play Box

When the field “Allow Editing“ is marked, the caller has the chance to listen again to the recorded message or to delete it, by pushing the keyboard or by saying the appropriate number of the system prompt.

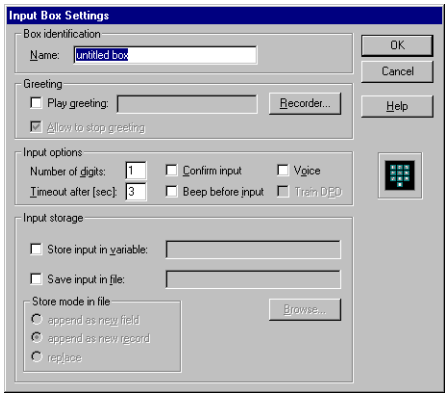
ASR in Record Box



When the field “Allow Editing “ is marked, the caller has the possibility to listen again to the recording, to confirm, or to repeat it.

If “Voice” is marked, the input can be given either by using the keyboard or by saying the appropriate number provided by the systems prompt.

ASR in Input Box



If the field “Voice” is marked, Smartphone Server accepts vocal input as well. Each input is repeated. “Confirm Input” should be marked to give the caller the possibility of repeating or correcting the given input.

There are two voice input possibilities within the input box: individual words as well as appropriate numbers are recognized.

The art of speech recognition to be used can be set in the file ...\windows\smphone.ini. Open the file with an editor and go to item:

[Speech]

EnableConnectDigit=0

The items mean the following:

EnableConnectDigit=0: individual words will be recognized

EnableConnectDigit=1: appropriate numbers will be recognized. This option uses “continuos recognition” which is not supported by Smartphone Server.

Choose the desired art of recognition and save the data by going into menu File and clicking on save.

In both cases the caller can give input by numbers or keyboard.

Individual words In this mode the caller gives input number by number. The system answers with “your input was [Number]“ after every accepted input. If the input is not recognized, Smartphone Server will answer with” Please repeat entry“. After the third incorrect or unsuccessful try, Smartphone Server will branch off to the right exit. The input can be given by saying “Stop“ or by coming to Timeout time (Timeout after...) ending.

Connected digits In this mode, the caller has the possibility of saying all numbers after each other. The system will recognize up to 25 numbers.

This option uses “continuos recognition” which is not supported by Smartphone Server.

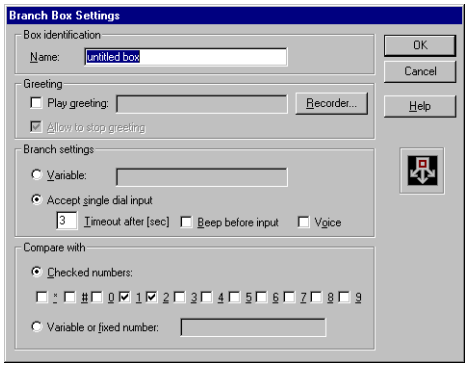
For example: To use the entry 123, the caller says “1, 2, 3“. The recording ends when the limited time of allowed quietness within the speech box has been reached. (Timeout after...)

Key word recognition

It is also possible to work with key words. Here, individual key words are recognized from within sentences.

For example: The key words are Smith and Jones. The caller will be asked to give the name of the person he/she wishes to speak to. The caller says: “I would like to speak to Mr. Smith please”. Smartphone Server recognizes the word Smith and branches to the person accordingly.

ASR in Branch Box



When the fields “Accept single Dial Input” and “Voice” are marked, the caller can give input by voice or by keyboard.

Use Your Own Words

Delivered with the Smartphone Server speech recognition option is a program that enables you to add your own vocabulary to the existing word list. The program (Lexicon Toolkit) can be found in the installation of chosen program groups or in the standard Smartphone Server folder.

Start the program by choosing Lexicon Toolkit Application in the start menu in the Smartphone Server folder.

For further information about the Lexicon Toolkit see “Lexicon Toolkit Overview” chapter.

The Phonetic Alphabets

Table 1: British English

Phonetic Symbols	Pronounced as in
I	beat
I	bit
E	bed
@	map
A	car
A+	pot
^	but
O	bought
U	book
u	boot
$	about
E0	turn
I&$	here
E&$	there
U&$	pour
e&I	bait
O&I	boy
a&I	buy
a&U	down
o&U	show
j	you
w	wit
R+	ride
l	let
p	pan
t	tan
k	can
b	boy
d	day
g	got

Table 1: British English

?	(glottal stop)
f	fine
T	thin
s	sin
S	shine
v	vine
D	that
z	zone
Z	vision
h	head
t&S	church
d&Z	jungle
m	my
n	no
Nk	sing

Table 2: German

Phonetic Symbols	Pronounced as in
I:	Riese
y:	grün
I	Milch
Y	Küste
e:	Kehle
e+	Öl
E	letzte
E+	löschen
a	Stadt
a:	Wagen
O	voll
o:	groß
U	Kunst
u:	Fuß
$	Taste

Table 2: German

O&y	heute
a&i	Teil
a&u	Baum
j	jemand
l	Licht
R	Reise
p	Post
t	Tinte
k	klein
b	Bein
d	dich
g	liegen
?	(glottal stop)
f	Feld
s	Fels
S	Schnee
v	wach
z	Saal
Z	Journal
C	Milch

Table 2: German

x	Bach
h	Hand
p&f	Pferd
t&s	Zug
t&S	klatschen
d&Z	Gin
m	Mann
n	Norden
Nk	Ring

To view phonetic alphabets for other languages supported by Smartphone Server please consult the appropriate topics (‘Phonetic Alphabet for French’, etc.) in the Lexicon Toolkit online help (distributed with the product).

Special signs

Table 3:

‘	primary stress
‘2	secondary stress
.	syllable boundary
-	word boundary
#	silence (pause)

Chapter 8: Databases

Smartphone Server provides access to databases via ODBC version 3.0, and supports all database functions provided by the appropriate database driver (and includes also the driver's limitations). For detailed information about operations supported by a driver, refer to the ODBC driver documentation.

Two methods of accessing databases from voice applications can be used:

- Script language database functions (recommended for advanced users)
- Database box (supports simple operations, left for compatibility with the previous versions)

The script language fully provides full SQL support. All advanced databases are supported, including MS SQL Server 6.5, 7.0 and Oracle 8.0.

It is not recommended to access Oracle databases from voice applications if ASR support is installed on your Smartphone Server.

For more information about using script database functions and Database box see "Databases Access Functions" on page 155 and "Database Box" on page 111 in this manual.

Introduction to ODBC

ODBC Sources

Use Smartphone Server's ODBC32 (MDAC) package for Windows NT/2000/XP to create ODBC sources which can be accessed directly by a voice application. ODBC is a powerful tool that provides access to databases via abstract database references. ODBC sources can also be managed in the Windows Control Panel.

SQL is used for accessing ODBC sources. Standard packages supported by *ODBC32* include X/Open and SQL Access Group SQL CAE specification (1992) and is a subset of the emerging ISO SQL-92 standard. Consult ODBC online Help (installed with *ODBC32*) to get differences and extensions between this standard and actual specification.

Making a new ODBC source

More detailed explanations can be found in the Windows Manual.

Buttons

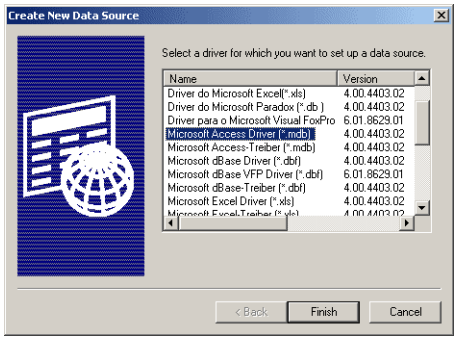
Close: Closes the window and takes over the chosen driver.

Help: Shows the online help available for this window.

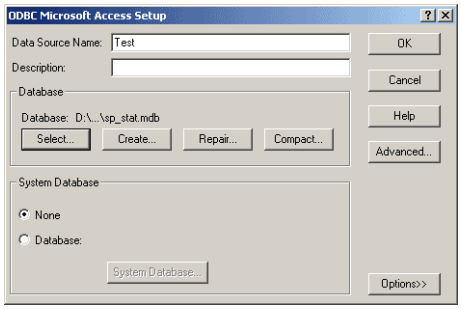
Setup: The ODBC-sources are entered here. Explanations regarding the Setup-window are found further on at **Add**.

Delete: Eliminates an ODBC-driver from the systems configuration.

Add: Use this button to add an ODBC-source. The following window will appear and ask which driver the new source should work with. Choose the desired driver; should the desired driver not be listed, click on **Cancel** and click on **Drivers**....



Once the driver has been chosen, the following window appears (only valid for MS Access-drivers; the setting window for other ODBC-sources could look a bit different):

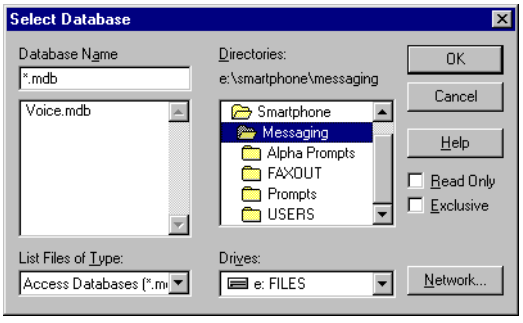


Data Source Name: Name of the data source

Description: Comments referring to the Data source.

Database: Select: Choose the appropriate database.

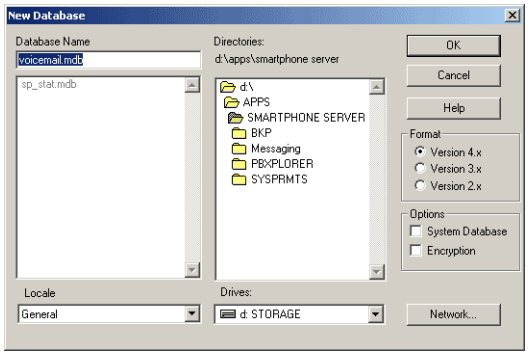
Using existent database



Choose the database and click on “OK”.

Creation of a new database

Create...: Click on Create...to make a new database:



Enter the name of the new database and click on OK.

Other ODBC options

Repair...: Click on this button to repair a database in MS Access Format.

Compact...: Click here to compact an MS Access-Database. First, enter in the name of the file to be compacted. Second, enter in the name under which the compacted file should be saved. Click on “OK” and the file will be compacted.

Please refer to the MS Windows manual for further information about ODBC sources and settings.

Database box

Database compatibility

The **Database Box** provides access to dBase III databases with direct interface, MS Access 2.0 and 7.0 databases via ODBC interface and any other ODBC sources installed on your system. NOVAVOX AG guarantees only MS-Access compatibility.

The following database formats were tested with Smartphone Server using Microsoft ODBC version 3.0 drivers:

Table 1: Databases compatibility

Database format:	Test results:	Additional comments:
MS Access 7.0, 97, 2000	OK	You can use MS Access 97 files choosing “MS Access 7.0” option.
MS Excel 95, 97	READ ONLY	Only “load variables” operation is available.
dBase III, IV, 5.0	OK	
Paradox 3.x, 4.x, 5.x	OK	“Primary key” must be defined in Paradox tables. There is a known bug in Microsoft Paradox ODBC driver: while saving a date type variable to a DATETIME type field, value sensibility is corrupted a little (i.e. storing 20:20:20 you can get 20:20:19 in your database). In that case text
FoxPro 2.6	OK	
FoxPro 3.0+	NOT WORKS	
Text Files	READ ONLY	Plain ASCII text files in Comma Separated format are supported. It is necessary to include field names on first row.

In general, access to a database involves the following steps:

1. Generate an SQL query to a database and get entries that match this query (records set).
2. Scan the records as defined in the *Goto...* fields.
3. Get and change values in a database.
4. Get a records number.

Direct dBase interface is the most simple, oriented to applications with strongly restricted database usage. *Dbase III* files can be created and edited with Smartphone Server’s built-in Database Designer and Database Editor.

It is not recommended to use *Dbase* files on a network (due to a lack of access control and collision avoidance).

Microsoft Access is recommended database standard. Smartphone Server supports both Microsoft Access v2.0 and v7.0 database standards. Use Microsoft Access to edit these databases.

When creating a blank Smartphone Server database in Microsoft Access, clear the default values.

Only the following field types cannot be used in any operations (SQL Query, etc.):

- Replication ID
- Replication ID Auto
- OLE Object

All other field types can be used in expressions, including:

- True/False
- Yes/No
- On/Off
- byte variable types

The ‘Memo’ field type cannot be used in:

- conditional expressions in a Query
- a “WHERE” expression within an SQL dialog.

The “Memo” field cannot contain more characters than the maximum for Smartphone Server string variables.

Use Smartphone Server’s *ODBC32* package for Windows NT/2000/XP to create ODBC sources that can be accessed directly by **Database Boxes**. ODBC sources can be managed in the Control Panel. Consult ODBC Help

There are two methods of making database requests.

The first one, *‘Query’*, makes the process of building a query fast and simple. The second one, ‘SQL’, is oriented for advanced users who are familiar to SQL specification. Smartphone Server uses the following SQL standard: the grammar for the Integrity Enhancement Facility (IEF) is taken directly from the X/Open and SQL Access Group SQL CAE specification (1992) and is a subset of the emerging ISO SQL-92 standard.

Using SQL directly you will be able to generate any query to your ODBC sources (restrictions are mentioned above) by forming a standard "SELECT**"** command.

Note: *According SQL standard you should always use a wide format of date, like '2000-12-31 18:30:00' without reducing of any part. Therefore FDATETIME is the only system variable to be used in condition expressions. Try also to make only full date types of fields in MS Access. If you need to have date only or time only, make proper fields in a database and use only SQL directly (not Query). For more information please consult SQL specification.*

Database sharing

One box is defined as the "master". Other boxes may be connected to the master using the 'Share...' option. If the option 'Each pass' is enabled for the master box, then the set of records for all shared boxes is updated when the master box is executed. For example:

The "master" generates a query and receives a records set.

A **Database Box** (that shares the query with the "master") may read a record (or many records with the Repeat box) from the records set formed by the "master". Another **Database Box** (that shares the records set with the "master") may save these records to a database.

If a shared box is executed before the master box has executed, the shared box error exit is taken.

VAP Editor database tools

Go to the menu "Tool" (VAP Editor) and choose the command "Database Designer". Choose a name for the database (i.e. vocmail.dbf). Confirm by clicking "OK".

A new window will appear called "Database Designer".

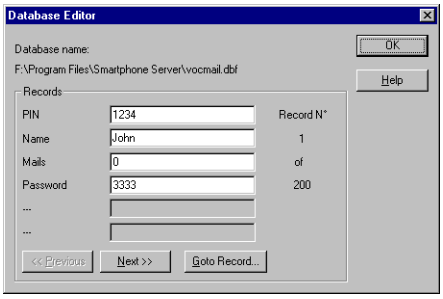
Database Designer

Now you begin with the labelling of the fields (at most 10 letter) as well as their size and enter the total number of the desired data records. Please note that in case of a Voicemailbox database, the number of the data records should agree with the number of the mailboxes. As in the example above, this means that altogether 200 mailboxes with 4 fields each will be created. The sixth field remains empty in this example.

Click **OK** to confirm the information for the respective mailboxes.

Database Editor

Then comes the following window called:



Go through each data record and do the same for each. Click "Goto" to get to a specific data record. The individ-

ual fields (like passwords) can also be changed at a later date. If you would like to make this database compatible to another dBase program, end with “OK”. Start up your database program and open the newly created database.

Note: *Should your database be defined in more than six fields, we recommend opening the dBase database to work in with a dBase compatible software. Complications can occur when importing to MS Excel and MS Word.*

Reference Guide

Chapter 9: VAP Editor Overview

This part describes the use of Smartphone Server from the user's perspective and includes the following information:

- The program development environment (the programming desk)
- The sound recorder
- The Voice Application (VAP) configurations

These points will be illustrated and followed by practical examples and instructions.

Menubar

The Menu Bar is located at the top of the Smartphone Server Screen. It displays the names of pop-up menus:

File: basic operations with Voice Applications and Smartphone Server program;

Edit: functions to help you while building a Voice Application;

Tools: starts additional tools included in Smartphone Server program;

Settings:helps to configure your VAP, Smartphone Server station or license;

Voicemail/Unified Messaging:helps to administer the Smartphone Server system;

Window:helps to arrange Smartphone Server windows on your desktop;

Help: opens the Smartphone Server online help.

Click on a name (or press the corresponding hot key) to bring up the corresponding pop-up menu.

File menu

Select from the File menu to create, open, save or print a VAP or to set up the phone lines for a VAP.

New: select or click on the New icon in the toolbar to open a new VAP Editor window for a new VAP.

Open: select or click on the Open icon in the toolbar to open a VAP. For each opened VAP a new VAP Editor Window is opened.

Save: select or click on the Save icon in the toolbar to save the opened VAP.

Save as...:select or click on the Save icon in the toolbar to save the opened VAP in a new file.

Page setup...:set up printer options.

Print...:select or click on the Print icon in the toolbar to print the opened VAP.

Setup Lines:select or click on the Setup Lines icon in the toolbar to open the Setup Lines dialog.

Exit: exit from Smartphone Server.

Edit menu

Select Edit to edit a VAP file: to undo a previous command, to create , edit, delete, select, cut, copy or paste a box.

Select a single box by clicking on the box. Select multiple boxes by holding the SHIFT key down and clicking on the boxes.

Undo: undo the previous changes you made to a VAP file.

New Box:select or click on the New Box icon in the toolbar or press the F2 key to open the Box-Selector window and select a new box (a new box can also be created by dragging a box from the Box Bar into the design area of the VAP Editor).

Edit Box:select or double-click on the box in the VAP Editor window to edit a box.

Delete Box:select or press the DEL key to delete the selected box(es).

Select All:select all boxes in the VAP.

Cut: select or click on the Cut icon in the toolbar, or press CTRL-X to cut all selected boxes to the clipboard.

Copy: select or click on the Copy icon in the toolbar, or press CTRL-C to copy all selected boxes to the clipboard.

Paste: select or click on the Paste icon in the toolbar, or press CTRL-V to paste all boxes in the clipboard to the VAP Editor.

Tools menu

Recorder:select or click on the Recorder icon in the toolbar to open the Voice File Recorder window.

Mailbox Administrator:open Mailbox Administrator.

Database Designer:open the Save As… window. Enter a file name (for the dBase III format (*.dbf) file) and then the Database Designer window appears.

Database Editor:open the Open window. Enter a file name (for the dBase III format (*.dbf) file) and then the Database Editor window appears.

Online Statistics:display call statistics for all current calls.

Report Generator:start the Report Generator.

Auto Arrange:automatically arrange all boxes of the current VAP in the design area. The boxes will be arranged as a tree with any Start Boxes at the top of the tree. Note: If the VAP does not include a Start Box, the boxes will not be arranged.

Demo Mode:run Smartphone Server in demo mode. Demo mode supports communication via a single phone line or message windows. If the License codes are not correct, Demo Mode will be selected and cannot be changed. If the hardlock is not present in the LPT port longer than 2 hours after start, Smartphone Server will also switch to Demo Mode.

Settings menu

Select Settings to edit VAP, telephone, and statistics settings, edit the “black list”, grid settings and license information.

TTS Settings: open the TTS Settings dialog and change the settings of TTS.

Voice Application Settings:open the VAP Settings dialog and change settings for the open VAP.

Edit Black List...:open the ‘Black list’ window and edit the list of disabled telephone numbers.

Define variables...:open the VAP variables window and edit the list of *Shared or Global variables.*

Telephone Lines...:open the Telephone Settings window and edit settings for all VAPs.

Statistic Settings...:open the Statistics Settings window.

Show Grid:display (default) or hide the grid in the design area of VAP Editor window.

Align to Grid:align or not align boxes to the grid in the design area of the VAP Editor window.

Box Palette:choose the color pallet for displaying the box icons.

License...:open the license window and edit license codes.

Window menu

Select Window menu to rearrange the displayed Smartphone Server windows.

Cascade:rearrange the displayed VAP windows;

Tile: rearrange the displayed VAP windows in another style;

Arrange Icons: rearrange minimized windows;

Close All: close all VAPs and minimize all system windows;

Log Window:switch to the Log window;

Setup Lines dialog:switch to the Setup Lines dialog;

VAP Windows: switch to the one of VAP windows.

Help menu

Select Help to view online documentation for Smartphone Server product.

Contents:view the table of contents of all online documentation for all Smartphone Server applications.

Overview:view the overview topic of online help for Smartphone Server main program.

Glossary:view the Smartphone Server glossary that contains many telephony, computer, speech recognition, CT terms and their descriptions.

About…:view the short description of the current version of Smartphone Server and get some general information about your system.

