

SAC Engine Control

Version 3.1

User's Guide

Issue 1.6

March 2007

Windows 9x / Me / NT / 2K / XP

Purpose

The purpose of this document is to explain how to use the SAC Engine Control. It describes the technical design, how to use the control, its functionality, the properties, methods and events, plus general information about the use of the software.

www.ScreenKeys.com



SAC Engine Control

Information in this document is subject to change without notice.

The latest revisions of the ScreenKey documentation and software are placed on the ScreenKeys Web site.

Web: www.ScreenKeys.com

Technical Support is available

via Email: support@ScreenKeys.com

via Web: www.ScreenKeys.com

© 2006 SK Interfaces Ltd.

All rights reserved.

DISCLAIMER:

SKI reserves the right to revise program interfaces, data file formats and functionality, at any time.

Foreword

The SAC Engine Control is a Windows COM control that manages a ScreenKey console from the information residing in a SAC file. The application can be written in any COM-enabled programming language, i.e. using a development environment utilizing the qualities of COM (ActiveX) controls. It is designed to work with languages like MS Visual Basic and MS Visual C++, using MS Visual Studio 6.0.

This document describes how to use the SAC Engine Control from the most popular programming languages.

Previous releases of this document included information on the wrapper module for non-COM enabled programming languages. This information is now available in a separate document (*SAC Engine Wrapper User's Guide.pdf*).

SKI – ScreenKey Interface

ScreenKey technology has been in use for a considerable number of years. In this time, SKI has developed a range of software tools to simplify the task of integrating a ScreenKey console into an application or hardware solution.

SKI Software Toolset (SKI) implements a new approach to controlling ScreenKey consoles and keyboards. SKI is a toolset designed specifically for Windows 32-bit platforms and utilizes Microsoft COM technology.

SKI is particularly targeted at Application Developers, offering a host of developer tools to allow tight and dynamic integration of ScreenKey™ technology.

The SAC Engine interacts with a ScreenKey™ console according to a pre-designed SAC file but also offers the developer the ability to trap Windows events and change individual ScreenKeys, jump to a different SAC menu, display application specific data, etc.

When the end-user application cannot be modified to incorporate the SAC Engine, the SAC Controller may be employed. This provides a mechanism to statically feed keystrokes to an application based on a pre-designed SAC file. *Note that the SAC Controller does **not** control the large LCD display integrated into the SK-7000 console.*

Alternatively, developers may write a user-specific application (using the SAC Engine) to provide a highly customized SAC Controller equivalent. SKI publishes sample source code for this opportunity.

Table of Contents

SAC ENGINE CONTROL OVERVIEW	7
<i>What is a ScreenKey Console ?</i>	7
<i>What is SAC ?</i>	7
<i>What is SAC Engine Control ?</i>	7
<i>What is SKI ActiveX Control ?</i>	8
<i>Who should read this document ?</i>	8
<i>Requirements</i>	8
<i>Cross reference</i>	8
INSTALLATION PROCEDURE.....	9
Hardware Installation	9
Software Installation.....	9
Uninstallation of Software.....	10
Redistributable Components / Files.....	12
<i>Downloadable Console Firmware.....</i>	13
<i>Console Interface (Handshake) Specification</i>	14
<i>SAC File Icons.....</i>	14
<i>DCOM & Windows 95.....</i>	14
<i>PSAPI.DLL.....</i>	15
TECHNICAL DESIGN.....	16
SAC Engine Control.....	17
Direct SAC Engine Control Access.....	18
SAC Engine using PlayBack	19
Key Numbering	20
<i>The SK-2000 Console</i>	20
<i>The OEM-5400 Console</i>	21
<i>The SK-6000 Keyboard</i>	21
<i>The SK-7000 Console</i>	22
USAGE.....	23
SAC Engine Modes	24
SAC Engine Advantages	25
SAC Engine in COM-Mode	25
SAC Engine in PlayBack-Mode	26
SAC Engine in combined mode (COM & PlayBack)	26
SAC Engine as standalone application – SAC Controller	27
Altering ScreenKeys dynamically	28
Advantages of the PlayBack mode	32
PlayBackMode and PlayBack methods	33
<i>Example of application using the PlayBack method.....</i>	35
Security levels	36
Target application – window tracking	38
The EventStatus mask	41
Methods, Properties & Events	42
Categories	43