Context pop-up menus

Most Smartphone Server dialog fields are linked with appropriate pop-up menus that are accessible by clicking the right mouse button on that field. Pop-up menus are

made to simplify many operations, like copying and moving data from one field to another.

Toolbar

The Tool Bar is located near the top of the Smartphone Server Window below the Menu bar.

You can move it anywhere you need to make working in Smartphone Server convenient for you.

The icons in the Tool Bar represent the following commonly used commands:

New

A new VAP Editor window is opened by selecting “New” from the File Menu or by clicking on the symbol **New** in the tool bar.

Open

Select “Open” in the File Menu or click the **Open** button in the toolbar in order to reach the open-file window, where a VAP can be selected. Each loaded VAP opens in a new window.

Save

To save a VAP that is currently being edited, select “Save” from the Menu File or click on the **Save** button. To save a newly created file, enter in the file name under “Save as”, which automatically appears.

Print…

Because of the often limited screen size of computers, it is often helpful to print out a program to get an overview. Simply press the “Print” button on the toolbar or select “Print” from the Menu File.

Cut

The selected box or group of boxes can be deleted by clicking on the “Cut” button in the toolbar. The deleted box (or boxes) is automatically copied onto the clipboard with its connecting lines, and can be recalled by pressing the “Paste” button in the tool bar.

Copy

The “Duplicate box” makes an exact copy of a box, or group of boxes (and the parameters) onto the clipboard, where they can be readily recalled.

Paste

To insert a box, or group of boxes, from the clipboard onto the programming desk, merely place the cursor at the desired place and press the **Insert** button on the toolbar or the Edit menu.

New Box

This is not to be confused with “New”, which opens up new VAP Editor windows. The command “New Box” allows one of thirteen available boxes to be chosen from a window called Box Selector. The selected box is then dragged and dropped from the box bar to the programming desk.

Recorder

To get to the Recorder window, press **Recorder**. Messages or prompts can be entered or amended.

Smartphone Server

Press Smartphone Server button to start Smartphone Server.

Status Bar

The Status bar is located at the bottom of the Smartphone Server main window. It is a one-string gray text line. Place the mouse pointer over any Menu bar command to display a short description about this command in the Status Bar.

VAP Editor windows

VAP Editor is a visual programming desk for creation and editting Smartphone Server Voice Applications (VAPs). Each VAP Editor window (see a picture above) allows to edit one VAP. There may be none, one or many editting windows opened.

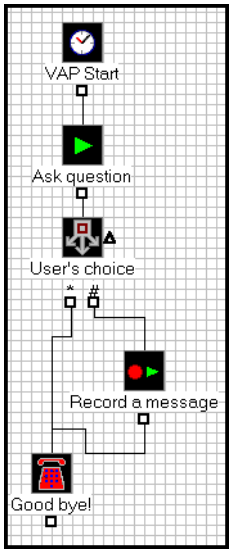
To create a new (blank) VAP Editor window, choose *File / New.*

To open a VAP for editing in VAP Editor, choose *File / Open.*

To save a VAP you are currently editing (its window must be active), choose *File / Save.*

To close a VAP Editor window, click on  icon at the top of the window.

When you open or create a new Voice Application in **VAP Editor**, a new window appears:



That window consists of:

- **window title** with a filename of the Voice Application;
- **boxbar** with the icons of all box types available;
- **design area**, where structure of the VAP (boxes linked with connecting lines) is located.

To know more about Voice Applications consult “Writing Voice Applications”.

Basic Functions Overview

Creating a VAP (design time)

Choose menu item *File / New to create a new blank VAP Editor window.*

When all necessary boxes are inserted into the body of the VAP, and all Voice Application Settings are configured, save your VAP to the file on a hard disk. To do this, choose menu item File / Save…. When you save a new VAP for the first time, or you choose a File / Save as… menu item, a dialog titled ‘Save as’ appears. Choose a filename for your Voice Application in this dialog and then click on Save.

Editing a VAP (design time)

To build a Voice Application:

- add boxes: the boxes can be moved by means of the Drag & Drop technique on the program design area from the boxbar;
- edit box settings: after adding a box to the VAP Editor window, you must edit the box settings by double-clicking on the box icon
- make connecting lines: you can create lines by clicking (with the mouse) on the exit of a box and dragging the line to the next box (still keeping the mouse pressed). The lines can be deleted by clicking the exit box where the connection starts;
- edit VAP settings: change custom VAP settings

Each VAP must contain at least one Start box.

Using clipboard

It’s easy to change VAP structure after all boxes are already in the program design area. You can use clipboard to copy, cut, paste, delete one or more boxes with connecting lines. A group of boxes can be moved at the same time by pressing the *Shift key. This allows connected groups of boxes and their connecting lines to be moved within the program design area.*

To finalize VAP design, you can run Auto-Arrange tool.

Other Dialogs Overview

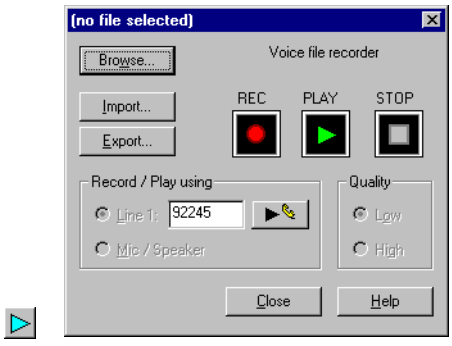
Voice file Recorder

Smartphone Server has a recorder in order to record and play back prompts. Recording and play back concepts are explained in more detail in this section.

Note: *Recordings can be made with any phone (by calling from the phone to the line 1 or dialing the phone number from Recorder), or with a line simulator (Promptmaster, AS-4) connected to line 1 of your voice board. Some voice boards provide a connection for a local phone or an audio device. You can also use the built in *.wav to *.vox converter or the service of a specialized telephony*

recording studio. If a play back is desired, make sure line 1 is not active.

The following picture shows the recorder and its function buttons:



How to start the recorder:

Choose [Tools | Recorder].

or

Click on the Recorder button on the toolbar.

How to dial the phone number

1 Start the recorder.

2 Enter the phone number in the “Line 1” field

3 Click the dial button to call to the specified number.

4 When the phone rings, pick up the phone.

How to record a new prompt on line 1:

5 Start the recorder.

6 Make sure “Line 1” is marked.

7 Click on Record

8 The following message will appear: ”Please pick up the receiver and press OK. The recording begins after the beep tone”

9 Click OK, and begin recording after the beep.

10 Click OK when you are ready to stop recording.

11 Enter the file name under which the prompt is to be saved.

How to play a prompt on line 1:

1 Start the recorder.

2 Choose the prompt which you want to listen to (click on the button “File” and choose the desired file (*.vox or *.wav).

3 Click Play.

4 It is not necessary to close the recorder everytime a file is created. Just click on Select File to open a new file.

How to convert a file from Wave-(.wav) into Smartphone Server format and vise versa.:

1 Start the recorder.

2 Click on Import or Export depending on whether you want to convert a *.wav-file into a *.vox-file or vice versa (Import: *.wav Ø*.vox; Export: *.vox Ø *.wav)

3 Choose the *.wav (resp. *.vox)-file to be converted

4 Enter the name under which the converted file should be stored.

5 Click on OK. The file is converted.

Changing System Prompts

System prompts are, as explained earlier, message prompts ‘spoken’ by the system itself. The system prompts are included in the purchase of Smartphone Server. The user of Smartphone Server can change these prompts (into another language/dialect, translate etc.) as he wishes. System prompts can be re-recorded or new prompts can be recorded.

How to change a system prompt:

1 Start the recorder.

2 Click on Browse.

3 Open the system prompt directory (see appendix).

4 Choose the file and system prompt to be changed.

5 Record the file.

6 Store the file.

Save Requirements

A minute of recording of this quality takes up about 200 KB of hard disk memory.

The meaning of the system prompts, which language is found in which directory, and which box is used with which prompt is all explained in the Appendix.

Replacing existing system prompts with one's own is not recommended if the user wants to add a language that is not currently supported. In this case, it is better to create new prompts spoken in the desired language. A list of the system prompts is found in the Appendix.

Other Settings Dialogs

Voice Application Settings

Select [Settings | Voice Application Settings] to change the settings of the current working VAP.

Settings

- Recording Select the type of recording:

- "High Quality" (WAV format). A WAV file has higher fidelity than a VOX file; however, WAV format is not supported by all Dialogic boards. A WAV file can be converted to a VOX file (compatible with Dialogic boards); however, the conversion process significantly distorts the original recording.
- "Low Quality" (VOX format). A VOX file has lower fidelity than a WAV file, but is supported by all Dialogic boards.

-Dial Out Select Tone Dialing or Pulse Dialing.

- Prompt language Select the language for all System Prompts. A different language may be selected during VAP execution by changing the system variable *language* in Assignment Box.

-Maximum application length Enter the maximum time (in seconds) for an incoming call (calls are normally ended by the Hangup Box).

-Accept Pulse Dialing input Select this option if Smartphone Server should process pulse dialing signals (in the Input Box and the Branch Box). You need Dialogic DPD (Dial Pulse Detection) Enablement Kit or hardware support of DPD installed.

Black List Dialog

Select [Settings | Edit Black List] to disable (prevent outward dialing of) the specified numbers or groups of numbers. Smartphone Server responds to a request to dial a disabled number as if the call is made but not answered. Each VAP has its own *Black List.*

Fields

- Disabled Number The list of all disabled numbers or groups of numbers.

- Comments User comments (for user information only).

Buttons

New Click on New to open the Edit Record window and add a new number or group of numbers (with a user comment) to the *Black List.*

Edit Click on Edit *to open the Edit Record window and edit a number or group of numbers (and/or the user comment) in the Black List.*

Delete Click on Delete *to delete the highlighted number from the Black List.*

Close Click on Close *to close the Black List Dialog.*

Any phone number starting with a sequence of numbers that matches any entry in the Black List will be disabled. For example, when calling from Switzerland, all calls to a foreign country begin with “00”. If “00” is specified in the Black List, no calls can be made from Switzerland to any foreign country.

Edit Record

Here you can define the settings of the disabled phone number.

1. Enter the number in the “Disabled number” field. To disable a group of similar numbers (e.g. all international numbers), enter the prefix of those numbers. For instance, to disable calling all numbers beginning with “113” enter “113” in the *Disabled number field.*
2. Enter any text corresponding the current record of the Black List in the “Comment” field. For example, the reason why the number was disabled.
3. Press OK when finished.

Design Area Settings

By default, the Design Area background is a grey grid and all boxes are aligned to this grid.

Select the circled items to change these settings.

Database Designer

Select menu [Tools|Database Designer...] to open the "Save as..." window. Enter a filename to open the "Database Designer..." window (the fields of which are described below) and to create a small DBase III database. After entering the fields below, clicking **OK** will open the "Database Editor" window (to edit the newly created database).

Settings

1. Database Name. The path/name of the new database (cannot be editted).

2. Fields. The database can contain a maximum of 6 fields. Fields cannot be deleted or added after the database has been initialised.

- Field Name The name of the database field (10 characters maximum, first character is a letter).

- Field Size The size of the database field in bytes (999 maximum).

3. Max. Number of records The maximum number of records in the database (once set, this field *cannot be changed).*

Database Editor

Select menu [Tools|Database Editor...] to open the "Open" window. Enter the filename of or double-click on the database (DBaseIII only) to edit. The "Database Editor" window (the fields of which are described below) is opened for the selected DBaseIII database. You could create such a database in Database Designer tool.

Settings

1. Database Name. The path/name (cannot be editted) of the database to be editted.

2. Records The *Records window displays the fields (6 maximum) of the current selected record of the database.*

- Record Nr. The number of the displayed record (cannot be changed).

- Of The total number of records in the database (cannot be changed).

- <<Previous / Next>> Click on *<<Previous / Next>> to display the previous/next record.*

- Goto Record... Click on Goto Record... to open the Goto... window. Enter the number of a record to display that record.

***Note:** Field names, the number of fields and the maximum number of records cannot be changed within Database Editor. Select Database Designer to create a new database.*

Change Password

When saving a VAP, select the Open as read-only checkbox to protect it with a password. The Change Password dialog will appear. Enter and confirm a new password for write-protection of your VAP; the password can be from 1 to 9 characters long. Your VAP can now only be modified if the set password is entered upon opening or saving of the VAP. To protect the VAP from being viewed at all, select the Do not show application checkbox in the Change Password dialog. Now, without the defined password, the VAP cannot even be viewed; it is now executable only.

To change the set password for your VAP, when saving it check the Open as read-only checkbox. The Change Password dialog appears. After entering the old password, enter a new password and confirm it. The password protecting the VAP has now been changed.

If you have set a password for a VAP and no longer want it to be write-protected you may disable the password. When saving the file, *do not* select the Open as read-only checkbox. The Change Password diaolog will appear. Enter the old password for the VAP and click OK. Pass-

word protection has now been disabled for this VAP and it may be opened and modified by anyone.

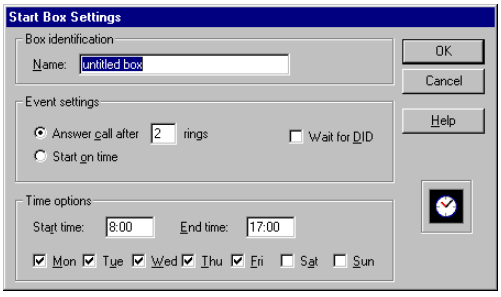
Password

A password can be set for write-protection of your VAP. If a password is set for a VAP, nobody can modify it without that password. Upon opening a password-protected VAP, the ***Password*** dialog will appear. Enter the set password for the particular VAP in order to open it for modification. If allowed, the VAP can be opened in read-only format without entering the password and by pressing the **Read Only** button. The VAP can then be viewed, but no changes can be made to it. If the **Read Only** button is dimmed, the VAP cannot even be viewed without entering the password. A VAP password can be set when the VAP is saved.

See also change password.

Chapter 10: Boxes Settings

Start Box



The **Start Box** performs two basic functions:

1. Detect and answer incoming calls on the assigned line only during the specified time period (*Start Time* to *End Time*).
2. Start processing at a specified time independent of any incoming calls.

Each VAP must contain at least one **Start Box** (in order to be called). A single VAP can contain 2 or more **Start Boxes**, each configured to detect calls or start processing at different times.

Settings

1. Box identification.

The name of the Start Box.

2. Event Settings.

- Answer Call after ... Rings Every call will be answered after the specified number (required) of rings. Note: If 0 rings are specified, no call will be answered.

- Start on Time The VAP starts processing independently of any incoming calls at the time specified under *Time Options.*

- Wait for DID The VAP waits for DID sequences from the PABX and writes these sequences to system variables. DID tone parameters are defined in the "smphone.ini file (in the WINDOWS folder) in the section [DIDSettings]. Consult Smartphone Server User Guide for full list of these settings.

3. Time Options.

-Start Time / End Time The VAP accepts calls only during the specified period of time (from *Start Time to End Time) or starts executing (if Start on Time selected) at the specified time (Start Time) on the selected days.*

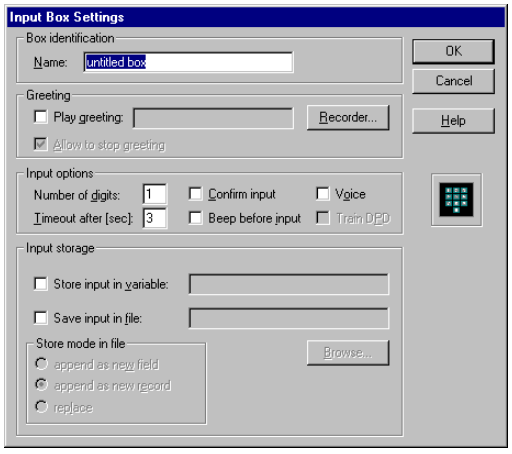
If Start Time is later (greater) than End Time, the VAP answers calls from Start Time of the selected day until 24:00 (midnight) (if the next day is not selected) or until End Time of the next day (if the next day is selected).

If no Start Time or End Time is specified, the VAP answers calls for the entire selected days (from 00:00 to 24:00) (if Answer Calls After is selected) or the VAP restarts after every ended call (if Start on Time is selected).

-Weekday checkboxes The VAP accepts calls or starts executing automatically on the selected days.

Note: *No box can precede a Start Box. A VAP can only start with a Start Box or Hangup Box (with option Global Hangup enabled).*

Input Box



The **Input Box** inputs voice, pulse (defined in the menu item VAP Settings), or tone (DTMF) input from the telephone line and stores the data in a variable or file. Other boxes which can accept input cannot store it and must therefore respond to the input immediately.

If stored in a file, the data can be accessed by applications such as Access, Excel, MultiEdit, Star Office etc.

Pulse inputs can be processed by software or hardware (refer to the Dialogic documentation). A DPD (Dial Pulse Detection) Enablement Kit must be installed to process pulse inputs.

Settings

1. Box Identification.

The name of the Input Box.

2. Greetings.

- **Play Greeting** The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- **Allow to stop Greeting** The user can stop the greeting by pressing a telephone key.

3. Input Options.

-**Number of Digits** The number of input digits expected (required). Maximum 25 digits can be accepted by Input Box.

-**Timeout after [sec.]** The length of time with no input before the box takes the right exit. Any entered digits are written to the variable (file).

-**Confirm Input** After the required Number of Digits has been entered, the caller hears the message “Your input is...” and then the digits the caller entered. The caller then has 2 choices:

1 Confirm input.

2 Make input again.

-**Beep before Input** Generate a beep before the caller input is accepted.

- **Voice** If the language option is active, all digital data can be entered by voice.

- **Train DPD** Check this option to enable Smartphone Server Dial Pulse Detection (DPD) training.

This option may be enabled only if the *Accept Pulse Dialing Input* option in Voice Application Settings is checked.

DPD training enables Smartphone Server to better recognize dialing pulses from a pulse-dialing telephone. Signal characteristics for pulse telephones can vary widely; therefore, if Smartphone Server is allowed to “listen” to a dialed number, it can automatically determine the particularities of the given pulse-dial telephone.

Setup: Insert an **Input Box** in the beginning of the VAP. Check the DPD training checkbox. The box greeting should instruct the caller to dial 0 on his rotary dial phone for DPD training. After the greeting is played, Smartphone Server will wait for the caller to dial 0. After the pulse signals have been detected, Smartphone Server will wait for the other digits.

4. Input Storage.

-**Store Input in Variable** Write the entered data to the specified string variable.

-**Save Input in File** The input data is stored in the specified path/text file. The path/file name can be specified as a string variable. Select Set File to browse for the required file. For example, if the path to (folder of) your VAP is ‘C:\Program Files\Smartphone Server\VAP’,

Entering the file name “User\Choice.txt” will store the Choice.txt file in “C:\Program Files\Smartphone Server\VAP\User”,

Entering the filename “C:\Base\Choice.txt” will store Choice.txt in “C:\Base”.

This field needs a Smartphone Server Expression of string type.

-Store Mode in File Define how the input is written to the file:

-append as new Field The data are separated from the previous data in the file with a TAB symbol (ASCII 9).

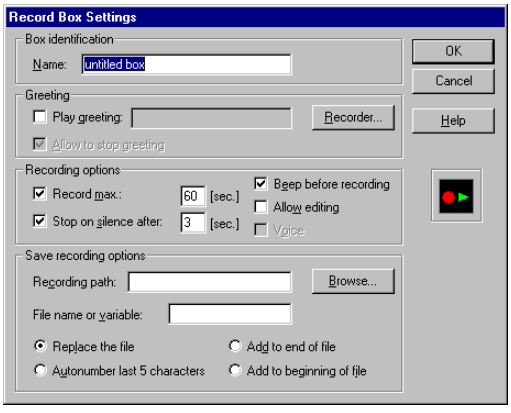
-append as new Record The data are separated from the previous data in the file with a Carriage Return symbol (ASCII 13).

-replace The previous data in the file are overwritten.

Note: *The advantage of storing input in text files becomes evident in applications like a mail order system. If customer number, article number and quantity ordered are input by the caller and stored in a text file, this file can be directly read onto spreadsheets or into database systems like Access, Excel or Filemaker.*

Note: *The Textout Box provides additional features for writing to text files*

.Record Box



The **Record Box** records messages via a telephone line. All incoming messages can be recorded to a single voice ((Wave file) file (with each incoming file overwriting the previous file) or each incoming message can be recorded to a separate file. The method used is specified in the "Save Recording Options" field (below).

The default recording quality (high or low) is set in the VAP Settings menu.

Settings

1. Box identification.

The name of the Record Box.

2. Greetings.

- Play Greeting The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- Allow to stop Greeting The user can stop the greeting by pressing a telephone key.

3. Recording Options.

- Maximum length Maximum length of all recorded messages in seconds.

- Beep before Recording Play a short beep before recording starts.

- Stop on Silence after [sec] Stop recording after a period of uninterrupted silence of the duration specified in the adjoining field.

- Allow Editing Allows the caller to edit the recorded message. The caller hears the following menu after the recording stops:

1 confirm recording

2 replay recording

3 redo recording

If the call is terminated during recording the processing of the Record Box is terminated. Execution of the VAP is resumed only if the Record Box is followed by a Hangup Box.

- Voice Allow the caller to edit the recorded message (as above) with voice commands (without pressing a telephone key) using the Speech Recognition feature.

4. Save Recording Options.

- Recording Path The path to (folder of) the recorded message file (relative to the folder of the current VAP).

Click on Browse to select a different folder. For example, if the path to (folder of) your VAP is 'C:\Program Files\Smartphone Server\VAP':

Entering the path "Messages\" specifies all VOX files in 'C:\Program Files\Smartphone Server\VAP\Messages'.

Entering the path "C:\Base\" specifies all VOX files in 'C:\Base'.

This field needs a Smartphone Server Expression of string type.

- **Browse** Click on this button to select a file from a directory.

- **File Name or Variable** The file name or variable (that contains a file name) for the recorded message. If the variable includes a path, the path set in *Recording Path* is ignored.

Note: The variable must contain a valid file name when the **Record Box** is processed.

This field needs a Smartphone Server_Expression of string type.

- **Autonumber last 5 Characters** If this checkbox is checked, Smartphone Server will automatically add a unique 5-digit sequential number to the end of a file name defined in the "File Name or Variable" field.

The file name of the last recorded message is written to the system variable LASTRECORDFILE.

- **Replace the file** The specified file is overwritten by each new recording.

- **Add to end of file** The digitized recorded speech is appended to the end of the specified file (none of the original file is overwritten).

- **Add to beginning of file** The digitized recorded speech is add to the beginning of the specified file (none of the original file is overwritten).

File name of the last recorded prompt is stored in the system variable LASTRECORDFILE.

Play Box

The function of a **Play Box** is similar to that of a play button on a common tape player. However, rather than playing cassette tapes, the **Play Box** plays pre-recorded waveform files.

Settings

1. Box Identification

The name of the Play Box.

2. Greeting

- **Play Greeting** The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of type string.

- **Allow to stop Greeting** The user can stop the greeting by pressing a telephone key.

3. Playback Options

- **Voice files** Switches to playback of the waveform files from the selected folder.

- **Messages** Switches to playback of the Microsoft Exchange messages. Plays all voice and fax messages and new text messages from the selected folder.

For every message Smartphone Server plays the following:

- subject field (using text-to-speech technique);
- (if any) message body (using text-to-speech technique);
- (if any) attached files - voice or text.

- Play all matching Files Play all files in the folder (whose path is specified in Playback Path) that match the file name (as specified in File Name or Variable) specified in Playback Files. Files are played according to the file creation date (more recent files are played first).

- Time Stamp State the date and time of each played file as it is played.

The format of the stated date depends on the current date (see Speak function). For example, if today is 24 October 1997 (Friday), various file dates will be announced as:

File dateAnnouncement

24. October.97Today

23. October.97Yesterday

22. October.97Wednesday

21. October.97Tuesday

20. October.97Monday

19. October.97Sunday

18. October.97Saturday

17. October.97

17 October (the year is not stated)

The full date (without the year) is announced only if the date is at least 7 days prior to the current day.

- Allow Editing Allow the caller to edit the list of played message files. The caller hears the following menu after each prompt is played:

1 Replay Prompt

2 Skip

3 Delete Prompt

4 Exit this menu

The caller presses the corresponding key on the telephone.

- Voice Allow the user to edit the list of played messages (as above) with voice commands (without pressing a telephone key) using the Speech Recognition feature.

4. Playback Items

- Playback Path

If the "Voice files" option was selected (see above):

The path to (folder of) voice files (relative to the folder of the current VAP). Click on **Browse** to select a different folder. For example, if the path to (folder of) your VAP is 'C:\Program Files\Smartphone Server\VAP':

Entering the path "Messages" specifies all VOX files in 'C:\Program Files\Smartphone Server\VAP\Messages'.

Entering the path "C:\Base\" specifies all VOX files in 'C:\Base'.

If the "Messages" option was selected:

The folder which contains the items (messages) to play. Click on **Browse** to select a different folder.

Note: *You may select any folder in any message store enabled in the default profile on the Smartphone Server computer. Make sure that one of the profiles is set as default (consult your mail software documentation on how to do this).*

This field needs a Smartphone Server Expression of string type.

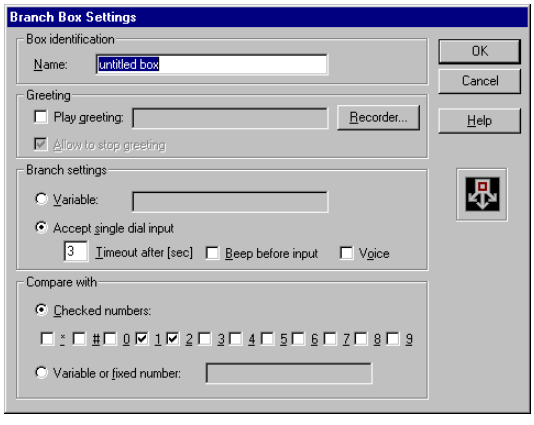
- Browse Click on this button to select a file or a folder.

- File Name or Variable The name of the message or file to play.

This field needs a Smartphone Server Expression of string type.

Note: *If Play all matching files is active, the messages will be played back in reverse order, i.e. the message recorded last will be played back first.*

Branch Box



The **Branch Box** compares one of the following values (as defined in the "Branch Settings" group box):

- The value of an expression (as defined in the field "Variable:" if the corresponding radio button is selected)
- Caller input (a pressed telephone key) (if the radio button "Accept single Dial Input" selected)

with one of the following values (as defined in the "Compare With:" group box)

- A telephone keypad digit (as specified with the checkboxes if the radio button "Checked Numbers:" selected)
- The value of an expression (as defined in the field "Variable or fixed Number" if the corresponding radio button is selected)

The number and type of **Branch Box** exits depends on the selected **Branch Box** options:

If the radio button "Checked numbers:" (in the "Compare With:" group box) is selected: The **Branch Box** has the following exits:

1 exit corresponding to each checked keypad digit. An exit is taken if the caller pressed a keypad digit that matches the digit for this exit.

If the radio button "Variable or fixed Number:" (in the "Compare With:" group box) is selected:

The **Branch Box** has 2 exits

- The ✓ exit. This exit is taken if the value of the variable in the "Variable or fixed number" field matches "the value of the variable or caller input (in the "Branch settings" box).
- The ✖ exit. This exit is taken if the value of the variable in the "Variable or fixed number" field does NOT match "the value of the variable or caller input (in the "Branch settings" box).

If the radio buttons "Accept single Dial Input" or "Checked Numbers" are selected, the following exit also exists:

- Error exit (on the right side of the box). The error exit is taken if:
- No valid key was pressed (if "Accept single Dial Input" was selected)

The key pressed or variable defined in the field "Variable" did not match any numbers selected with the checkboxes belonging to the "Checked numbers" radio button.

Settings

1. Box identification

The name of the Branch Box.

2. Greeting

- Play Greeting The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- Allow to stop Greeting The user can stop the greeting by pressing a telephone key.

3. Branch Settings

- Variable The first value of the comparison is a variable, specified in the adjoining field.

This field needs a Smartphone Server Expression.

- Accept Single Dial Input The first value of the comparison is the first key pressed on the telephone.

-Timeout after [sec.] The length of time with no input before the box takes the right exit.

- Beep before Input A short beep is played before a telephone key pressed by the caller is accepted (selectable only if Accept Single Dial Input is selected).

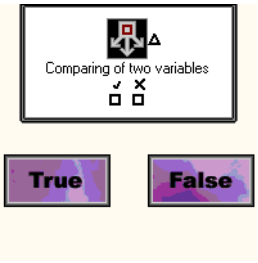
- Voice The first value of the comparison is a spoken variable (using the Speech Recognition feature).

4. Compare With:

- Checked Numbers The Branch Box has 1 exit corresponding to each checked key in the list of telephone keys. The exit that corresponds to the value specified above in "Branch Settings" is the exit taken from the Branch Box. The Branch Box also has an error exit (on the right side of the box).

Comparing a variable with a numeric constant specified in *Checked Numbers* is not recommended.

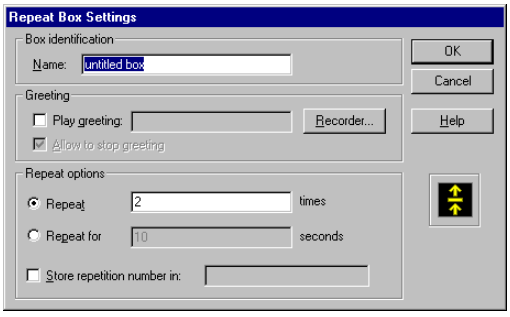
- **Variable or Fixed Number** The Branch Box compares the value specified in "Branch Settings" (see above) with a fixed number or variable specified in the adjoining field. This field needs a Smartphone Server Expression of string type. In this situation the **Branch Box** has two exits:



✓ The values are equal.

✗ The values are not equal.

Repeat Box



The **Repeat Box** makes it possible to repeat a section of a VAP a specified number of times. It has two exits:

- *Repeat* exit (on the right side of the box)The exit chosen when the repeat loop has not yet been executed the specified number of times or specified time is not up.

- *Regular* exitThe exit chosen when the repeat loop has been executed the specified number of times or time in *Repeat for ... seconds* is up. The loop counter is restarted when the regular exit is taken.

Settings

1. Box Identifications.

The name of the Repeat Box.

2. Greetings.

- **Play Greeting** The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a of string type Smartphone Server Expression.

- **Allow to stop Greeting** The user can stop the greeting by pressing a telephone key.

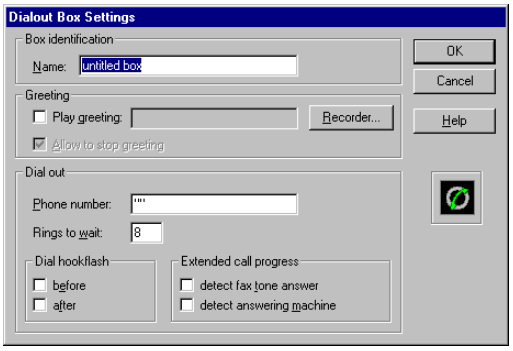
3. Repeat Options

- **Repeat ... Times** The number of times the Repeat Exit is taken. If "Repeat ... Times" is checked, the Repeat Exit can only be connected to the Repeat Box input if Play Greeting has been checked and a valid greeting file has been selected.

- **Repeat for ... seconds** The length of time the loop will repeat. After this time has elapsed the Regular exit is taken.

- **Store Repetition Number in:.** Store the number of executed repetitions in the specified variable. This field needs a Smartphone Server Expression of integer type.

Dialout Box



The **Dialout Box** performs two functions:

1 Dials an actual phone number (this includes going off-hook, waiting for the dial tone, dialing the number, etc.). In this case **Dialout Box** has 3 - 5 possible exits:

- ***Regular (V)* exit** (at the bottom of the box)the call was answered by voice.

- ***Regular (F)* exit** (at the bottom of the box)the call was answered by a fax machine *(detection of a fax machine is optional)*.

- ***Regular (A)*** exit (at the bottom of the box)the call was answered by an answering machine *(detection of an answering machine is optional)*.

- ***N-exit*** (on the left side of the box) the call was not answered (NO ANSWER).

- ***B-exit*** (on the right side of the box)the number was busy (BUSY).

2 Generates DTMF tones on the line. In this case **Dialout Box** has only the *Regular Exit*.

Settings

1. Box identification.

Specifies the name of the box which is displayed under the box icon or in the “Setup Lines” window when this box executes.

2. Greetings.

- Play Greeting Plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a prompt file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- Allow to stop Greeting Enables stopping the greeting by pressing a telephone key.

3. Dial out.

Dialing string Specifies the telephone number (spaces are ignored and non numeric special characters (for example “&” are processed)). Dialing mode (Pulse Dial or Tone Dial) is set in the dialog [Settings | Voice Application Settings].

This field needs a Smartphone Server Expression of string type.

- Rings to wait Specifies the number of rings after which an unanswered call is cancelled. The N-exit (not answered exit) is taken.

3.1. - Dial hookflash

Specifies that a hookflash (generated for the PABX) should be dialed before or after dialling the number (flash hook signals are defined in the Telephone Lines window).

- before Hookflash is generated before dialling the number.

- after Hookflash is generated after dialling the number.

If a dialed number is busy or not answered after *Rings to wait* rings, a hookflash is sent and the N-exit (no answer) or the B-exit (busy) is taken. The flash hooks need only be entered once. Their values are dependent upon the PBX being used, so read the PBX manual carefully. Go to the Settings menu, select “Telephone Lines”, and enter the values here, along with the time and length of the flash hooks. These values are then saved as default values. If there is any doubt as to the correct value, contact the producer or retailer

No PBX has the same settings as any other. Therefore, it is very important that you study the Handbooks that are delivered with the PBX and/or contact your retailer.

- Extended call progress Enables the detection of the type (voice, answering machine, fax) of party answering the call.

- detect FAX Tone Answer Enables the following box exits:

- F (Fax)
- Regular (will be labelled as V (voice))

- detect Answering Machine

- A (answering machine)
- Regular (will be labelled as V (voice))

Box exits taken:

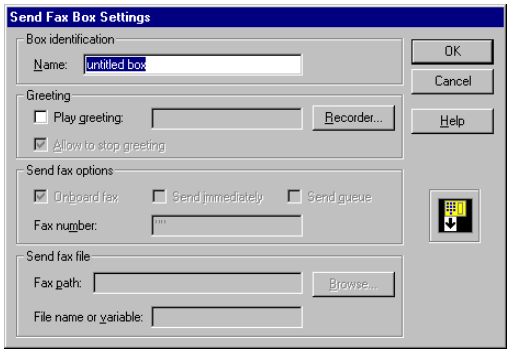
- F exit (if enabled): If the call is answered by a fax machine.
- A exit (if enabled): If the call is answered by an answering machine.
- V exit: If neither the A or the F exits are taken.

For previous VAP versions: A Dialout Box in a VAP written for previous versions of Smartphone Server will be displayed in the new version with “Detect FAX tone answer” and “Detect Answering Machine” disabled.

Setting up an ACD system

1. Enter the phone number (e.g. 1111) of the desired internal extension in the space marked Dial Out / Phone number. Since the flash hook values depend on the PBX being used, it is strongly recommended that they be tested.
2. Enter in the number of rings Smartphone Server should wait for before taking the N-exit.
3. Tick the correct settings of the PBX being used in the hookflash space. (See [Settings | Telephone Settings]) The dial out box uses the (!) to signify a short flash hook. This symbol (!) can be used in Dial Out / Phone number in the Dialout box, which allows combinations of numbers and flash hooks to be sent. The (&) symbol signifies a long flash hook. Remember, however, that your PBX itself may require some setting up and/or fine tuning, too, for your ACD system to function properly.

Send Fax Box



The **Send Fax Box** sends faxes using the built-in fax features of Dialogic voice boards (enabled with the ‘Onboard fax’ option in the License window). Refer to your Dialogic User’s Manual. The D41 boards family uses a daughter-board to support this function. The D41E/ESC board with fax daughter-board FAX/40E is named VFX40/ESC.

Smartphone Server can convert ASCII (*.txt) and TIFF/F graphics (*.tif) files to fax files (TIFF/F is a special type of TIFF file).

Note: *Before sending a TIFF file, the file must be converted to TIFF/F (with a file converter).*

One of the programs that can make such a conversion for you is the Smart-4-Fax utility included with Smartphone Client.

Contact your local software dealer for more information on available utilities that convert any data format to TIFF/F.

The **Send Fax Box** also has an error exit (on the right side of the box). It is taken if:

- The fax document does not exist
- The fax license option is not enabled
- No VAP is operating on a fax line
- An error occurred

Settings

1. Box Identification.

The name of the Send Fax Box.

2. Greetings.

- Play Greeting The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- Allow to stop Greeting The user can stop the greeting by pressing a telephone key.

3. Send Fax Options.

-Onboard Fax Select the method to send a fax document:

Check Onboard Fax to send the fax via the fax board.

-Send immediately

Check if a fax should be sent immediately (during the current telephone call) without breaking the connection. The voice conversation can be continued after the fax has been sent without interrupting the line.

If not selected, the fax will be sent to the fax queue which will be sent during the execution of a later Send Fax Box or at the end of the VAP.

-Send queue If enabled: all fax documents in the queue will be sent immediately (the 'Send Immediately' option is automatically selected when 'Send queue' is enabled). The queue will contain fax documents (up to 100) from previous Send Fax Boxes which had the 'Send Immediately' option disabled.

If not enabled (for all **Send Fax boxes** in the current VAP): the fax queue will be sent to the fax number specified in the "Fax Number:" field (required) only after VAP execution.

***Note:** If a fax number is not dialed successfully, the number cannot be redialed. Therefore, use of the Dialout box and **Send Fax box** with "Send queue" option enabled is recommended.*

-Fax Number The destination fax number or variable (required if the "Send immediately" option is not selected).

4. Send Fax File.

- Fax Path The path to the fax file (relative to the folder of the current VAP). Click on Browse to select a different folder. For example, if the path to (folder of) your VAP is 'C:\Program Files\Smartphone Server\VAP':

Entering the path "Doc\" specifies all fax files in 'C:\Program Files\Smartphone Server\VAP\Doc'.

Entering the path "C:\Fax\" specifies all fax files in 'C:\Fax'.

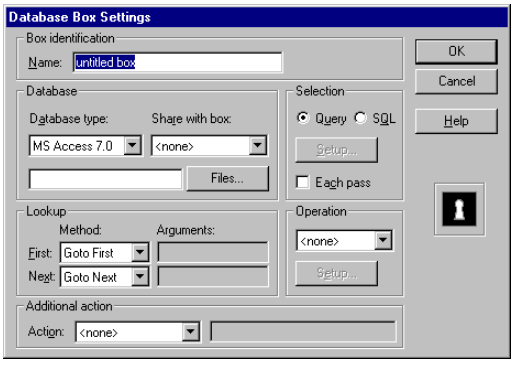
This field requires a Smartphone Server Expression of string type.

-File Name or Variable The file name or variable (that contains a file name) for the fax file. If the variable includes a path, the path set in "Fax Path" is ignored.

This field requires a Smartphone Server Expression of string type.

***Note:** Text files can be converted on-line to fax files.*

Database Box



The **Database Box** provides access to databases.

Each **Database box** has its own settings (for pointers, record sets, etc.). To continue a search in a record started in a previous **Database Box**, use system variable LASTDBRECORD to retrieve previous database pointers.

A **Database Box** cannot create a database.

Settings

1. Box identification

The name of the Database Box.

2. Database

- **Database type** The type of database (Dbase III, MS Access 7.0 (default), or ODBC Data Source).

- **Share with box** Share the records set or query with other boxes.

Various Database boxes with similar construction can be interconnected, thus helping Smartphone Server to run more quickly. Both the Database type and Selections characters are carried over. In this case Smartphone Server does not repeat a query to a data source in next connections.

-Files Select a database/ODBC source file name or enter the ODBC connection string.

The ODBC connection string can be used for a database that requires authorised connection or / and has no registered DSN in your Windows NT/2000/XP ODBS settings and is not an MS Access database file.

3. Selection

Here you define the one of available methods to form your query to a database. This function is not available with the dBASE III files.

- **Query/SQL - Query** Result records will be formed with a database query (MS Access file and ODBC source). Click on **Setup** to open the Database Query Setup window.

- **SQL** Result records will be formed with an SQL SELECT command (MS Access file and ODBC source). Smartphone Server uses a SELECT command of SQL specification. Click on **Setup** to open the SQL Query design window.

- **Each pass** Reexecute the query every time this Database Box is executed. This Database Box and any Database Boxes that are sharing a database with this Database Box will receive a new record set.

4. Lookup

- **Method** The method to move the database pointer to a record set (MS Access and ODBC) or to a field (Dbase). The following methods are provided:

- Go to Rec: jumps to a specified data record
- Go to New: starts a new data record
- Go to first: jumps to the first data record
- Go to last: jumps to the last data record
- Go to next: jumps to the data record ahead
- Go to prev: jumps to the proceeding data record
- Don't move: does not jump to a different record

The method Goto First is used when the Database Box executes for the first time. After the first execution of the Database Box, the method Goto Next is used. The numbering of the records is determined by the query.

- **Arguments** Parameters (arguments) for the corresponding method. An Smartphone Server expression can be used.

This field needs a Smartphone Server Expression of integer type.