METHODS	46
StartSAC	46
StopSAC	48
GotoMenu	49
UpdateKey	50
PlayBackMode.....	51
PlayBack	53
SetSecurityLevel.....	53
SetKeyAttribute	55
SetWindow.....	56
TrackWindow	57
WriteTextODA	58
WriteTextTDA.....	59
WriteTextCDA.....	60
SelectCDA	61
SelectTDA	61
WriteGraphicTDA	62
 PROPERTIES	 63
KeyCodes.....	63
KeyNum.....	64
KeyLockPos.....	64
MSRTracks	65
MSRTrack.....	65
CurrentMenu	66
CurrentWindow.....	66
MenuName.....	66
GraphicName	67
EventStatus	67
ErrorNumber	68
ErrorDescription	68
LLErrorNumber	68
LLErrorDescription.....	69
 EVENTS.....	 70
KeyStroke	70
KeyLockTurn.....	70
MSRSwipe	71
MenuChange	71
TrackChange	72
SACError	72
 ERROR HANDLING	 73
Error numbers and messages.....	74
 TROUBLESHOOTING.....	 77
SAC Engine Errors	77
 LANGUAGE INTERFACES	 89
Visual Basic	89

Visual C++	91
Using MFC	92
<i>UpdateKey method</i>	92
<i>KeyCodes property</i>	93
Delphi 5	94
<i>Installation in Delphi</i>	94
<i>Using Control in project</i>	94
<i>Event handling</i>	95
<i>Error handling</i>	95

APPENDICES

DOCUMENTATION CONTROL..... 96

A.1	Change Control	96
A.2	Abbreviations Used/Terms of Reference	96
A.3	Historical Change Reference.....	97
A.4	Change Summary	97

SAC Engine Control Overview

What is a ScreenKey Console ?

The ScreenKey console is an Input/Output device for use with Point-of-Sale (POS) and other specialized applications, such as process control, dealer desks, call centers etc. The ScreenKey console combines the best features of a standard keyboard and a TouchScreen. It is made up of conventional keys and special ScreenKeys. A ScreenKey acts like a conventional key but each ScreenKey has a built-in LCD display panel. This enables the ScreenKey's legend to be changed dynamically. ScreenKeys are available in several configurations:

- Resolution: LC16 (32x16 pixels) or LC24 (36x24 pixels)
- Backlight Colors: RG (red-green) or RGB (red-green-blue) LED backlighting

What is SAC ?

SAC is short for ScreenKey Active Control. SAC files are the cornerstones of the ScreenKey Windows developer toolset. The SAC system facilitates quick and easy integration of ScreenKey technology into Windows applications, with minimal or no application modification. SAC is a self-contained data-file containing a set of drill-down menus for the ScreenKeys, the relationship between these menus, and other applicable rules and data. SAC files are generated and maintained using a special purpose Windows file editor, the SAC Editor. Each ScreenKey, in each menu, has properties such as a caption (text or graphics) to display on the key's LCD display, background and flashing colors, a menu to display when the key is pressed, a security level etc. In addition, each key can have one or more keystrokes assigned to it, which are sent to the application when the key is pressed. Other devices, such as the MSR unit and KeyLock, also have properties attached to them.

The SAC system is a software system driven by the keyboard itself on the basis of the contents of the SAC file, producing keystrokes for the application(s).

What is SAC Engine Control ?

The SAC Engine Control is a software package that forms the interface between the ScreenKey keyboard/console and Windows applications. It is a COM (ActiveX) module, and can be used by modern programming languages like Visual C++, Visual Basic etc., that support COM technology.

The Control can be integrated fully into an application, using the properties, methods and events to exchange information with the ScreenKey console. It can be integrated with a minimum number of changes, feeding the keystrokes back via the keyboard message system. It can also be used as a separate side-application, sending the keystrokes to the currently active window.

What is SKI ActiveX Control ?

SKI ActiveX Control is a software package that allows the ScreenKey console to be easily integrated into Windows based applications. It can be used by modern programming languages, e.g. Visual C++, Visual Basic, etc that support COM technology. SKI ActiveX Control does not work with SAC files but offers control of individual ScreenKeys and their properties.

Who should read this document ?

This User's Guide is intended for the person who will interface the ScreenKey console to an application, using the SAC approach. The programmer must be familiar with ActiveX controls in general, the programming language, the development tools, the operating system, and know how the ScreenKey console works.

This guide assumes that the reader is familiar with all of the above, Microsoft Windows and Windows commands.

Users should have read the SAC Editor User's Guide.

Requirements

Most applications should be suitable for use with SAC Engine and SKI ActiveX Controls as long as they meet the following requirements...

Operating System:	Windows 9x, Me, NT, 2K, or XP
PC Hardware:	The application must run on standard PC hardware. One COM port must be available.
Available PC Resources:	The PC must have sufficient available resources to run the ScreenKey console software. The SAC Engine Control and the SKI ActiveX Control can run on a PC with the amount of memory required for running the operating system.