5. Operation

Select the operation:

- *<none>*
- Load variables
- Save variables
- Delete record

Load variables: Having selected this command, clicked **Setup**, the dialog appears. Here you can decide which field in which variable should be loaded in Smartphone Server. The fields can be selected from a list box (drop-down menu).

Save variables: To save Smartphone Server variables in a database, use the command "Save Variable" and click on **Setup**. The dialog appears. Fill in the name of the Smartphone Server-variables on the right side; on the left, the name of the database field where the value should be saved. Again, the database fields can be chosen from the drop-down menu.

Delete Record is for Microsoft Access and ODBC only.

Setup opens the corresponding settings window.

6. Additional action

- **Action** Select an additional action:

- <none>
- Number of records
- Reset lookup
- Reset on error
- Close db

- Close on error

Number of records: Select *Number of records* to store the total number of records in the variable specified in the adjoining field.

Reset lookup: The next time this Database Box is executed, the database recordset lookup method specified in the field *Lookup: / Method: / First:* will be used (the actual recordset will not be changed).

Reset on error: If the Database Box *error exit* is taken: The next time this Database Box is executed, the database recordset lookup method specified in the field *Lookup: / Method: / First:* will be used (the actual recordset will not be changed).

Close db: The recordset is updated and the database is closed (the next time this Database Box is executed, the database recordset lookup method specified in the field *Lookup: / Method: / First:* will be used).

Close on error: If the Database Box *error exit* is taken: The recordset is updated and the database is closed (the next time this Database Box is executed, the database recordset lookup method specified in the field *Lookup: / Method: / First:* will be used).

Note: *Smartphone Server controls system variables in which the number of the current data record is saved. This variable is called "lastdbrecord" (which reads: database-record). This variable can be used in Smartphone Server as your own.*

Note: *If you would like to process a database in the dBase-format but the Access option is available, it is recommended that the database first be imported into MS Access and then further processed.*

SQL query design

An **SQL Query design** selects specified types of records in a specified format from a database.

- **SELECT** The command SELECT (number of columns) selects number of columns in the database. Set number of columns to a number to select that number of columns, set to "*" to select all columns, or set to column names to select individual columns. Aggregate functions (AVG, COUNT, MIN, MAX, SUM) and Smartphone Server functions and operators are also supported.

- **FROM** The command FROM (database name) declares the source database name.

- **WHERE** The command WHERE (search conditions) specifies the database search conditions. Multiple search conditions can be specified with the logical operators AND, OR, and NOT. Relation operators and aggregate functions can also specify search conditions.

- **ORDER BY** The command ORDER BY (column name(s)) sorts a query result according to the values in the specified columns. A column name can be an actual column name or the position of the column. The options ASC and DESC are supported.

Database query setup

A **Database Query** retrieves records in a specified order from a database table or query. The **Database Box Settings** "Database Query Setup" window options define the **Database Query** setup.

Query Source

- **Tables** Click on *Tables* to select database tables from the database file as the query source. Select a table from the available tables for the selected database via the selection list on the right.

- **Queries** Click on *Queries* to select database queries from the database file as the query source. Select a query from the available queries for the selected database via the selection list on the right.

Query properties

- **Order by** Select an available database field from the selection list. The records in the query will ordered according to the selected field.

Query design

Define the query operands (*Field and Argument) and the comparison operation performed between the operands.*

- **Field** Select an available field from the database.

- **Operation** Select the type of query comparison operation.

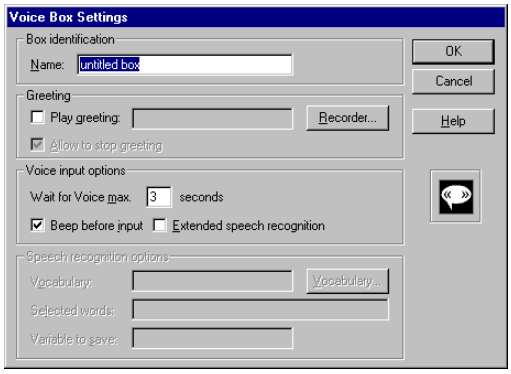
- **Argument** Enter any Smartphone Server -expression.

Note: *Field and argument must be of the same type.*

Database fields which are not Smartphone Server **expressions** and not logical values are converted to strings.

Logical "False" and "True" are converted to integer values 0 and 1 respectively.

Voice Box



The **Voice Box** provides Speech Recognition.

Smartphone Server by default reacts to any speech on the line. However, ***Extended Recognition*** provides Speech Recognition.

The **Voice Box** has two exits:

- *Regular exit*If ***Extended Recognition*** enabled: Taken if voice information was successfully recognized. If ***Extended Recognition*** not enabled: Taken if any speech was detected.

- *Error exit*If ***Extended Recognition*** enabled: Taken if no voice information was successfully recognized. If ***Extended Recognition*** not enabled: Taken if no speech was detected.

The Regular Exit is on the bottom of the box and the Error Exit is on the right side of the box.

Settings

1. Box identification

The name of the Voice Box.

2. Greetings.

- **Play Greeting** The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- **Allow to stop Greeting** The user can stop the greeting by pressing a telephone key.

3. Voice Input Options.

-Wait for Voice max ... seconds The maximum length of time with no voice input that the Voice Box will continue to wait for input.

This option is available only if *Extended Speech Recognition* is disabled (if *Extended Speech Recognition* is enabled: speech recognition time parameters are defined in the “smphone.ini” file.

-Beep before Input A short beep is played before the Voice Box accepts voice input.

-Extended Speech Recognition Enable full Speech Recognition capabilities.

4. Speech Recognition Options

If Extended Speech Recognition is enabled, Smartphone Server attempts to not only recognize the presence of speech on the line, but also recognize what was spoken. A subvocabulary may be chosen to improve speech recognition by limiting the number of words that can be recognized to a group of words consisting of all valid possible answers for this Voice Box.

- **Vocabulary** IThe vocabulary selected in the “Words Selector” dialog. From this vocabulary all words in the “Selected Words” field are selected.

This field requires a Smartphone Server Expression of string type.

-Button “Vocabulary”

If ASR speech recognition installed:

Click on the button “Vocabulary” to open the “Words Selector” dialog. Dialog “Words Selector”

- Vocabulary Name:

In the drop list box “Vocabulary Name” choose 1 of 2 vocabularies:

Cx: Single or multiple digits only (0...9)

Lx: Single digits (0...9) and single words.

“x” is the number of the language (see system variables for more information about available languages).

In the “Vocabulary words:” field is a complete list of all available words for the vocabulary. In the “Selected

words:" field is a list of all selected words from that vocabulary. The Voice Box will try to match any spoken word with only words from the list "Selected words:".

- Vocabulary words:

To add a word or group of words to the "Selected words" list:

1 Click on the needed word(s) (holding down the shift key while selecting multiple words is not necessary).

2 Click on "Add>>".

To add the entire vocabulary to the "Selected words" list:

- Click on "Add All>>".

- Selected words:

Use the "<<Remove" and "<<Remove All" to delete selected words or all words respectively from the "Selected words:" list.

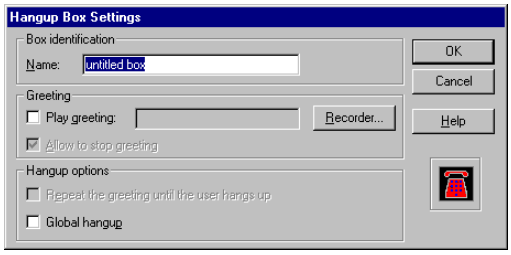
Note: *Custom vocabularies can be created with the Lexicon Toolkit (provided with Smartphone Server Speech Recognition software).*

Note: *Vocabularies of the type "Continuous recognition" are not supported in Smartphone Server.*

-Selected Words The selected words from the vocabulary (selected in Words Selector) that will be considered the only possible valid voice responses (an attempt will be made to match any voice response with only the displayed words).

-Variable to Save Any recognized voice input is written to the string variable in this field.

Hangup Box



The **Hangup Box** ends a call (hangs up the line) or handles an ended call.

Processing of the VAP can continue through the **Hangup Box** exit.

The **Hangup Box** is not required for the VAP to successfully end a call or handle an ended call, but is recommended.

Settings

1. Box identification

The name of the Hangup Box.

2. Greetings.

- Play Greeting The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- Allow to stop Greeting The user can stop the greeting by pressing a telephone key.

3. Hangup Options.

-Repeat greeting until the user hangs up Repeat the greeting until the caller hangs up.

-Global Hangup Processing of the VAP continues from this Hangup Box immediately after the user hangs up (execution from any other box is transferred to this box). Only one Hangup Box can select *Global Hangup.*

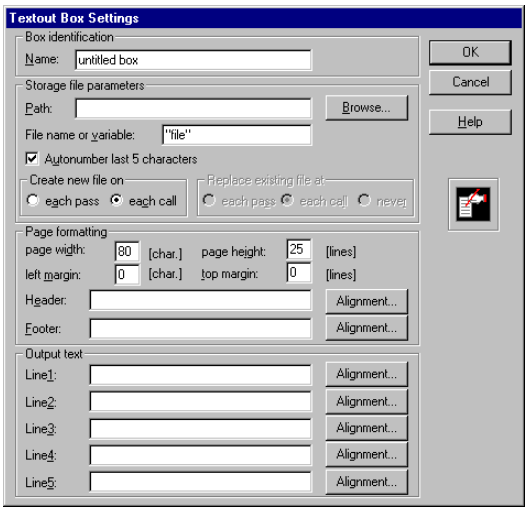
Exception: if hangup occurs during Script Box execution, the operation flow will not be switched to the Global Hangup box. However, you can still track the hangup event by using the "IsHangup" Script Language function.

The OnEvent Script Box is more reliable than Global Hangup feature.

Note: *If Global Hangup is selected and the caller hangs up the application is not halted.*

Note: *The processing of the VAP will automatically be ended if the duration of the current call exceeds the maximum duration of a VAP as specified in VAP Settings.*

Textout Box



The **Textout Box** writes text and Smartphone Server -expressions (text or variable values) to text files.

Settings

1. Box identification

The name of the Textout Box.

2. Storage file Parameters

- Path The path to the textout file (relative to the folder of the current VAP). Click on Browse... to select a different folder. For example, if the path to (folder of) your VAP is 'C:\Program Files\Smartphone Server\VAP*':*

Entering the path "Reports" specifies that all text files will be written to 'C:\Program Files\Smartphone Server\VAP\Reports'.

Entering the path "C:\Archive\" specifies that all text files will be written to "C:\Archive".

This field requires a Smartphone Server Expression of string type.

- Browse Click on this button to select a file from a directory.

-File Name or Variable The file name or variable (that contains a file name) for the text file. If the variable includes a path, the path set in "Path" is ignored.

This field needs a Smartphone Server Expression of string type.

- Autonumber last 5 Characters If this checkbox is checked, Smartphone Server will automatically add a unique 5-digit sequential number to the end of a file name defined in the "File Name or Variable" field.

The file name of the last recorded text file is written to the system variable LastTextOutFile.

If this option is not selected, each newly written text file overwrites the previously written text file.

- Create new file on If "Autonumber last 5 Characters" (see above) is selected, the conditions under which a text file will be written must be specified. There are 2 options:

- Each pass
- Each call

- Replace existing file at If "Autonumber last 5 Characters" (see above) is not selected, the conditions under which a text file will be written (and the previous text file overwritten) must be specified. There are 3 options:

- Each pass
- Each call
- Never

3. Page formatting

These fields define the page format of the written text file:

- page width the number of characters per line.

- page height the number of lines per page.

- left margin the number of spaces between the left margin and the first character of each line.

- top margin the number of blank lines at the top of each page.

- Header the text of the page header.

- Footer the text of the page footer.

These fields need Smartphone Server Expressions of string type.

Click on **Alignment** and then select the number of blank lines after the header or footer ("Number of lines feeds after...") and the type of header or footer alignment:

- Left
- Centre
- Right

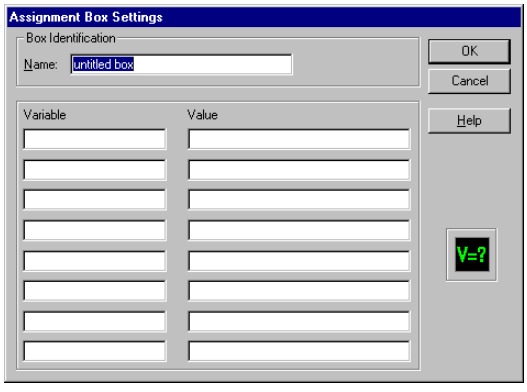
4. Output text

-Line1, Line2, Line3, Line4, Line5 Smartphone Server string expressions to be written to the text file in the corresponding line.

Click on **Alignment** and then select the number of blank lines to be written after the text line and the type of text alignment:

- Left
- Centre
- Right

Assignment Box



The **Assignment Box** assigns values to variables.

The Script Box supports more powerful operations with variables.

Settings

1. Box identification

The name of the Assignment Box.

2. Variables and Values

- Variable The variable name.

- Value Smartphone Server expressions for variable values.

Example:

The following variables are defined in the assignment box as:

String1 as“the value of the strings”

Number 1 asthe number 5

String2 asas “the value of the strings”

Number 2 asNumber 1

String3 as“12 &*()*&“

String4 asString1+“and“+$Number1

Number 3 asNumber1*2+7

The contents of the variable look then as follows:

String1 contains“the value of the strings”

Number1 containsthe value 5

String2 contains“the value of the strings”

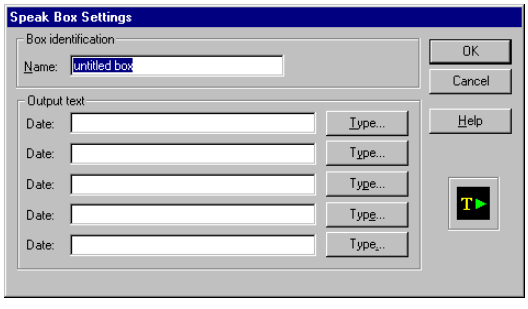
Number2 containsthe value 5

String3 contains“12 &*()*&“

String4 contains”the value of the strings and 5”

Number3 containsthe value 17 (5*2+7)

Speak Box



The **Speak Box** performs some kinds of Smartphone Server speaking, for instance, 'reading' a text file to the caller.

Settings

1. Box Identification

The name of the Speak Box.

2. Output text

- Date / File / Time / SNmb / CNmb / TTS / TTSF There are five strings available to define the source information to speak. Click on the Type button corresponding the current string to change the type of the information to speak. All parameter fields need Smartphone Server Expressions of string type.

File Choose this option to play back a prompt. Then click to the '...' button to browse to the waveform file you need.

It is quite similar to use a Play Box, but you can accumulate some speech tasks in one **Speak Box**.

Date Makes Smartphone Server read a date specified in the field. You can use system variable DATE to get the current date.

Note: *The format of the date corresponds the Regional Settings of Windows NT/2000/XP.*

Time Makes Smartphone Server read a time specified in the field. You can use system variable TIME to get the current time.

Note: *The format of the time corresponds the Regional Settings of Windows NT/2000/XP.*

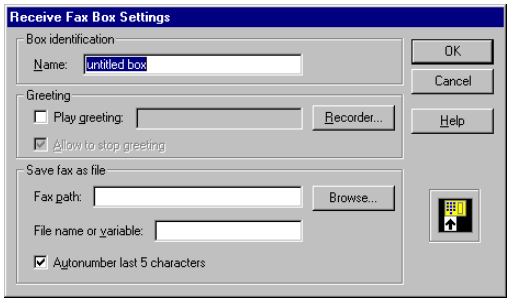
SNmb (Separated numbers) Select this option to make Smartphone Server read the sequence of numbers and special symbols (e.g. '514*0') as separate ones.

CNmb (Composed numbers) This option is similar to the previous one, but in this case Smartphone Server speaks all sequence of numbers as one big number which consists of up to 9-digit number.

TTS (Text To Speech) Select this option if you want Smartphone Server to read a sentence.

TTSF (Text File Reading) Select this option to make Smartphone Server read all sentences in the text file specified in the string. Click to the '**...**' button to browse to the *.TXT file you need to be read.

Receive Fax Box



The **Receive Fax box** receives fax documents within a VAP.

The Receive Fax box has two exits:

- *Regular*Operation successful.

- *Error* An error occurred within the Receive Fax box (the fax was not received).

If the fax turnaround detection feature is enabled (in the "smphone.ini" file), then the specific value will be stored in the LastErrorInt and LastErrorStr variables (if a fax polling request was received):

LastErrorInt=-1

LastErrorStr="Box: rec fax: Line: 1: Error: -1 <Polling request detected.>"

The LastErrorInt variable should be check to detect and handle the turnaround polling request at both exits.

Settings

1. Box Identification

The name of the Receive Fax box.

2. Greetings.

- Play Greeting The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field needs a Smartphone Server Expression of string type.

- **Allow to stop Greeting** The user can stop the greeting by pressing a telephone key.

3. Save Fax as File:

- **Fax Path** The path to the file (relative to the folder of the current VAP) where the fax document received by this Receive Fax box will be stored. Click on Browse to select a different folder. For example, if the path to (folder of) your VAP is 'C:\Program Files\Smartphone Server\VAP':

Entering the path "Doc\" specifies all fax files in "C:\Program Files\Smartphone Server\VAP\Doc".

Entering the path "C:\Fax\" specifies all fax files in "C:\Fax".

This field needs a Smartphone Server Expression of string type.

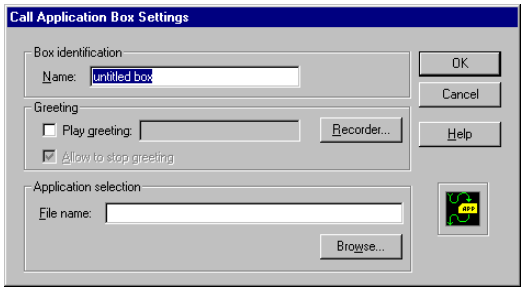
-File Name or Variable The file name or variable (that contains a file name) for the fax file where the received fax document will be stored. If the variable includes a path, the path set in Fax Path is ignored.

This field needs a Smartphone Server Expression of string type.

- **Autonumber last 5 Characters** If this checkbox is checked, Smartphone Server will automatically add a unique 5-digit sequential number to the end of a filename defined in the "File Name or Variable" field.

If this option is not selected, each newly written file overwrites the previously written file.

Call Application Box



The **Call Application Box** can start a VAP or a Windows NT/2000/XP application within the current VAP.

The Call Application Box has two exits:

- *Regular*No errors occurred within the Call Application Box.

- *Error* An error occurred within the Call Application Box (the VAP cannot be found, there is no Start Box in the VAP specified, etc.).

Settings

1. Box Identification

The name of the Call Application Box.

2. Greetings.

- **Play Greeting** The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field requires a Smartphone Server Expression of string type.

- **Allow to stop Greeting** The user can stop the greeting by pressing a telephone key.

3. Application selection

- **File Name** Specifies the path and the file name of the application (Smartphone Voice Application or Windows NT/2000/XP application) to execute. Click on Browse to select an application from a different folder. Specify the full path to the application, or do not specify it at all if the application path is included in PATH environment variable.

This field requires a Smartphone Server expression of string type.

You can run both Smartphone Server VAP's and Windows NT/2000/XP applications (*.exe, *.cmd, *.bat, etc.).

During execution of the specified VAP, execution of the VAP containing this Call Application box is halted.

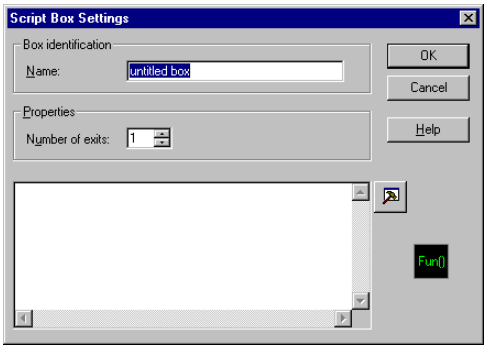
Note: *Any pending notification of received messages (via phone lamp, phone, or email) will be cancelled if there are no VAP's executing on any line.*

Note: *A VAP that is running in a Call Application box stops executing when the Call Application box is exited. Therefore, if all Voicemail/Unified Messaging VAP's are executing from within Call Application boxes, it is possible that at some point no Voicemail/Unified Messaging VAP's are executing on any line.*
To avoid this situation: Run the Voicemail/Unified Messaging VAP on a single line (in the Setup Lines dialog, not from within a Call Application box).

Note: *When you start a VAP in the Call Application box, the launched VAP has ANI, DID and DNIS system variables equal to the parent VAP. No other variables are copied. Tip: You can use shared variables, list functions or inheritance features for data exchange between the VAPs.*

Note: *Please remember that Smartphone Server supports only five levels of nested VAP calls. This means that if one VAP loads and calls a second one, the second one loads and calls a third one and so on, only the first five VAPs in this sequence will be loaded.*

Script Box



The **Script Box** contains the text for a subroutine written with the Script Language. Specialized applications can sometimes be implemented more efficiently and effectively with a Script Language routine. The complete functionality of all boxes can be implemented within a Script Box using the Script Language.

The **Script Box** has 1 - 31 *Regular* exits and one *Error* exit.

Settings

1. Box Identification

The name of the Script Box.

2. Properties

- Number of exits Defines the number of Regular exits (1 - 31) for the box. After the entire script in the Script Box has been executed, the exit defined by the last executed BoxOut function is taken.

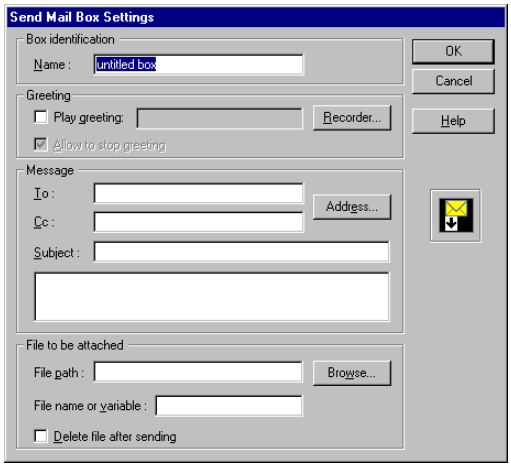
Editor window

Contains the text of the Script Language routine. When the OK button is clicked, Smartphone Server checks the script. Any invalid statement will generate an error message.

Expression builder icon

Opens the Expression builder dialog.

Send Mail Box



The **Send Mail Box** provides the following functions:

Play a greeting. The greeting might inform the caller that an email message will be sent.

Send a text message to an Email address. The Email address can be specified as a string or as a string variable.

Send a file as an attachment to the message. The file path and name can be specified as a string or as a string variable. The specified file can be automatically deleted after being sent.

The **Send Mail Box** can be used for:

New message notification. When a caller leaves (records) a message, the text message in the "Mail options" box is sent to the address(es) specified in the "To" and "Cc" fields. The recorded message can also be automatically forwarded (as an audio file attachment to the text message) by specifying the path/file name (as a variable) of the recorded message.

Auto-forward of received messages. A user working at a remote location can auto forward all received messages to an email account by specifying the path/file name (as a variable) of a recorded message.

Settings

1. Box Identification.

The name of the Send Mail Box.

2. Greetings.

- **Play Greeting** The VAP plays the specified greeting (the path of the greeting is relative to the folder of the current VAP). Click on Recorder to choose a file for the greeting or to record the greeting.

This field requires a Smartphone Server Expression of string type.

- **Allow to stop Greeting** The user can stop the greeting by pressing a telephone key.

3. Message.

- **To** The email address of the recipient of the message. This field requires a Smartphone Server Expression of string type. Multiple addresses should be separated with a semicolon and space ("; ").

- **Cc** The email address of the recipient of a copy of the message. This field requires a Smartphone Server Expression of string type. Multiple addresses should be separated with a semicolon and space ("; ").

- **Subject** The subject of the messages. This field requires a Smartphone Server Expression of string type.

- **Message** The text of the messages. This field requires a Smartphone Server Expression of string type.

4. File to be attached.

- **File Path** The path of the file to be attached to the email message (relative to the folder of the current VAP). Click on Browse to select the path. For example, if the path to (folder of) your VAP is "C:\Program Files\Smartphone Server\VAP":

Entering the path "Messages\" specifies the path "C:\Program Files\Smartphone Server\VAP\Messages".

Entering the path "C:\Base\" specifies the path "C:\Base".

This field needs a Smartphone Server Expression of string type.

- **File Name or Variable** The file name or variable (that contains a file name) for the message to attach. If the variable includes a path, the path set in *Recording Path* is ignored.

This field needs a Smartphone Server Expression of string type.

- **Delete file after sending** Check this checkbox if the file included with the email message (as an attachment) should be deleted after the email message has been sent.

Chapter 11: Lexicon Toolkit Overview

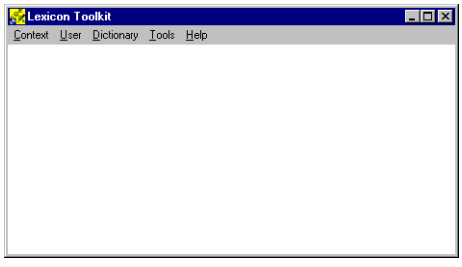
The Lexicon Toolkit (LexTool) is a tool for managing contexts provided by Lernout & Hauspie Corp. A context can be created, deleted, imported from a grammar file and exported to some target platform format.

LexTool allows the editing of a context's lexicon, so words with phonetic transcriptions can be added, deleted and changed. The phonetic transcriptions can be generated automatically, using an underlying conversion engine and exception dictionaries. The user can edit and adapt these transcriptions.

LexTool contains the ability to manage users. A user can be created, deleted, renamed and copied. Contexts can be copied between users. A user can be registered or unregistered for using languages and contexts. Also, a user can be exported, so that his/her speech characteristics can be installed later.

LexTool enables exception dictionary management. An exception dictionary can be created, edited, deleted, renamed, copied and exported to be installed at a later time.

Lexicon Toolkit main window:



Only part of the functions are explained in the Smartphone Server documentation. Please use the online help for further information. Some things are described in Smartphone Server online help, and overall, Lexicon Toolkit has its own help file.

The build-up of voice recognition

The symbols L0 to L6 are reserved for the Smartphone Server system prompts. (L0-user defined, L1-German, L2-English, L3-French, L4-Italian and L6-Russian). These symbols also contain the voice recognition information for the following words:

Table 1:

0 to 9	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
Star	(*)
Pound sign	(#)
Yes	(Y)
No	(N)
Stop	(S)

Should you wish to create your own words, we recommend using a new context and to save such words in this context. You can build both "Individual words"- and "Connected digits" (continuous recognition)-type vocabularies. The names of all standard vocabularies which use continuos recognition starts with "C" symbol.

This type of recognition is NOT supported by Smartphone Server. The term "context" in Lexicon Toolkit environment means the same as "vocabulary" in Smartphone Server program. So to get a new words collection you must create a new context in LexTool. Here is how to go about it:

Creating a new Context

Start Lex Tool and choose Context from the menu with the command den "New...". The following window will appear:

The fields of data entry are explained below:

- Database: Because voice recognition runs over the telephone, the words are saved in a telephone-specific format.
- User: The standard Smph User is alleged.
- Language: Choose the language of the new words here. All previously installed languages will be listed here.
- Syntax: Specify here in which context the new words are to be used and recognized.

Table 2:

Isolated Words:	Recognition of single words runs in this mode.
Key Word Recognition:	The words will be taken from the key word recognition vocabulary.
Context:	Enter the name of the new context here.

Click “OK“.

How to Add Words to a Context

Click Add.

Enter in the following information :

Orthographic Expression: This is where to enter the new word.

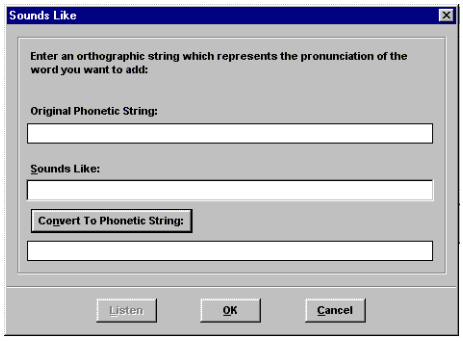
Context Class: The way the voice recognition is to understand the word is entered here, just as in “Syntax”.

Phonetic Expressions: This is for the phonetics translation of the new word to enable voice recognition without training. A compilation of phonetic writing is found at the end of this section. The translation can also be turned over to Lex Tool by clicking on “Convert”. Should you not find the correct pronunciation of the word within “Orthographic Expression“, use “Sounds Like“.

Convert: Translate the word in “Orthographic Expression“ (or number) in to voice recognition -specific phonetic writing.

Add Another: Click here to add another word.

Sounds Like: If the word is to be pronounced other than as it is in “Orthographic Expression“ , then use this to specify the pronunciation. This is recommended in most cases.



Enter in the pronunciation of the word. By using “Convert to Phonetic String:“ the expression will be directly translated into phonetic spelling. Click “OK” to add the word into the dictionary.

Deleting Words from the Context:

Here is how to delete a word from the dictionary :

Start Lex Tool.

Choose from the Context menu the command “Open...“.

Choose the dictionary in which the word to be deleted is found. (See list (Dropdown) Context) and press “OK”.

Mark the expression to be deleted and press “delete“. The expression has now been deleted.

How to Change Expressions:

To change an expression:

1 Start Lex Tool.

2 Choose from the Context menu the command “Open...“..

3 Choose the dictionary in which the word to be changed is found. (See list (Dropdown) Context) and press “OK”.

4 Mark the expression to be deleted and press “change“. The expression has now been changed.

Now the phone can be given. Follow the same steps as by adding a new expression.

! For more information about Lexicon Toolkit consult its online help system.

Appendixes

Appendix A: Script Language Statements

‘break’ statement

```
break;
```

Exits a for or while loop.

Parameters

None.

Remarks

Execution continues with the first statement after the innermost loop that contains the "break" statement.

A "break" not included in a for or while loop will generate an error.

‘continue’ statement

```
continue;
```

Causes execution to continue at the beginning of a for or while loop.

Parameters

None.

Remarks

Execution continues at the beginning of the loop that contains the "continue" statement (in a "for" loop, execution continues with the third statement in the parantheses in the "for" statement (typically loop counter incrementation)).

If a "continue" statement is encountered in a nested loop, the "continue" is interpreted as applying to the innermost loop.

A "continue" not included in a for or while loop will cause Smartphone Server to report an error.

‘float’ statement

```
float <Variable>[=<expression>][,
<Variable>[=<expression>]]…;
```

Defines a local variable(s) of type float.

Parameters

variable: The name of the variable.

expression: (optional) The initial value of the variable.

Remarks

Multiple values can be declared simultaneously.

Type float specifies a fixed 15-digit mantissa and a 3-digit exponent.
Range:5.0 * 10e-324 < x < 1.7 * 10e+308.

‘for’ statement

```
for(<statement1>; <statement2>;
<statement3>)
<statement4>
```

Defines a "for" loop.

Parameters

statement1: int b=0, cnt=0;

The singular statement that is executed before the loop is started (not a part of the loop). Normally the initialization of a counter. See remark below.

statement2: cnt < 10 && b<100;

The required condition to continue execution at the beginning of the loop (a singular statement that normally compares the counter with another value).

statement3: cnt+=[a=1, cnt+1], b=12*b

The statement that is executed after the last statement in the loop (a singular statement that normally increments or decrements the counter). See remark below.

statement4:

The body of the loop (a singular statement or a compound statement (multiple singular statements enclosed in brackets: "{}")).

Remarks

"statement1" and "statement3" may use the comma operator within an expression. For example "c=1, b=2" is an expression with the comma operator.

'if…else' statement

```
if( <expression> )
<statement1>
[else
<statement2>]
```

Defines an "if else" conditional branch.

Parameters

expression: The boolean condition that determines which branch will be executed.

statement1: The statement (singular or compound) that is executed if "expression" is true.

statement2: (optional) The statement (singular or compound) that is executed if "expression" is false.

'int' statement

```
int <Variable>[=<expression>][,
s<Variable>[=<expression>]]…;
```

Defines a local variable(s) of type integer.

Parameters

variable: The name of the variable.

expression: (optional) The initial value of the variable.

Remarks

Multiple values can be declared simultaneously.

Range:-2147483648 < x < 2147483647 (signed 32-bit).

'return' statement

```
return [<expression>];
```

Causes the Script Language program to terminate immediately.

Parameters

expression: (optional) The integer value returned to Smartphone Server.

'switch...case...default' statement

```
switch(<Expression>)
{
   [case <Expression1>:
   <statement1>]. . .
   [default:
   <statement_default>]
}
```

Executes an expression (singular or compound) from a list of expressions if the condition for the expression matches the condition following the switch statement.

Parameters

expression: The condition that determines which expression is executed.

expression1: The expression whose condition must equal to that of "expression" in order for statement1 to be executed.

statement1: The statement (singular or compound) that is executed if the boolean condition of "expression1" is equal to that of "expression".

statement_default: (optional) The statement that is executed if no previous statements were executed.

Remarks

Any number of case statements (including no case statement) is allowed.

"break" statements are not required.

A statement is required following the "case" expression (not ANSI C standard).

Expressions may be a variable, expression, or constant (ANSI C allows only a constant) as long as all expressions are of the same type.

'string' statement

```
string <Variable>[=<expression>][,
<Variable>[=<expression>]]…;
```

Defines a local variable(s) of type string.

Parameters

variable: The name of the variable.

expression: (optional) The initial value of the variable (a string expression).

Remarks

"expression" can include constants and other (previously defined) variables. For example: string v1="Good ", v2=v1+"bye".

'while' statement

```
while(<expression>)
   <statement>
```

Defines a "while" loop.

Parameters

expression: A boolean condition that must be true for the pending repetition of the loop to be executed. If false, execution is continued at the first statement after the while loop.

statement: The statement (singular or compound) that is executed during each repetition of the loop.

Appendix B: Script Language Functions

Access function

```
int Access( string path_filename );
```

Verifies that the specified file exists.

Return value (integer)

true: File exists.

false: File does not exist.

Parameters

path_filename: (string) Full path and filename of the file.

BoxOut function

```
BoxOut( int exit );
```

Sets the Script Box exit to take upon completion of the Script Box routine.

Return value

None.

Parameters

exit: (integer) The number of the Script Box exit.

Remarks

The quantity of Script Box exits is set in the "Properties / Number of Exits" field.

Call_vap function

```
string call_vap(string alias);
```

Starts the execution of the Voice Application (VAP) that was previously loaded from disk into memory and assigned a specified symbolic name (alias) with the load_vap function.

Return value (string)

OK:VAP executed successfully.

E_CALL:VAP was called incorrectly (e.g. not from a Script Box).

E_FAIL: execution of the VAP with the specified alias could not be started (e.g. no loaded VAP has the specified alias or the VAP is written to start at a different time, etc.).

Parameters

alias: (string)a valid and existing symbolic name assigned to the VAP previously loaded into memory.

Remarks

This function may be called from the Script Box only.

Please remember that Smartphone Server supports only five levels of nested VAP calls. This means that if one VAP loads and calls a second one, the second one loads and calls a third one and so on, only the first five VAPs in this sequence will be loaded. Other VAPs will not be loaded and calls will be refused (E_FAIL will be returned).

Any VAP can be called only on the telephone line on which it was initially loaded.

CanDial function

```
int CanDial( string ext );
```

This function checks if the given telphone number is on the Smartphone Server black list.

Return value (integer)

true :
The specified telephone number is not on the black list and thus can be dialed.

false :
The specified telephone number is on the black list and thus cannot be dialed.

Parameters

ext (string):
The telephone number to search for on the Smartphone Server black list. The string may contain either the whole number or only part of it (following the same rule as the numbers on the black list).

Remarks

None.

Char function

```
string Char( int ascii );
```

Returns a string with a single character.

Return value (string)

Specified ASCII character.

Parameters

ascii: (integer) ASCII code (decimal) for the character.

CharAt function

```
int CharAt( string source_string, int
position );
```

Returns the ASCII code of the character at the specified position in the specified string.

Return value (integer)

ASCII code of the character in the string.

Parameters

source_string: (string) The string that contains the character.

position: (integer) The position in the string of the character (the first character in the string is at position 0) If the string is entered incorrectly or the wrong position is used, the return value will read 0.

Clock function

```
int Clock( );
```

Returns the elapsed time (msec) since Windows NT/2000/XP was started.

Return value (integer)

The elapsed time (msec) since Windows NT/2000/XP was started.

Parameters

None.

ClrDtmfs function

```
ClrDtmfs( );
```

Clears the input and output line buffers on the Dialogic boards.

Return value

None.

Parameters

None.

Remarks

This function should be called before functions such as Playback, Record, etc. to ensure that the Dialogic buffers are empty.

ClrHangup function

```
ClrHangup( );
```

Clears the Hangup attribute (set when a caller hangs up). Allows VAP execution to continue if execution was halted due to a caller hangup.

Return value

None.

Parameters

None.

Remarks

When a caller hangs up while execution is in a Script Box, the "Hangup" attribute is set. Upon exiting this Script Box, one of the following will occur:

If Global Hangup is defined: Execution branches to the Hangup Box with the "Global Hangup" option checked.

If Global Hangup is not defined: Execution of the VAP stops.

CreateFileName function

```
string CreateFileName( string folder[,
string prefix=""[, string
extension=".vox"]] );
```

Returns a unique filename (created automatically) that includes the specified prefix (optional) or file type extension (optional).

Return value (string)

The filename (without path).

Parameters

folder: (string) The destination directory for the created file.

prefix: (optional) (string) The name of the file without the file extension. If not specified, a unique filename is created automatically (recommended).

extension: (string) The file extension.

Remarks

VAP's running on different lines should use this function to create unique filenames.

Day function

```
int Day( );
```

Returns the number (1-366) of the current day in the year.

Return value (integer)

The number (1-366) of the current day in the year.

Parameters

None.

Delay function

```
Delay( int time );
```

Causes VAP execution to pause for the specified time period.

Return value

None.

Parameters

time: (integer) The length (in 50msec units) of the pause.

Dial function

```
int Dial( string phone, int rings );
```

Calls the specified phone number.

Return value (integer)

Specifies the result of the attempted call:

-1 - error

0 - answered by voice

1 - busy

2 - no answer

3 - answered by fax machine

4 - answered by answering machine

5 - special information tone (SIT) detected

Note: To enable detection of a SIT, it must be described in a call progress initial file "custom.ini". For further information see Smartphone Server Administrator's Guide.

Parameters

phone: (string) The phone number to call.

rings: (integer) The maximum number of rings before cancelling (hanging up) an unanswered call.

DialPostBusy function

```
string DialPostBusy( );
```

Returns the dialing sequence needed to reconnect to the previous phone if the current number (dialed using a flash hook) is busy.

Return value (string)

The dialing sequence needed to reconnect to the previous phone if the current dialed number is busy.

Parameters

None.

Remarks

The dialing sequence is configured for the Tenovis/Bosch PBX and should be dialed using the "Dial" function.

DialPostNA function

```
string DialPostNA( );
```

Returns the dialing sequence needed to reconnect to the previous phone if the current number (dialed using a flash hook) is not answering.

Return value (string)

The dialing sequence needed to reconnect to the previous phone if the current dialed number is not answering.

Parameters

None.

Remarks

The dialing sequence is configured for the Tenovis/Bosch PBX and should be dialed using the "Dial" function.

FaxAdd function

```
int FaxAdd( string filename );
```

Adds a file to the fax queue.

Return value (integer)

true: The file was successfully added to the queue.
false: An error occurred.

Parameters

filename: (string) The filename of the file to be sent.

Remarks

Only those file types that are supported by your Dialogic board can be sent (consult your Dialogic documentation).
Up to 100 documents per line can be held in the queue.

FaxCheck function

```
int FaxCheck( [int linenumber=0] );
```

Determines if the fax option is supported on the specified telephone line.

Return value (integer)

true: The fax option is supported on the specified telephone line.
false: The fax option is not supported on the specified telephone line.

Parameters

linenumber: (integer) (optional: default = 0) The number of the telephone line to check (If 0 or if not specified: the line for the current VAP is checked).

Remarks

Sometimes only a portion of the Dialogic boards in a computer support the fax option. If such a situation is possible, support for fax should be verified before adding a file in the fax queue.

FaxClear function

```
FaxClear( );
```

Clears the fax queue for the line for the current VAP.

Return value

None.

Parameters

None.

Remarks

Files are only removed from the fax queue (not deleted).

FaxGetCount function

```
int FaxGetCount( );
```

Returns the number of fax documents in the queue for the telephone line for the current VAP.

Return value (integer)

The number of fax documents in the queue for the telephone line for the current VAP.

Parameters

None.

FaxGetFile function

```
string FaxGetFile( int index );
```

Returns the filename of the file in the specified position in the fax queue for the current telephone line.

Return value (string)

The filename of the file in the specified position in the fax queue.

Parameters

index: (integer) The position in the fax queue of the specified file (0 for the first file in the fax queue).

Remarks

All faxes in a queue are sent as a group. Therefore, indexes do not change.

FaxGetPhone function

```
string FaxGetPhone( );
```

Returns the destination fax phone number for the fax queue for the current telephone line.

Return value (string)

The destination fax phone number for the fax queue for the current telephone line.

Parameters

None.

Remarks

All faxes in the queue for the current telephone line are sent to a single fax phone number.

FaxPrint function

```
int FaxPrint( string tifffile );
```

Prints the specified TIFF/F file on the default printer of the host computer.