Cross reference

- [1] ScreenKey Getting Started & Installation Guide
- [2] SAC Editor User's Guide
- [3] SAC Controller User's Guide
- [4] SKI ActiveX Control User's Guide
(formerly ScreenKey ActiveX Control User's Guide)
- [5] ScreenKey Console Technical Reference Manual
- [6] ScreenKey Low-Level Interface Programmer's Guide
- [7] SAC Engine Wrapper User's Guide

Installation Procedure

Hardware Installation

See the **ScreenKey Console, Getting Started & Installation Guide** document for details of hardware installation.

Software Installation

The SAC Engine Control is supplied on CDROM as part of the SKI Software Toolset. Alternatively it may be downloaded as an individual component from the web (www.ScreenKeys.com).

When distributed by CDROM, the CD usually incorporates an autorun facility that launches an installation helper utility (e.g. HTML file). Follow the on-screen instructions as described by this utility. If the utility is not present or the autorun feature is disabled on your computer, you can install the software directly from the CD. The ScreenKey software package typically includes several software applications that reside separately on the CD. Find and open the SAC Engine sub-folder using Windows explorer from the root of the CD and run Setup.exe from this folder.

The SKI ActiveX Control, which is used by the SAC Engine Control, is installed automatically.

In all cases the installation can be started using the **Setup.exe** program, then follow the on-screen instructions.

- After the Welcome dialog, the Software License Agreement has to be accepted before the installation can proceed.
- During the installation, the user is asked if the computer should be scanned for old PWD installs. This procedure trawls the users hard-disk(s) and can take several minutes. Although not essential, it is advised that old PWD installs should be removed prior to installing this software. It is safe to bypass this search.
- The default destination location for the software is **C:\Program Files\ScreenKey**. Click on the Browse button to select a new destination folder. This only applies if ScreenKey console software has not been installed onto the computer previously. If it has, this folder will be selected automatically, and the user will be informed with a dialog. The existing ScreenKey software must be uninstalled to be able to install to a different destination. The installation will build a folder structure from the destination location for the different parts of the software. The control will be put in .\Bin, the documentation in .\Doc, and the samples in .\Samples, etc.

The user may specify which components to install individually:

SAC Engine Development

The Control itself with all interfaces, also the SKI ActiveX Control.

SAC Engine Control Samples Sample programs for SAC Engine development.

ScreenKey ActiveX Control Samples Sample programs for the ScreenKey ActiveX Control.

The first component is mandatory while the sample programs can be deselected.

The installation program will now display a list over the chosen components before the actual installation starts.

- When the control has been installed, a number of files have to be registered in the registry. This is done automatically but if errors occurs the user will be notified, and the Control will not function correctly. In this case, try the following:

1. Uninstall earlier versions, and reinstall the control.
2. Reboot the PC, stop all applications and install again.
3. Log on as Administrator, and install again.
4. View the Windows Event Log, correct any errors and install again.
5. Contact support@ScreenKeys.com

If the control fails to register, a separate batch file is provided (Register.bat), which can be run from a DOS shell when the errors have been corrected.

- After registration of the SKI ActiveX Control, the user is requested to specify the type of interface that will be used communicate with the ScreenKey console. This primarily refers to the type of interface cable being used (see below for further information). *Note: All SK Interfaces product releases use a standard RTS/CTS crossover serial cable.*

Uninstallation of Software

To uninstall the SAC Engine Control:

1. Terminate any application using the SAC Engine Control, and the SKI ActiveX Control, and make sure no other application is using any of the other files, like documentation and sample code.
2. Start **Add/Remove Programs**, from the **Control Panel**.
3. Locate the line **SAC Engine Control** in the list of installed programs.
4. Click the **Add/Remove** button.
5. Click **Yes** in the dialog if you want to completely remove the SAC Engine Control an all of its components.
6. The SAC Engine Control with all of its components, documentation and sample programs will then be removed, all folders will be deleted if they are empty, and all registry entries will be removed. Since the control files may be shared, the uninstall program might ask the user if these shall be deleted, and that will normally be ok.

7. If parts of the control could not be removed, a button named Details... will be visible. Click on this to see what was not successfully removed.
Any changed files, like sample code etc., will be left, and if the control was in use by an application it will not be possible to remove this.
8. The software should now have been removed, and **Add/Remove Programs** can be terminated.

Redistributable Components / Files

When using the SAC Engine Control in an application some files have to be redistributed. They should all reside in the same folder to avoid problems. The application may be distributed with the following files:

File name	Description	Registering
SACEngin.exe	SAC Engine Control. This file is the control itself, and is ALWAYS necessary.	YES
SKPlayBack.dll	ScreenKey PlayBack DLL. This file is used by the SAC Engine Control when in PLAYBACK mode. It is a necessary part of the control only if in this mode, but will always be installed.	NO 1)
Psapi.dll	This is a Microsoft DLL. It is used by the SAC Engine Control when the TrackWindow method is used. It is only necessary for Windows NT and 2000. When running on Windows 95/98 or ME, Windows built in functionality is used instead. This DLL is a part of the Windows SDK and can be freely distributed.	NO 1)
SACEnginps.dll	Proxy Stub for SAC Engine Control.	YES
SACEngIf.dll	SAC Engine Control Interface. This file is necessary if the control is used from e.g. Visual Basic, or Visual C++ with the control running in the applications process space.	YES
pkficon.ico pkbicon.ico	SAC Icon files. See below.	YES

The SAC Engine requires the SKI ActiveX Control to be installed too. The following files are required for redistribution.

File name	Description	Registering
SkAxCtl.exe	SKI ActiveX Control. This file is the control itself, and is ALWAYS necessary.	YES
SkAxCtlps.dll	Proxy Stub for SKI ActiveX Control.	YES
SkAxCtl.dll	This file is a re-distributable file that comes with the SKI ActiveX Control, but is not used by the SAC Engine Control.	YES
*.skf	ScreenKey Font files. These files are distributed for CodePages other than 1252. They should be distributed with the control, residing in the same folder as SkAxCtl.exe.	N/A

- 1) These files are installed to the Windows System folder (WINNT/SYSTEM32 or WINDOWS/SYSTEM)

Some of the EXE and DLL files have to be registered in the registry. SACEngin.exe and SkAxCtl.exe can do the registering by themselves, but the DLL files need to be registered with the program RegSvr32.exe. RegSvr32.exe is distributed from Microsoft Corp., and can be used freely. It can be found in the Windows System folder (WinNt\System32 or Windows\System).

To register the components one has to execute the following commands, in the following order:

```
RegSvr32 SkAxCtlPS.Dll
SkAxCtl.exe /RegServer
RegSvr32 SkAxCtl.Dll
RegSvr32 SACEnginPS.Dll
SACEngin.exe /RegServer
RegSvr32 SACEngIf.dll
```

This can be done by executing a batch file, or by including the commands in an installation program.

Be aware that if both files, Regsvr32.exe and the DLLs, are not in the current working folder or the RegSvr.exe program is not in the standard path, it might be necessary to specify a path for the either one or both files.

SkAxCtl.dll is installed with the SAC Engine Control. This is not needed by the control, but is installed in case other applications uses it. It may be removed, but SkAxCtl.dll should then be unregistered first.

Note that the components can be installed in any suitable directory, as long as all supplementary files are installed together with EXE-files, and that they are correctly registered.

Downloadable Console Firmware

The ScreenKey console supports downloadable firmware as a means of adding new functionality and/or correcting software bugs. SKI will issue new firmware from time to time.