Return value (integer)

true: The file was successfully sent to the operating system to be printed.
false: An error occurred.

Parameters

tifffile: (string) The filename of the TIFF/F file to print.

Remarks

If the operating system encounters an error while trying to print the file, the return value will not indicate the error.
The print procedure should be specified in the Windows NT/2000/XP Registry ‘Print’ entry for the ".tif" extension.
For Windows NT with "Imaging" accessory option installed: Imaging is used to print TIFF/F fax documents.

FaxPrintOK function

```
int FaxPrintOK( );
```

Checks if the FaxPrint function supports printing of TIFF/F files.

Return value (integer)

true: The FaxPrint function supports printing of TIFF/F files.
false: The FaxPrint function does not support printing of TIFF/F files.

Parameters

None.

Remarks

In the Windows NT/2000/XP registry the ‘Print’ entry for the ".tif" file extension is required.

FaxReceive function

```
int FaxReceive( string filename );
```

Receives a fax and saves to the specified file.

Return value (integer)

true: The fax was received and saved to the specified file.
false: The fax was not received.

Parameters

filename: (string) The filename of the file to save the fax to.

If the fax turnaround detection feature is enabled (in the “smphone.ini” file), then after the function has executed the specific value will be stored in the LastErrorInt and LastErrorStr variables (if a fax polling request was received):

LastErrorInt=-1

LastErrorStr="Box: rec fax: Line: 1: Error: -1 <Polling request detected.>"

This LastErrorInt variable should be checked to detect and handle the turnaround polling request.

FaxSend function

```
int FaxSend( [string phone=""] );
```

If no phone number is specified: Immediately sends the current faxes in the fax queue on the current phone line.
If phone number is specified: Sends the faxes in the current fax queue to the specified phone number after the current call has ended normally.

Return value (integer)

true: The fax queue was sent successfully.
false: An error occurred.

Parameters

phone: (string) (optional) The phone number to send the fax queue to (if not specified, the faxes are sent on the current line).

Remarks

If the faxes are sent successfully: The FaxClear function is executed automatically to clear the fax queue.

FCopy function

```
int FCopy( string dest, string source );
```

Creates a copy (in the specified directory with the specified filename) of the specified file.

Return value (integer)

true: The file was copied successfully.
false: An error occurred.

Parameters

dest: (string) The directory and filename of the copy of the original file.
source: (string) The directory and filename of the file to be copied.

FMerge function

```
int FMerge( string dest, string source );
```

Concatenates the second specified file to the end of the first specified file.

Return value (integer)

true: No error.
false: An error occurred.

Parameters

dest: (string) The name of the file to concatenate to it.
source: (string) The name of the file that will be concatenated to the end of the file dspecified by dest parameter.

FRename function

```
int FRename( string dest, string source
);
```

Renames and/or moves the specified file or directory (if a directory is moved, all subdirectories and files in those subdirectories are moved also).

Return value (integer)

true: No error.
false: An error occurred.

Parameters

dest: (string) The directory (optional) and filename of the renamed and/or moved file.
source: (string) The directory (optional: if not

specified, the current directory is assumed) and filename of the file to be renamed.

Remarks

A file cannot be moved to a different disk drive.

To simply rename a file: Do not specify the directory for dest.

To simply move the file: Specify the same filename for source and dest.

GetDefLang function

```
int GetDefLang( int type = 1 );
```

Returns the Smartphone Server system language or the default language set in Voicemail/Unified Messaging.

Return value (integer)

Specifies the language:

0 - user defined language

1 - German

2 - English UK

6 - Russian

10 - English U.S.

12 - Spanish Latin America

Parameters

type: (integer) Specifies which language setting is required:

0:For the Smartphone Server system language

1:For Voicemail/Unified Messaging (Voicemail/Unified Messaging menu Voicemail/Unified Messaging / Settings / System greeting: Language)

Remarks

If the language for the VAP is changed using the LANGUAGE system variable, the value returned by GetDefLang will not reflect this change.

<language> system variable

GetDigit function

```
string GetDigit( int timeout, int DoBeep
= 1 );
```

Inputs 1 DTMF digit from the caller.

Return value (string)

DTMF digit received.

Parameters

timeout: (integer) Timeout (sec) for caller input.

DoBeep: (integer) If DoBeep = 0, no beep is being played (DoBeep = FALSE). In all other cases a beep is played to prompt the caller for input (DoBeep = TRUE).

GetDigits function

```
string GetDigits( int quantity, int
timeout, string OKCancel = "*#", int
DoBeep = 1, string
FirstSymbolsOfImmediateExit );
```

Inputs a specified number of DTMF digits from the caller.

Return value (string)

The string returned depends on which of the following DTMF digits is received:

OK character (character 1 of the OKCancel string; usually "*"): All recorded DTMF digits except the current OK symbol are returned.

CANCEL character (character 2 of the OKCancel string; usually "#"): Only the CANCEL symbol is returned.

Any character in the *FirstSymbolsOfImmediateExit string: The last DTMF digit received.*

The Nth DTMF digit (where N = quantity): All DTMF digits received.

When one of the above DTMF digits is received, input is halted.

Parameters

quantity: (integer) The maximum number of DTMF digits to receive.

timeout: (integer) The length (sec) of an input pause that will cause input to stop.

OKCancel: (string) A 2 character string. Normally set to "*#" (star key + pound key).

Character 1 specifies the DTMF digit that will be interpreted as "OK" input (accept input) (normally "*").

Character 2 specifies the DTMF digit that will be interpreted as "CANCEL" (do not accept input) (normally "#").

DoBeep: (integer) If DoBeep = 0, no beep is being played (DoBeep = FALSE). In all other cases a beep is played to prompt the caller for input (DoBeep = TRUE).

FirstSymbolsOfImmediateExit: (string) A multi-character string. Normally set to "" (an empty string). If the first DTMF digit entered by the caller matches any digit in the string, input terminates and this first DTMF digit that was entered is returned.

GetProfileInt function

```
int GetProfileInt( string filename,
string section, string entry, int default
);
```

Returns the integer representation of the specified profile entry (integer or string) in the specified section of the specified file.

Return value (integer)

The specified entry (integer or string) in the specified section of the specified file. If the entry, section, or file is not found, the value specified by variable "default" is returned.

Parameters

filename: (string) The filename of the file (usually an *.ini file) where the needed profile entry is located. If this parameter is an empty string, the main Smartphone initial file “smphone.ini“ is used.

section: (string) The section in the file where the needed profile entry is located.

entry: (string) The needed profile entry.

default: (integer) The default profile entry if the specified file, section, or entry is not found.

GetPromptFile function

```
string GetPromptFile( int promptnumber[,
int language=-1[, int worktype=-1]] );
```

Returns the path and filename of the specified Voicemail/Unified Messaging prompt.

Return value (string)

The path and filename of the specified prompt. If the prompt doesn’t exist, an empty string is returned.

Parameters

promptnumber: (integer) The number of the prompt (0…999).

language: (integer) The language of the prompt:

-1 or not specified: The system language (specified by the system variable LANGUAGE).

0 - user defined language

1 - German

2 - English UK

6 - Russian

10 - English U.S.

12 - Spanish Latin Americaworktype: (integer) The time period of the prompt:

-1 - the current time period within Voicemail/Unified Messaging (if no VAP is currently running, the "day" time period is used).

0 - day

1 - night

2 - weekend

3 - holiday

Remarks

Prompts 0…499 are reserved for the Voicemail/Unified Messaging system.

GetSysFile function

```
string GetSysFile( string SysPromptName,
int language );
```

Returns the path and filename of the specified system prompt.

Return value (string)

The path and filename of the specified system prompt.

Parameters

SysPromptName: (string) The filename prefix ONLY (without the directory or file type extension) of the system prompt.

language: (integer) The language of the system prompt:

-1 or not specified: The system language (specified by the system variable LANGUAGE).

0 - user defined language

1 - German

2 - English UK

6 - Russian

10 - English U.S.

12 - Spanish Latin America

GetVar function

```
string GetVar( string varname );
```

Returns the value (string) of the specified variable (global or shared).

Return value (string)

The value of the specified variable (global or shared).

Parameters

varname: (string) The name of the global or shared variable.

GetVMIInt function

```
int GetVMIInt( string section, string
entry );
```

Returns the integer representation of the specified entry (integer or string) in the specified section of the Voicemail/Unified Messaging configuration file "config.vmi".

Return value (integer)

The specified entry in the specified section of the file "config.vmi".

Parameters

section: (string) The section in "config.vmi" where the needed entry is located.

entry: (string) The needed profile entry.

GetVMIString function

```
string GetVMIString( string section,
string entry );
```

Returns the string representation of the specified entry (integer or string) in the specified section of the Voicemail/Unified Messaging configuration file "config.vmi".

Return value (string)

The specified entry in the specified section of the file "config.vmi".

Parameters

section: (string) The section in "config.vmi" where the needed entry is located.

entry: (string) The needed profile entry.

Hangup function

```
Hangup( );
```

Disconnects (hangs up) the current line.

Return value

None.

Parameters

None.

Remarks

The HangUp attribute is set when the line is disconnected.

IsHangup function

```
int IsHangup( );
```

Returns a non-0 integer if the HangUp attribute is set to "on-hook" (line is disconnected).

Return value (integer)

true: HangUp attribute is set to "on-hook" (line has been disconnected).
false: HangUp attribute is set to "off-hook" (line is connected).

Parameters

None.

Remarks

The HangUp attribute is checked (the actual phone line is not checked). This requires less time than the "IsOffHook" function, but sometimes returns a value that does not reflect the actual condition of the line.

IsOffHook function

```
int IsOffHook( int linenumber = -1 );
```

Returns **true** if the specified phone line is busy ("off-hook").

Return value (integer)

true: The specified phone line is busy ("off-hook").
false: The specified phone line is not busy ("on-hook") or does not exist.

Parameters

linenumber: (integer) The line to check.

Remarks

The actual phone line is checked (the HangUp attribute is not checked). This requires more time than the "IsHangup" function, but always returns a value that reflects the actual condition of the line.

Load_vap function

```
int load_vap(string alias, string
vap_file_name);
```

Loads the Voice Application (VAP) from the specified file on disk into memory and associates the specified symbolic name (alias) with it, which other functions can now use to call this VAP as many times as they need it executed.

Return value (integer)

true: VAP loaded successfully

false: loading of the VAP failed

Parameters

alias: (string)the symbolic name by which the loaded VAP can now be referenced from other functions.

vap_file_name: (string)the valid path and file name for the VAP to be loaded. If the specified string is not a full path to a file, it will be considered relative to the folder of the current VAP.

Remarks

This function is intended for use with VAPs that are supposed to be executed frequently but may take a long time to be loaded and to initialize and/or close their working environment (such as "Unified Messaging.vap"). Now such VAPs can be loaded and initialized only once and executed without delay (with the call_vap function) at any time when another application needs them.

An attempt to call this function with the file name of a VAP that it is not yet loaded and an alias that is already assigned to a loaded VAP will associate the alias with the newly loaded VAP and unload the one that owned the alias before.

Log function

```
Log( string text );
```

Outputs the specified string to the log window.

Return value

None.

Parameters

text: (string) The text to be output to the log window.

MessageBox function

```
int MessageBox( string text, string
title, int flags );
```

Displays a Message box with the specified style and text.

Return value (integer)

Indicates which button in the Message box was selected:

0 - System error.

1 - OK.

2 - Cancel.

3 - Abort.

4 - Retry.

5 - Ignore.

6 - Yes.

7 - No.

Parameters

text: (string) Text to be displayed in the Message box.

title: (string) Text to be displayed as the Message box title.

flags: (integer) Type of Message box buttons and icons to display. This value corresponds to the standard Windows NT/2000/XP codes for Message box formats. The most commonly used codes are:

Table 1: Codes

"OK" button	0x0000
"OK" & "Cancel" buttons	0x0001
"Abort", "Retry" & "Ignore" buttons	0x0002
"Yes", "No" & "Cancel" buttons	0x0003
"Yes" & "No" buttons	0x0004
"Retry" & "Cancel" buttons	0x0005
"Help" button	0x4000

Table 1: Codes

Icon "Hand"	0x0010
Icon "Question"	0x0020
Icon "Exclamation"	0x0030
Icon "Asterisk"	0x0040
User icon	0x0080

To display the required buttons and icon, add the corresponding values from the table above.

Remarks

The Message box closes if any button in the box is pressed.

MsgBox function

```
MsgBox( string text );
```

Displays a Message box with the specified text, the "OK" button, and no icon.

Return value

None.

Parameters

text: (string) Text to be displayed in the Message box.

Remarks

For a Message box with additional features see the MessageBox function.

Playback function

```
Playback( string filename[, int async=0]
);
```

Plays a prompt file on the active line.

Return value

None.

Parameters

filename: (string) Name of the prompt file.

async=: (integer) (optional) Specifies the playback mode:

0 (default): synchronous: The prompt is played in entirety before continuing execution of the Script.

1: asynchronous: The execution of the Script is continued immediately after the prompt starts playing. The Script continues to execute until the first Wait is encountered in the Script.

Remarks

1. Asynchronous mode is useful because background functions (such as database access) can be executed while the prompt is played.

2. The function works only when the call is in the “connected” state. The call is considered to be in the “connected “ state during the following periods:

- For incoming calls: from going off-hook to hang up.
- For outgoing calls: from the moment the connection to the called party is set up (after off-hook and call progress) until hang up.

It is possible to enforce the “conected” state during call progress (it is needed in some applications, for example, music on hold) by calling the LineSetParam function before the PlayBack function call:

```
LineSetParam(2301,1);
```

But this is not recommended, because may cause memory leaks.

PlayPrompt function

```
PlayPrompt( int PromptNum[, int async=0]
);
```

Plays a Voicemail/Unified Messaging prompt.

Return value

None.

Parameters

PromptNum: (integer) Number of the prompt (0…999).

async=: (integer) (optional) Specifies the playback mode:

0 (default): synchronous: The prompt is played in entirety before continuing execution of the Script.

1: asynchronous: The execution of the Script is continued immediately after the prompt starts playing. The Script continues to execute until the first Wait is encountered in the Script.

Remarks

Asynchronous mode is useful because background functions (such as database access) can be executed while the prompt is played.

Prompts 0…499 are reserved for the Voicemail/Unified Messaging system.

The language of the prompts played is the language defined in the system variable LANGUAGE.

The worktype of the played prompts is:

If the current VAP is running: The worktype defined in Voicemail/Unified Messaging Settings.

If the current VAP is not running: Day (0).

PlaySys function

```
PlaySys( string SysPromptName );
```

Plays a system prompt.

Return value

None.

Parameters

SysPromptName: (string) Filename (without the path and filename extension) of the system prompt to play.

Random function

```
int Random( int range );
```

Returns an integer random number within the specified range.

Return value (integer)

Random number within the specified range.

Parameters

range: (integer) Range: 0 <= x <= (range-1)

Record function

```
Record( string filename, int DoBeep, int
maxtime, int maxsilence );
```

Records a voice file.

Return value

None.

Parameters

filename: (string) Name of the file to record.

DoBeep: (integer) If true: a beep is played to prompt the caller for input.

maxtime: (integer) Maximum length (sec) of the recorded file.

maxsilence: (integer) Length (sec) of silence on the voice line that will be accepted as end of input.

Remarks

The voice file format (VOX (low quality) or WAV (high quality)) is defined in: Voice Appli-

cation Settings / Recording / Low Quality (VOX) | High Quality (WAV).

SetHangup function

```
SetHangup( );
```

Sets the HangUp attribute.

Return value

None.

Parameters

None.

Remarks

If the HangUp attribute is set:

if Global Hangup **defined in this VAP:** After this Script Box has finished execution branches to the Hangup Box with the "Global Hangup" option checked.

if Global Hangup **not defined:** VAP execution will halt after the current Script Box has finished .

SetVarNum function

```
SetVarNum( string varname, int value );
```

Assigns an integer value to the specified global or shared integer variable.

Return value

None.

Parameters

varname: (string) Name of the integer variable (Global or Shared) to assign.

value: (integer) Value to assign to the variable.

Remarks

"varname" must specify an integer variable. If a string variable is specified the value will not be assigned and no error message will be displayed.

SetVarStr function

```
SetVarStr( string varname, string value
);
```

Assigns a string value to the specified global or shared string variable.

Return value

None.

Parameters

varname: (string) Name of the string variable (Global or Shared) to assign.

value: (string) Value to assign to the variable.

Remarks

"varname" must specify an string variable. If an integer variable is specified the value will not be assigned and no error message will be displayed.

Sleep function

```
Sleep( int time );
```

Pauses execution of the VAP for the specified length of time.

Return value

None.

Parameters

time: (integer) Length of time (sec) to pause VAP execution.

SMS_send function

The Script Language function. All functions

```
string SMS_send( string address, string
message_text );
```

Sends a specified text message to a mobile phone with the specified number via SMS (GSM Short Message Service).

Return value

The text string describing a response from the GSM modem is as follows:

- “OK“ - success.
- “E_PORT“ - error opening the COM port indicated for the GSM modem in the Smartphone Server SMS settings.
- “E_NORESPONSE“ - COM port opened successfully, but there is no response from modem.
- “E_PIN“ - the modem has not acccepted the PIN code specified in the Smartphone Server SMS settings. PIN card will be locked after three failed attempts to enter the PIN code.
- “E_ERROR“ - the modem returned “ERROR“.
- “E_UNKNOWN” - unrecognizable response recieved from the modem.
- “E_NOTINIT“ - the modem is ready, but the PIN code and SMS Message Centre number are missing.
- “E_ARGS“ - the function has been called with incorrect argument(s).
- “E_FAIL“ - operation has failed due to some serious system failure (software or hardware).
- “E_CALL“ - the function has been called from somewhere other than a function box.

Parameters

address: (string) Mobile phone number of the message recipient.

message_test: (string) Text of the message to be send to the recipient.

Remarks

- The total length of a message text must not exceed 160 characters (SMS technology restriction).
- Sending SMS messages is possible if either SMS messaging is set up in the Smartphone Server Unified Messaging settings or the line **StartSMSServices=1** is included in the [SMS] section of the smphone.ini file.

Speak function

```
Speak( string text[, int async=0] );
```

Speech is generated on the phone line from the specified text.

Return value

None.

Parameters

text: (string) Text to be converted into speech.

async=: (integer) (optional) Specifies the playback mode:

0 (default): synchronous: The prompt is played in entirety before continuing execution of the Script.

1: asynchronous: The execution of the Script is continued immediately after the prompt starts playing. The Script continues to execute until the first Wait is encountered in the Script.

Remarks

The text string can contain the following control characters:

\s: separate number ("123*4" is read as "one two three star four")

\n: composed number ("123" is read as "one hundred twenty three"; the maximum number is 999,999,999)

\t: time (format hh:mm)

\d: date (format mm.dd.yy)

\r: relative date (format mm.dd.yy)

\f: system prompts (without extension)

\p: existing prompt file (full path must be specified)

There should be no space between a control symbol (for example: '\s') and the text.

The space between items is required.

Asynchronous mode is useful because background functions (such as database access) can be executed while the prompt is played.

StrLen function

```
int StrLen( string Str );
```

Returns the length of the specified string.

Return value (integer)

Length of the specified string.

Parameters

Str: (string) The string.

StrStr function

```
int StrStr( string Str, string Substring
);
```

Returns the position of the first character of the first occurrence of the specified substring in the specified string (if it exists).

Return value (integer)

The position of the first character of the first occurrence of the specified substring in the specified string (if the string is not found: -1 is returned).

Parameters

Str: (string) String that contains the substring.

Substring: (string) Substring to be searched for in the string.

Remarks

The position of the first character in a string is 0.

Null (ASCII 0) character marks the end of a string.

StrUpr function

```
string StrUpr( string source );
```

Returns a string that contains the characters in the specified string converted to uppercase.

Return value (string)

Contains the characters in the specified string converted to uppercase.

Parameters

source: (string) Source string (possibly with lowercase characters).

SubStr function

```
string SubStr( string Str, int index, int
count );
```

Returns a string (substring) containing the specified number of characters starting at the specified position in the specified source string.

Return value (string)

Substring containing the specified numbers of characters starting at the specified position in the specified source string.

Parameters

Str: (string) Source string.

index: (integer) Starting position of the substring in the source string.

count: (integer) Number of characters in the substring.

Remarks

The position of the first character in a string is 0.

Unlink function

```
Unlink( string filename );
```

Deletes the specified file.

Return value

None.

Parameters

filename: (string) Path and filename of the file to delete.

Wait function

```
Wait( );
```

Pauses the execution of a Script program if a prompt is currently being played on the phone line in asynchronous mode. Execution of the Script program continues at the first statement after the Wait statement after the prompt has finished playing.

Return value

None.

Parameters

None.

Remarks

Wait is used with the Playback and Speak functions.

WeekDay function

```
int WeekDay( );
```

Returns the current day of the week.

Return value (integer)

Current day of the week:

1 - Monday

2 - Tuesday

3 - Wednesday

4 - Thursday

5 - Friday

6 - Saturday

7 - Sunday

Parameters

None.

WinExec function

```
int WinExec( string filename, string
commandline[, int showflag=1] );
```

Starts the specified Windows NT/2000/XP application with the specified input file (optional).

Return value (integer)

true: No error.
false: An error occurred.

Parameters

filename: (string) Path (required) and full filename of the input file for the application specified in the string "commandline".

"filename" must be an empty string ("") if:

No input file is required by the application.

The input file is specified in the string "commandline".

commandline: (string) Command line parameters including:

Path (optional) and filename of Windows NT/ 2000/XP application: If the path is not specified, the executable file will be searched for in the following directories:

1. The Smartphone Server directory.

2a. The 32-bit Windows system directory (SYSTEM32).

2b. The 16-bit Windows system directory (SYSTEM).

3. The Windows directory.

4. The directories that are listed in the PATH environment variable.

(optional) Path (required) and filename of the input file for the application: Not required if:

No input file is required by the application.

The input file is specified in the string "filename".

Showflag: (integer) (default: 1) Specifies the style of the program window:

Table 2:

HIDE	0
SHOWNORMAL	1
NORMAL	1
SHOWMINIMIZED	2
SHOWMAXIMIZED	3
MAXIMIZE	3
SHOWNOACTIVATE	4
SHOW	5
MINIMIZE	6
SHOWMINNOACTIVE	7
SHOWNA	8
RESTORE	9
SHOWDEFAULT	10

Remarks

An input file with long file names that includes spaces should be specified in the "filename" field in order to avoid possible confusion in the "commandline" field.

Write function

```
int Write( string filename, string Str );
```

Appends the specified string to the specified file.

Return value (integer)

True: No error occurred.

false: Error occurred.

Parameters

filename: (string) Filename of the file to append the string to.

Str: (string) String to append to the file.

WriteVMIInt function

```
WriteVMIInt( string section, string
entry, int value );
```

Writes an integer value as the specified entry in the specified section of the Voicemail/Unified Messaging configuration file "config.vmi".

Return value

None.

Parameters

section: (string) Section in "config.vmi" where the value should be written.

entry: (string) Profile entry.

value: (integer) Value for the profile entry.

WriteVMIString function

```
WriteVMIString( string section, string
entry, string value );
```

Writes a string value as the specified entry in the specified section of the Voicemail/Unified Messaging configuration file "config.vmi".

Return value

None.

Parameters

section: (string) Section in "config.vmi" where the value should be written.

entry: (string) Profile entry.

value: (string) Value for the profile entry.

Appendix A: Call Transfer Functions

TransferCall function

```
int TransferCall( string phone, int
rings, string filename1, string filename2
);
```

Tries to connect the party initiating the call ("Caller") to the specified phone number ("Called party"). In the event the transfer is not completed or is interrupted, returns the flow of operations to the Caller. The function plays corresponding prompts for both parties.

Return value (integer)

Specifies the result of the attempted transfer:

-1 - the transfer has not yet been initiated and there is nobody on the line (the Caller has already disconnected or is not yet connected)

0 - the Called party (transfer destination) hung up or a pre-defined interrupting DTMF digit was pressed by the Called party

1 - the Called party's phone is busy or no free channels are available for transfer

2 - the Called party's phone is not answering

3 - the transfer was initiated and the Caller hung up

4 - a pre-defined interrupting DTMF digit was pressed by the Caller

Parameters

phone: (string) The phone number to which the call should be transferred.

rings: (integer) The maximum number of rings before the unanswered call to the Called party is cancelled with a no answer result.

filename1: (string) The name of the .wav or .vox file containing the prompt to be played to the Caller while the Called party is being dialled.

filename2: (string) The name of the .wav or .vox file containing the prompt to be played to the Called party before transferring the call once it has been dialled.

Remarks

The SCbus is used for transferring calls.

In order for the function to work properly, the number of channels to be used for transferring calls must be specified in the Smartphone Server smphone.ini file in the section [General] in the key "LinesForTransfer". The number of channels specified will be taken from the end of the available channels pool. For example, if there are 30 available channels, and 10 channels are specified for transferring calls, channels 21-30 will be used for transferring calls for lines 1-20.

The duration in seconds of a call that was successfully transferred will be written into the system variable "TransferCallDuration".

If the conversation between the Caller and the Called party is interrupted, the operation flow returns to the Caller and the Called party is disconnected. If the Caller hangs up, the Voice Application will finish after the Script Box.

TransferSetParam function

```
TransferSetParam( string CallerDTMF,
string CalledDTMF, string RecordVOX,
string StartRecordDTMF, string
StopRecordDTMF );
```

Defines parameters for the call transfer operation performed by the ‘TransferCall’ function. These parameters describe two optional features:

Interrupting the transfer by pressing a DTMF digit (first two parameters): it is possible for a Caller and for a Called party to interrupt the transfer during the conversation by pressing a pre-defined key. If no keys are defined, it is not possible to interrupt the transfer with a DTMF.

Intercept Recording (last three parameters): this feature allows a conversation between a Caller and Called party to be recorded to a voice file. Recording can be started either automatically when the connection between the Caller and the Called party is established, or by the Caller pressing a pre-defined DTMF digit. Recording may be stopped by the Caller or the Called party by pressing a pre-defined DTMF key.

Return value (integer)

None.

Parameters

CallerDTMF: (string) The DTMF digit that a Caller should dial to interrupt the transfer.

CalledDTMF: (string) The DTMF digit that a Called party should dial to be disconnected.

RecordVOX: (string) The filename to which a conversation between a Caller and Called party (both voices will be written). The file format will be always VOX. The file path is optional. If it is not mentioned, the current Windows NT/2000/XP path will be used. If the file already exists, it will be overwritten.

StartRecordDTMF: (string) The DTMF digit that a Caller should dial to start recording.

StopRecordDTMF: (string) The DTMF digit that should be dialed to stop recording.

Remarks

All parameters are valid only for the nearest call of the TransferCall function. If you are going to call the TransferCall function again you must also call the "TransferSetParam" function again.

If multiple characters are specified in start/stop DTMF digit strings, only the first character will be used.

During the recording all start/stop digits may be used by both the Caller and the Called party. E.g., the Called party can interrupt recording by pressing a digit specified as ‘CallerDTMF’. In this case the TransferCall function may return a wrong value.

Recording can be started and stopped repeatedly during the single call transfer.

Appendix B: OLE Functions

Create function

```
int auto::Create( string sClass );
```

This creates a new object of the class specified with "sClass" string.

Return value (integer)

0 (zero): object not created

Object Handle Number: the number assigned as an identifier to the successfully created object. The object handle number will be used by other functions to refer to the object.

Parameters

sClass : (string) A string identifying the class of an OLE object to be created (it must match exactly to the class definition in your Windows NT/2000/XP registry), either in text (ProductID) or numeric (GUID -Globally Unique Identifier) form:

ProductID: a text string specifying the OLE server application for the object, type of the object and (optionally) a version number, for example: "Outlook.Application", "Access.Application", "Word.Document.6"

GUID: 32 (8+4+4+4+12) hexadecimal digits (uppercase) in curly brackets, for example:

"{00000010-0000-0010-8000-00AA006D2EA4}"

Remarks

The function accepts both text and numeric OLE object class identifiers in *string* format. If indicating a GUID, be sure to include set brackets and to quote the whole string.

PropGet function

```
variant auto::PropGet( int hObj, string
Prop );
```

This receives a value of a requested property of a specified OLE object.

Return value (variant)

property value: The value of the requested property. The type of the returned value is object/property-dependent and is treated as *variant that can be automatically converted to any valid expression type.*

Parameters

hObj : (integer) The handle number assigned to the requested object on creation.
Prop : (string) The name of a requested property.

Remarks

If the value returned by PropGet is a pointer to another object, it will be automatically converted to an OLE object handle number format that may be used by Smartphone Server Script Language OLE automation functions (see hObj parameter above).

PropSet function

```
int auto::PropSet( int hObj, string Prop,
variant Value );
```

This sets the specified property of the specified OLE object to a specified value.

Return value (integer)

true : if the property was set successfuly.

false : if the property was not set successfuly.

Parameters

hObj : (integer) The handle number assigned to the requested object on creation.

Prop : (string) The name of the requested property.

Value : (variant) The value to which the requested property should be set.

Invoke function

```
variant auto::Invoke( int hObj, string
Method, <parameters> );
```

This invokes the requested method of the specified OLE object with a variant number of parameters (if required).

Return value (variant)

The value returned by the requested object. The type of the value is object/method-dependent and is treated as *variant.*

Parameters

hObj : (integer) The handle number assigned to the requested object upon creation.

Method : (string) The name of the requested method.

Parameters : (variant) Parameters that should be passed to a requested method.

AddRef function

```
int auto::AddRef( int hObj );
```

This increments the number of references to a specified OLE object by 1. It allows the user to add a reference to an open OLE object without creating an extra copy of OLE server application for it.

Return value (integer)

The actual number of references to the requested object after incrementation.

Parameters

hObj : (integer) The handle number assigned to the requested object on creation.

Remarks

Any application that uses an OLE object must create a reference to it. When the object is created with the Create function, the reference is created automatically. If any function is going to access an object that already exists (has been created by some other function or process), AddRef must be called to add a reference to such object for this function. Once the function does not need access to such object any longer, the Release function must be used to eliminate the reference to the object that exists for that function.

Release function

```
int auto::Release( int hObj );
```

Reduces the number of references to a specified OLE object by 1.

Return value (integer)

The actual number of references left on the requested object after they have been reduced.

Parameters

hObj : (integer) The handle number assigned to the requested object upon creation.

Remarks

If the reduced number of references to an object equals 0, the object is deleted.

Appendix C: Databases Access Functions

Connect function

```
int db::Connect( string DSN );
```

Opens a connection to a database indicated by "DSN".

Return value (integer)

true: connection opened successfully

false: connection opening failed

Parameters

DSN: (string) A string identifying a database to connect to. May present one of the following:

- Registered ODBC data source name (DSN) (set up in your Windows NT/2000/XP Control Panel ODBC settings).
- Explicit MS Access database file name (doesn't require a registered DSN for this database file).
- ODBC connection string - for a database that requires an authorised connection or / and has no registered DSN in your Windows NT/2000/XP ODBS settings and is not an MS Access database file.

A connection string must begin with "DSN=", "DBQ=", "UID=" or "PWD=", for example: "DSN=Voicemail;" or "DSN=MS Access Database; DBQ=C:\Program Files\Smartphone Server\sp-stat.mdb; ".

Remarks

Only one active database connection with not more than 16 opened recordsets may simultaneously exist for each VAP.

Disconnect function

```
int db::Disconnect();
```

Closes an existing database connection opened with the use of "Connect" function.

Return value (integer)

true: connection closed successfully

false: connection closure failed

Parameters

none

Remarks

All open recordsets will be closed correctly before the connection is closed.

Execute function

```
int db::Execute( int recordset,string sql
);
```

Executes SQL query to a database for which the connection has been opened from the current VAP.
Unlike the Database box that allows you to execute SELECT queries only, the Execute may be used to execute ***any*** valid SQL query.

Return value (integer)

true: SQL query execution success

false: SQL query execution failure

Parameters

recordset : (integer):

For SELECT queries:

a number (from **0** to **15**) that will be assigned to a recordset if opened successfully. Any further reference to this recordset from any other function will be made by this number, and no other recordset will be able to use this number until this recordset is closed (with Drop function).

For other SQL queries: **-1**

sql : (string)

Any string that presents a valid text of an SQL query or a stored procedure call, for example:
"SELECT * FROM Users WHERE Code BETWEEN 110 AND 911;"
or
"INSERT INTO Users (Code, Name) VALUES ("911", "Rescue");"

"{call StoredProcSelectAll}"

Remarks

If the Execute function is called to open a new recordset with the number that was already assigned to some other open recordset, the recordset that had this number is closed automatically, and the requested number is assigned to a new recordset.

Queries that do not return recordsets (CREATE, INSERT, UPDATE etc.) still do use temporary ones during execution. Such temporary recordset is automatically assigned a first free number from 0 to 15. Since no recordsets should be returned by such query, its temporary recordset is closed when the query execution ends, and the auto-assigned number becomes available again.

If you have all numbers from 0 to 15 assigned to open recordsets and you run the Execute function with "-1" as a first parameter, this new query will not automatically close any recordset in order to get a free number. In this situation, a new query will not be able to open a temporary recordset and the Execute function will return an error.

Drop function

```
int db::Drop( int recordset );
```

Closes a recordset previously opened with SELECT SQL query.

Return value (integer)

true: recordset closed successfully

false: recordset closure failed.

Parameters

recordset : (integer)A number assigned to a recordset to be closed.

Remarks

Since the pool of recordset numbers is limited (0-15), it is a good practice to close any recorset with Drop function once you don't need it open any more. It may help you to avoid the situation when the Execute function with "-1" as a first parameter returns an error because the query is unable to open a temporary recordset due to lack of free numbers to be assigned to it automatically.

Fetch function

```
int db::Fetch( int recordset, int go, int
param=0 );
```

Navigates through the records in the open recordset.

Return value (integer)

true: successful move

false: move failed

Parameters

recordset : (integer) A number (from 0 to 15) assigned to the recordset you are going to navigate through;

go : One of the following constants:

Table 1:

Go_First	Moves to the first record in the recordset.
Go_Next	Moves to the next record (one record towards the end of the recordset)..
Go_Prev	Moves to the previous record (one record towards the beginning of the recordset).
Go_Last	Moves to the last record in the recordset.
Go_Abs	Moves to a record with the absolute position in the recordset indicated by **param**.
Go_Rel	Moves to a record with offset indicated by **param** (positive or negative) from the current record.

param : (integer)

For "Go_Abs" and "Go_Rel" parameters only:

"Go_Rel" : a number of records (positive or negative) to move for.
A positive value of **param** means that a move will be made in the "forward" direction (towards the end of the recordset), and the negative value of this parameter means a move in a "backward" direction (towards the beginning of the recordset).

"Go_Abs" : an absolute number of a record to move to.

Remarks

"To move to a record" means to make it a current record.

If Fetch returns the **false** value, it may mean that the beginning or the end of a recordset has been reached. You may use the Error function to analyse the nature of an error.

Get function

```
string db::Get( int recordset, string field );
```

Reads data from the "Field" recordset column of the current record.

Return value (string)

Field value : if operation was successful.

Empty string : if an error occurred.

Parameters

recordset : (integer)A number (from 0 to 15) assigned to the recordset you are going to read data from;

field :(string)A name of the recordset column you are going to read data from.

Remarks

The data types supported by Get function depend on the implementation of the ODBC driver used for the particular database connection.

Set function

```
int db::Set( int recordset, string field, string value );
```

Assigns the value presented by the "value" parameter to a field in the current record in the recordset indicated by the "recordset" parameter and in the column named as specified by the "field" parameter.

Return value (integer)

true: success

false: failure

Parameters

recordset : (integer)A number (from 0 to 15) assigned to the recordset you are going to write data to.

field :(string)A name of the recordset column you are going to write data to.

value :(string)Data to be assigned to the "field".

Remarks

The data types supported by Set function depend on the implementation of the ODBC driver used for the particular database connection.

Error function

```
int db::Error( string& s );
```

Assigns the description of the last error (returned by the ODBC driver) to the string "s".

Return value (integer)

0 (zero): no error

positive value: warning

negative value: error

Parameters

s :(string&) A string variable that will contain an error description.

Remarks

A description of an error is received from ODBC driver and returned "as is". This means that the text stored in the "s" string depends on the ODBC driver that is used for a particular connection to a database.

Delete function

```
int db::Delete( int recordset );
```

Deletes the current record from the recordset indicated by the "recordset" parameter.

Return value (integer)

true: success

false: failure

Parameters

recordset :(integer) A number assigned to the recordset from which you are going to delete a current record.

Remarks

After the current record in a recordset is deleted, no special action is taken to define some other record as a new current one. In order to make next, previous or some particular record in the recordset a new current record, always use the Fetch function with respective parameters after you execute Delete. To choose the correct parameters for the Fetch function, assume the deleted record is still the current one.

Appendix D: List management functions

Detach function

```
int list::Detach(string Key, int List
Status=-1);
```

Removes an item from a list.

Return value (integer)

true :
item has been removed.

false :
The given key is an empty string or the item is not found on the list.

Parameters

Key (string):

string to match the key variable of the required item.

ListStatus (integer):
0 - global list (available from any line);

-1 (default) or any other non-zero value - only available from the current line.

Remarks

None.

Clear function

```
void list::Clear(int List Status=-1);
```

Removes all items from the list.

Return value (integer)

true :
all items has been removed.

false :
error removing items from the list.

Parameters

ListStatus (integer):
0 - global list (available from any line);

-1 - (default) or any other non-zero value - only available from the current line.

Remarks

None.

Count function

```
int list::Count(int List Status=-1);
```

Counts the number of items on the list.

Return value (integer)

Number of items on the list.

Parameters

ListStatus (integer):
0 - global list (available from any line);

-1 - (default) or any other non-zero value - only available from the current line..

Remarks

None.

Get function

```
string list::Get(string Key, int List
Status=-1);
```

Searches the list by a given key variable and returns the value of the corresponding list item.

Return value (string)

Item value:
the corresponding item has been found on the list;

Empty string:
the corresponding item has not been found or the key variable is an empty string.

Parameters

Key (string): string to match the key of the required item .

ListStatus (integer):
0 - global list (available from any line);

-1 - (default) or any other non-zero value - only available from the current line.).

Remarks

None.

Set function

```
int list::Set(string Key, string Value,
int List Status=-1);
```

Sets the new value of the list item that corresponds to the given key

or

Adds the item to the list if the key is not found.

Return value (int)

true :
list item has been added/updated successfully.

false :
the given key is an empty string.

Parameters

Key (string):

string to match the key of the required item .

Value (string): New list item value.

ListStatus (integer):
0 - global list (available from any line);

-1 - (default) or any other non-zero value - only available from the current line..

Remarks

None.

Load function

```
int list::Load(string FileName, int List
Status=-1);
```

Loads a list from a disk file with a given name.

Return value (integer)

true :
list has been loaded successfully.

false :
file was not found or there was an error opening it.

Parameters

FileName (string):
any text string that presents a valid file name.

ListStatus (integer):
0 - global list (available from any line);

-1 - (default) or any other non-zero value - only available from the current line..

Remarks

None.

Save function

```
int list::Save(string FileName, int List
Status=-1);
```

Saves a list to a disk file with a given name.

Return value (integer)

true :
list has been saved successfully.

false :
file was not found or there was an error in saving it.

Parameters

FileName (string):
any text string that represents a valid file name.

ListStatus (integer):
0 - global list (available from any line);
-1 - (default) or any other non-zero value - only available from the current line.

Remarks

If the file with the given name exist, its contents will be overwritten.

If the file does not exist, one will be created.

Appendix E: Date and time management finctions

Get function

```
int dt::Get(datetime InDate, int Param);
```

Extracts a numeric value of seconds, minutes, hours, day, month or year from a value of given date an/or time, given in a specific ‘datetime’ format, according to the value of Param argument.

Return value (integer)

A numeric value of date/time component specified by Param argument.

Parameters

InDate (datetime):
A valid presentation of some moment of time in the ‘datetime’ format.

Param (integer):
0 or **‘s’**- seconds;

1 or **‘m’**- minutes;

2 or **‘h’**- hours;

3 or **‘d’**- day of month;

4 or **‘M’**- month;

5 or **‘y’**- year.

Remarks

None.

Add function

```
datetime dt::Add(datetime InDate, int n,
int Param);
```

Adds a specified numeric value to a given date/time. A numeric value to be added may be interpreted as a number of seconds, minutes, hours, or days depending on the value of Param argument.

Return value (datetime)

A resulting value in the ‘datetime’ format.

Parameters

InDate (datetime):
A valid presentation of some moment of time in the ‘datetime’ format.

Param (integer):
0 or **‘s’**- seconds;

1 or **‘m’**- minutes;

2 or **‘h’**- hours;

3 or **‘d’**- full days (24 hours).

Remarks

None.

Sub function

```
int dt::Sub(datetime Date1, datetime
Date2, int Param);
```

Finds the difference between two given moments of time. The resulting value can present the number of seconds, minutes, hours or days depending on the value of the Param argument.

Return value (integer)

A resulting difference in the ‘integer’ format.

Parameters

Date1 (datetime):
A valid presentation (in the ‘datetime’ format) of the moment of time from which to subtract.

Date2 (datetime):
A valid presentation (in the ‘datetime’ format) of the moment of time to be subtracted.

Param (integer):
0 or **‘s’**- seconds;

1 or **‘m’**- minutes;

2 or **‘h’**- hours;

3 or **‘d’**- full days (24 hours).

Remarks

A return value presents the integer number of full days (24 hours), full hours, full minutes or seconds. (Example: If the actual difference between Date1 and Date2 is 4 hours 30 minutes and 15 seconds, the Sub function returns for this time would be 0 for Param=’d’, 4 for Param=’h’, 270 for Param=’m’ or 16215 for Param=’s’.

ToStr function

```
string dt::ToStr(datetime InDate, string
sDateFormat='', string sTimeFormat='',
int Flags=3);
```

Converts the value of a given moment of time presented in the ‘datetime’ format to a text string. The format of date/time presentation in an output text depends on the value of Flags argument.

Return value (string)

A text string that presents a given moment of time in a specified format.

Parameters

InDate (datetime):
A valid presentation (in the ‘datetime’ format) of the moment of time to convert.

sDateFormat (string):
A format mask for an output date string (e.g. “yyyy MMMM dd (ddd)“).

sTimeFormat (string):
A format mask for an output time string (e.g. “hh:mm“).

Flags (integer):
the result of the boolean ‘OR’ operation applied to the required combination of the following values:

0x01 - short date format (cannot be used with 0x04);

0x02 - time format;

0x04 - long date format (cannot be used with 0x01);

Remarks

If no string is specified for the sDateFormat or the sTimeFormat argument, the default Windows settings for date/time format will be used.

Appendix F: Glossary

This is a glossary that explains all specific terms, abbreviations and acronyms that the user of the Smartphone family products may want to know. It includes computer telephony, general PC terminology and terminology specific to Smartphone. It may contain more information you might require. It can be consulted to get explanations of unfamiliar terms.

A

AAVM

Auto Attended Voicemail, a system that allows callers to leave messages. Registered users can save, delete or listen to the messages in their mailboxes by interacting with an automated attendant.

ACD

Automated Call Distribution, the routing of an incoming telephone call to the next available operator or agent.

ADPCM

Adaptive Differential PCM (Pulse Code Modulation), an advanced technique for encoding sound in digital format. Instead of coding an absolute measurement at each sample point, it codes the difference between samples and can dynamically switch the coding scale to compensate for variations in amplitude and frequency.

Agent

A person who answers any calls routed to a certain phone in an ACD system. Agents typically log in to an ACD system so that the system knows which agents are available to answer calls.

ASCEU, ASCF, ASCFC, ASCGB

Analog Subscriber Circuit card in the Tenovis/BOSCH PBX (for Europe (ASCEU), France (ASCF and ASCFC), Great Britain (ASCGB)). The card provides a physical interface to an external analog phone line.

ASR

Automated Speech Recognition. In general, speech recognition is used in computer science to allow the user to use voice commands instead of a keyboard interface.

ATA, ATB, ATC

Analog Trunk Interface card in the Tenovis/BOSCH PBX (different versions of the card are used in different countries).

ATLC

Analog TIE Line Circuit card in the Tenovis/BOSCH PBX.

Audiotex (audiotext)

A voice response application that allows users to enter and retrieve information over the telephone. In response to a voice menu, users press keys or answer questions to select their way down a path of choices. Audiotex systems can sometimes provide database access via telephone. Interactive audiotex provides complete instructions that guide the caller through an information selection process.

Automated Attendant

A sub-system of Smartphone that can answer incoming calls and direct them to their proper extension. If the person being called is unable to answer the call, Automated Attendant can also let callers to transfer to another extension or leave a voice message.

B

BLT

See Board Locator Technology.

Board Locator Technology (BLT)

Board technology from Dialogic that allows the determination and initialization of various board settings via software. Usually only a Board Identification Switch must be physically set on the board.

Box

A standard functional element in a VAP that is visually displayed in the VAP Editor window as a box. There are several different types of standard boxes available. The functionality of a box is determined by setting box fields and switches and by connecting the box to other boxes.

Boxbar

A panel within the VAP Editor that displays all available VAP boxes. A box may be added to a VAP with a drag & drop.

Busy

A busy signal is generated by the telephone switch to the calling party when the called party telephone line is already busy with another call. The typical busy signal includes 0.5 second tones with 0.5 second intervals between them. The busy signal can vary from country to country.

See also Disconnect.

C

Call Center

A highly-integrated customer specific solution for simplifying call processing. A Call Center can use a variety of different technologies, and usually provides system and personnel performance monitoring.

Call Progress

Call Progress settings are used to match the telephone signals on analog lines generated by a PBX with the signal environment of the PBX (typically determined by the country).

Caller ID

A service that provides the number of the calling party to the called party.

Central Office (CO)

The local public telephone switching station for a certain area.

See also PBX.

Client

The application that submits requests for information or service to the server application in the client-server framework.

Client-Server

A software architecture in which operation splits between two component programs and their respective communication elements: the client and the server. The clients communicate directly with the server, which processes all requests from the clients.

CLIP

Calling Line Identification Presentation, the standard in Europe for providing caller party information.

See also CLIR.

CLIR

Calling Line Identification Restriction, the ability of a caller to block the transmission of caller information to the dialed number (see CLIP).

CO

See Central Office.

CPU

Central Processing Unit, the electronic component of the computer that ultimately controls the functions carried out by the computer.

CTI

Computer Telephone Integration, the technology of integrating computers with a telephone switch so that the computer can automate many of the tasks associated with processing calls.

D

D/41ESC, D/41ESC+

A type of Dialogic telephone board. Officially described by Dialogic as an “International SCSA 4-Port Voice Processing Board”. The letters in the designation mean the following:

- D: Dialogic

- 4: 4 lines (or “ports”).
- E: Europe (designed for international use in Europe).
- SC: Supports the SCbus.
- +: “Plus”. Supports additional features.

DCOM

Distributed Component Object Model (DCOM) extends the Microsoft Component Object Model (COM) to support distributed model of object-oriented component technology. DCOM supports both local area networks (LAN) and wide area networks (WAN) (e.g. the Internet).

Dialogic

The leading manufacturer of computer telephony boards worldwide. The Dialogic product line includes the following boards supported by Smartphone Server: D/41ESC, ProLine/2V, VFX/40ESC.

Dialtone

The tone heard in the telephone handset after the handset has been picked up (if the phone was not ringing). A dialtone is an audio signal generated by the telephone switch to indicate that the telephone switch is ready to accept telephone input (via touch pad or rotary dial).

Disconnect

A disconnect tone is the signal (typically similar to the busy signal) that a PBX or CO generates when the opposite party hooks on the line.

DID

Direct Inward Dialing, a method of transmitting information, typically about a type of call and the person who called party. The information is sent from one telephone switch to either a telephone or another telephone switch by transmitting DTMF signals immediately after the call is answered.

DPD

Dial Pulse Detection, the detection of rotary dial pulses generated on an active voice line. Tones generated by pressing a key on a touch-tone keypad are transmitted on an active voice line because the spectrum of DTMF tone frequencies generated lies completely within the range of transmitted voice frequencies. However, only a limited portion of the frequency spectrum of a pulse signal generated by a moving rotary dial lies within the spectrum transmitted by telephone switches. Therefore, the pulses can only be detected by searching the transmitted voice spectrum for a set of frequency components typically generated by a rotary dial. This method of detection is inherently error prone and requires a high level of signal processing to distinguish dial pulses from line noise or voice.

DPD is supported by Dialogic boards either in hardware (a board with a designation including “DPD”) or in software (with the “Dialogic DPD Enablement Kit” software).

Drag & Drop

A graphical user interface (GUI) capability that lets you perform operations by moving the icon of an object or function with the mouse into another window or onto another icon. A typical drag & drop operation would be the following:

Position the mouse cursor over an object on the screen. Press and hold the left mouse button. The object is “dragged” across the screen as the mouse moves. When the object is in the proper location on the screen, the mouse button is released, and the object has been “dropped” at the necessary location.

DS0

Digital Linecard S0, a card in the Tenovis/BOSCH PBX. The card provides a physical interface for the ISDN line.

DSN (Data source name)

An ODBC term for the collection of information used to connect your application to a particular ODBC database. The ODBC Driver Manager uses this information to create a connection to the database. A DSN can be stored in a file (a file DSN) or in the Windows Registry (a machine DSN).

DTMF

Dual Tone Multi-Frequency, a term used to described a touch tone signal that consists of 2 separate signals of equal amplitude but with different frequency.

E

F

Fax message

A class of message containing a digitized image (TIFF/F format).

Firmware

A category of program code that runs on intelligent peripheral devices such as Dialogic boards, not on a CPU, and which defines the basic operations of such devices. Dialogic firmware is stored in *.fwl files. Dialogic drivers download it to the on-board memory on the drivers' start-up.

See also HW, SW.

G

H

Hangup

The act of finishing a call or "going on hook."

Hookflash

An analog signal on a telephone line that is interpreted by the telephone switch as a command or the start of a command sequence (with further commands in the sequence typically being DTMF signals). A hookflash is typically generated by quickly hanging up and then picking up the telephone handset. "Flash" refers to the speed of the action. "Hook" refers to the physical hooks for the handsets on the first telephones. Hookflashes are most commonly used to switch between callers or to forward a call.

HW

Hardware, machinery and equipment (CPUs, boards, disks, tapes, modems, cables, etc.). In operation, a computer is both hardware and software. One is useless without the other. The hardware design specifies the commands it can follow, and the instructions tell it what to do.

See also firmware, SW.

I

IC

See "Intercept".

Intercept

An Intercept tone (IC), or Operator Intercept, is the tritone signal that a PBX or CO generates to indicate an operator intercept. It is an SIT encoding.

See also SIT.

Intranet

A local office TCP/IP computer network.

IPN

Intelligent Private Network module in the Tenovis/ BOSCH PBX.

IRQ

Interrupt Request, a hardware interrupt to the CPU. A device (PC board) can request the attention of the CPU by driving an IRQ pin on the CPU (assuming that the board is connected to an IRQ line). The CPU will "immediately" process the IRQ by executing the program (usually the driver for the board) situated in the memory location allocated specifically for that IRQ line.

IVR

Interactive Voice Response, a method of connecting a caller to a computer system by providing information in reaction to a caller's input. The IVR system provides directions or menu options in the form of voice prompts that allow the caller to specify the required information by pressing telephone keys and/or stating voice commands.

J

Jumper

A removable metal strip that is used to connect 2 pins on a PC board. The presence or absence of a jumper is used

by a processor (the CPU processor or board processor) to determine the configuration of hardware or resources (IRQ, memory, I/O, etc.).

K

Keypad

The area of a telephone that includes the buttons pressed to generate dialing codes (pulse or DTMF digits, etc.).

L

L&H

Lernout & Hauspie, the company that produced the Automated Speech Recognition (ASR) and Text-To-Speech (TTS) software included with Smartphone Server.

LAN

Local area network. A data communications network that is geographically limited (typically to a distance of 1 km), allowing easy interconnection of computers.

M

Mail server

Software that manages all messages in a system and responds to requests of clients. For example, Microsoft Exchange Server.

Mailbox

The destination for messages sent to a user.

MDAC

Microsoft Data Access Components (MDAC) is a redistributable set of technologies that implement the Universal Data Access strategy. MDAC consists of the latest versions of ActiveX Data Objects (ADO), OLE DB components, and Open Database Connectivity (ODBC), which have now been released as an integrated set. Developers creating client/server and Web-based data-driven solutions select the components they need to create complete database solutions.

Menubar

The list of pop-up menus located at the top of the main window of the Windows application (the Smartphone Server menu bar includes ‘File’, ‘Edit’, etc.).

Message source

A creating platform (telephone, networked computer, etc.) used to send a message to a messaging system.

Middleware

A type of intermediary software that connects different parts of application software in a standard way. It is often used to link the client and server parts of an application. COM, DCOM, DCOM+, ODBC, ActiveX, RPC, CORBA are the most popular middleware standards.

Modem

A device that allows a computer to send information (fax, digital data) or voice over a telephone line.

MS Exchange Client

Microsoft Exchange Client is software that manages different kinds of messages from different sources (including MS Exchange Server) in an Inbox.

MS Exchange Server

Microsoft Exchange Server is a powerful mail server that provides advanced mail, scheduling and groupware applications. It manages all kinds of messages (including voice and fax) and responds to the requests of the networked clients. It can be accessed by MS Exchange Client software. MS Exchange Server can be connected to the LAN or Internet and can work as a gateway between MS Exchange and Internet mail standards.

MWI

Message Waiting Indicator, a symbol on the phone display or the light on the phone set that indicates that the user has a new message.

N

Notification

A short message (text or voice) sent to a user automatically to alert the user that a new message has been received.

O

ODBC

Open Database Connectivity. The Microsoft standard for access to databases from applications. Using ODBC software and database-specific drivers, it is possible to get transparent access to any type of database. Access to databases in ODBC is based on SQL. The 32-bit version of ODBC is provided with Smartphone.

Off Hook

"Off hook" means that a line is activated (a handset has been picked up).

On Hook

"On hook" means that a line is on standby and ready to recieve calls.

Operator Intercept

See "Intercept".

P

PBX (PABX)

Private [Access/Automatic] Branch Exchange, a private telephone switch that performs many of the switching functions traditionally provided only by telephone companies.

See also Central Office.

Private Playback

A special function of Voicemail/Unified Messaging software that allows the user to play a voice message stored in a mailbox on his/her private phone set by clicking a button on the voice form.

PLMN

Public Land Mobile Network. Wireless telephone network.

See also PSTN.

Proline/2V

An inexpensive Dialogic telephone board.

Prompt

An audio message generated from a digitized audio file (VOX, WAV format) that provides information or gives a caller instructions. Smartphone Server provides a full suite of tools for recording and managing prompts. Prompts can be both played and recorded from within a VAP.

PSTN

Public Switched/Service Telephone Network, which includes all public telephone switches. Lines provided by public switches are available for any customers in a certain geographical area.

See also PLMN.

Pulse dial

One of the two methods available (the other being touch-tone dial) for transmitting telephone number digits from an analog phone to a phone switch. The term "pulse dial" originates from the round "dial" on older telephones and the electrical "pulses" created as the dial returns to its original position. The pulses are actually short breaks in a DC current loop.

See also DTMF.

Q

R

Recordset

A set of records in a database that matches the selection criteria of a query.

Reorder

A reorder tone (RO), also called a fast busy or system busy, is the tri-tone signal that a PBX or CO generates to indicate that the switching path is busy. It is an SIT encoding.

Ring

A ring signal is generated by the telephone switch to the called party when the local switch has received a request (from another switch) to complete a call to the called party. The local switch generates a 60 Hz AC signal that causes the mechanical bell in the called party telephone to ring (digital telephone systems emulate this analog method of ringing a bell). The ring signal varies from country to country. The typical ring signal includes alternating 0.7 second bell ring intervals and 4.3 second no-ring intervals.

See also Ring Back.

Ring back

A ring back signal is generated by the telephone switch and is sent to the calling party phone. The ring back signal is generated only after the switch local to the called party has signalled the switch local to the calling party that the phone of the called party is ringing.

See also Ring.

RJ-11

A single line telephone jack termination with 4 wires.

RJ-14

A dual line telephone jack termination with 4 wires.

RO

This abbreviation has two meanings:

1 Read-only.

2 See “Reorder”.

S

SAPI

Speech Application Programming Interface - middleware supported by Microsoft Corp. It provides an API abstraction layer between applications like Smartphone and speech engines like Lernout & Hauspie. SAPI is used for both universal text-to-speech (TTS) and automatic speech recognition (ASR).

SCbus

A bus implemented by Dialogic for data exchange at a single SCSA host. Dialogic boards in one PC communicate via SCbus. A 131 Mbps data path that provides up to 2,048 time slots, the equivalent of 1,024 two-way voice conversations at 64 Kbps.

SCSA

Signal Computing System Architecture, an open software and hardware standard for digital signal processing. It is widely used in computer telephony and supported by Dialogic Corp. SCSA hosts consist of 3 basic modules: an interface module for telephone network connection, DSP processors, and software applications. The backbone of an SCSA host is the SCbus.

Server

The application that processes requests from client applications for information or service in a client-server framework.

SIT

Special Information Tones. SIT is a tri-tone signal generated by the PBX or CO. Detection of an SIT sequence indicates an operator intercept or other problem in completing the call. At least four SIT encodings have been defined: No Circuit found (NC), Vacant Code (or Vacant Circuit) (VC), Operator Intercept (IC), Reorder (RO).

SMS

Short Message Service, a service in mobile telecom networks (basically digital, basically in GSM standard) that allows the transmission of short text messages (approximately 160 characters) to another phone number, pager or email address.

SQL

Structured Query Language, a commonly used language for accessing databases. It is used in ODBC, for instance.

SPC

Smartphone Client, a software application by NOVAVOX AG that runs on the user’s computer. NOVAVOX AG Smartphone Client software provides a connection to

the Smartphone Server computer that is responsible for voice and fax messages, and properly presents all kinds of messages to the user via the installed email messaging client application (typically Microsoft Exchange Client or Outlook).

SPS

Smartphone Server, a software application by NOVAVOX AG that is responsible for voice and fax messages and runs on the computer that contains the telephony boards which interface directly with the internal and public telephone networks. SPS provides TUI to the users.

Status bar

The area at the bottom of a window where status information is displayed. For example, when the cursor is placed over any menu item, a program will typically display information about that menu item in the status bar.

SW

Software, a set of instructions to run on processor(s). These instructions are stored on disk in executable files. All computer programs are software.

See also HW, firmware.

T

TAPI

Telephony Applications Programming Interface (TAPI) is a convergence platform, which enables applications to provide a consistent user experience even when the applications are running on various topologies such as the Public Switched Telephone Network (PSTN), ISDN, PBX systems, and TCP/IP networks.

Text message

A any written message (with optionally attached files of any type) sent through a computer network (for example, an email message).

Toolbar

The toolbar is located near the top of the application main window below the menubar. Icons in the toolbar most often represent commonly used commands.

Touch-tone pad

See “keypad”.

TTS

Text-To-Speech, technology that generates speech from the text representation of speech.

TUI

Telephone User Interface, the user interface provided to the system via a telephone. Compare: GUI - Graphical User Interface.

U

UM

Unified Messaging. UM is a service provided with special hardware and software that unifies previously separate messaging systems (email, voice, and fax). UM includes software that connects the email messaging system with a voicemail server (which provides voice and fax messaging). The UM solution from NOVAVOX AG provides seamless integration with a user interface that hides the inherent complexity of messaging system unification from the user.

User

A person registered as a user on a UM system. A user can access his messages (email, voice, fax) stored on the Smartphone Server or Microsoft Exchange Server via his computer (with Smartphone Client software), phone (internal or external), or fax machine.

User ID

A unique sequence of characters that identify a user’s mailbox.

V

VAP

Voice Application, a visual program for processing phone line activity, executed by Smartphone Server. A VAP is created and edited with the VAP Editor, a tool that

displays the VAP as a collection of functional boxes and interconnections.

Variable

An identifier that represents a value that can change.

VFX series

A series of Dialogic cards that have an added fax daughterboard.

Voice message

A class of message containing a digitized audio signal representing human speech.

Voicemail

The service provided by a computer telephony system for processing its users' incoming calls. Smartphone provides a full range of Voicemail functions including message recording, forwarding, notification of new messages, etc.

VOX

Dialogic Voice, a file format for saving digitized voice data. Vox-format used by Smartphone Server 3.1: 8KHz sampling rate, 4bit resolution with ADPCM compression.

W

WAV

Windows waveform, a file format for saving digitized audio data, standard for Windows NT/2000/XP. Wav-format used by Smartphone Server 3.1: any PCM-encoded .WAV file..

Waveform

A visual representation (typically in graphical format) of the value of a physical measurement with time. A typical speech waveform displays the combined amplitude of all voice frequencies on the y-axis and time (in msec) on the X-axis.

X

Y

Z

Index

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Smartphone 3.1. Developer Guide.
Filename: devel.pdf
Edition: 3.120
Date: 10/4/2002

Special Symbols

Depending on your license and what hardware and software is available, some Smartphone features become enabled. The descriptions of these features in Smartphone documentation are marked with the special symbols shown in the table below. You may skip the sections about features which are not available in your case.

Table 1: Special features symbols

	Feature	Requirements
	Signaling	PBX and Smartphone Server should be properly configured
	Network connection	LAN and Smartphone Server should be properly configured
	Microsoft Exchange Server	Microsoft Exchange Server is installed and available
	SMS notification	GSM modem is installed and “SMS notification” option is licensed
	Email notification	Microsoft Exchange Server is installed and available and “Email notification” option is licensed
	Faxmail	Fax hardware is installed and “Faxmail” is licensed
	Smart-4-Fax Server	Fax hardware is installed and “Smart-4-Fax Server” is licensed

Smartphone 3.1

NOVAVOX AG
WWW:http://www.novavox.com/