The path and name of this firmware is defined in the Windows registry. The installation of the SAC Engine automatically creates the following registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\ScreenKeys\ScreenKey ActiveX  
Control\ROMFullPath
```

Set the value of this key to point to a new firmware download file, e.g.:

“C:\Program Files\ScreenKey\Bin\SK54_52.bin” for products such as the OEM-5400

or

“C:\Program Files\ScreenKey\Bin\SKC1_0.bin” for SK-7000 products

Console Interface (Handshake) Specification

During the installation, the user is requested to specify the type of interface to be used to communicate with the ScreenKey console. The user's response is stored in the registry under the following key:

HKEY_LOCAL_MACHINE\SOFTWARE\ScreenKeys\ScreenKey ActiveX Control\Handshake

Set the value of the key to one of the values in the below table:

Value	Card reader type
0	Autodetect interface
1	Use RTS/CTS (standard RS232 handshaking)
2	Use old RTI interface protocol (auto-detect 'blip' or 'no-blip' and use hardware reset)

Although it is tempting to set this value to 0 (auto-detect) this setting considerably slows down the startup procedure for establishing communications with the console. Auto-detection can take up to 10 seconds to establish a connection.

SAC File Icons

To define the SAC file icons, the icon files have to be installed to the hard disk and the following keys must be defined in the registry:

HKEY_CLASSES_ROOT\.pkf
(default) = pkffile

HKEY_CLASSES_ROOT\.pkb
(default) = pkbfile

HKEY_CLASSES_ROOT\pkffile
(default) = "SAC File"
DefaultIcon = "<full path>\pkficon.ico

HKEY_CLASSES_ROOT\pkbfile
(default) = "SAC Backup File"
DefaultIcon = "<full path>\pkbicon.ico

DCOM & Windows 95

The controls use COM to communicate with each other. COM is built in to all Windows operating systems, but there is a known error in COM with Windows 95. Before the controls can be installed on a Windows 95, DCOM95 has to be updated. This installation is available, free of charge, from Microsoft Corp., and is also available on the SAC Engine Installation CD.

PSAPI.DLL

The dynamic link library psapi.dll contains functionality for retrieving process information in Windows NT and 2000. When installing to these operating systems, this dll must be present. If an existing version of this file is older than the one distributed with the SAC Engine Control, this must be upgraded.

Windows 95, 98 and ME have this functionality built in to the operating system, and the file is therefore not needed on these systems.

Technical Design

The SAC Engine Control is designed for use with different programming languages, and is split into two main modules:

- SAC Engine Control
- SAC Engine Control Interface

The **SAC Engine Control** itself runs in a separate process space, and can therefore not be accessed directly from Visual Basic. The SAC Engine Control is the module that does the SAC processing, the other modules are just interfaces to this. It is accessible from C++ when it is created as an object. The control is an EXE file, called **SACEngin.exe**.

The **SAC Engine Control Interface** runs in the process space of the application hosting the SAC Engine, and is therefore accessible from Visual Basic. This is a DLL file, called **SACEngIf.dll**.

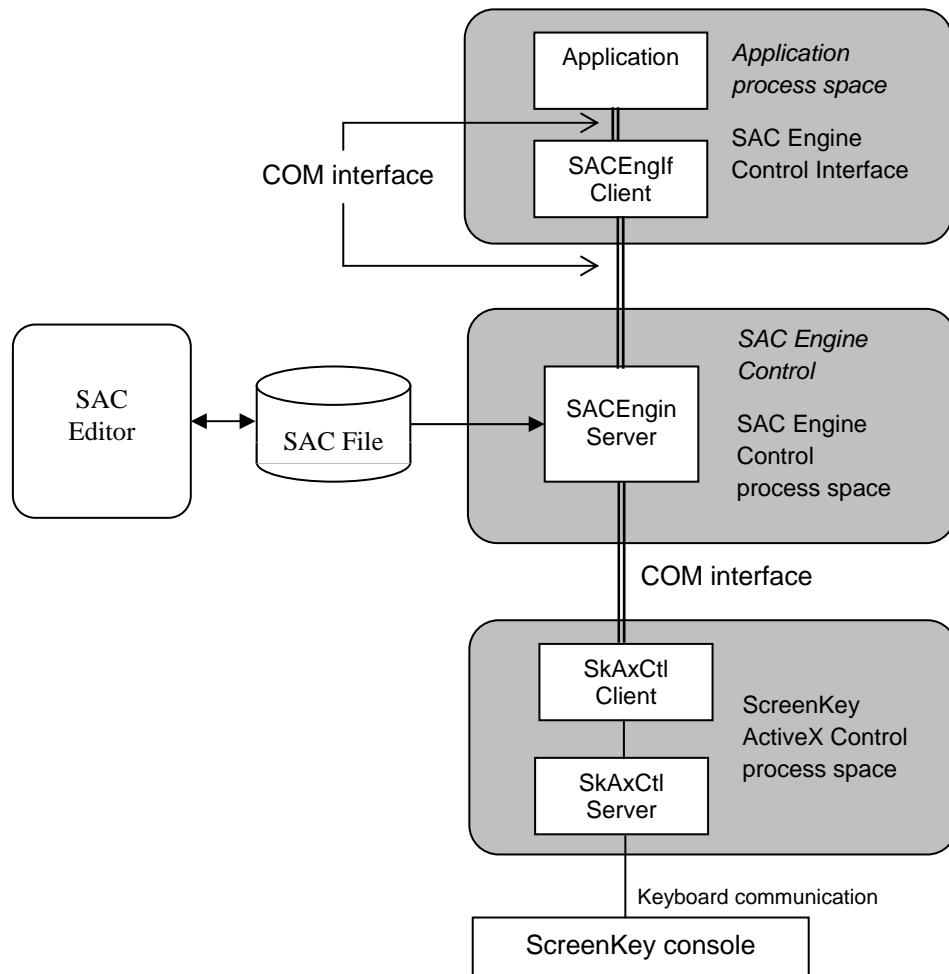
A Proxy Stub DLL is also necessary when using the SAC Engine Control; this is called **SACEnginps.dll**. Since this is necessary in all cases, when using the SAC Engine Control directly, or when used via the SAC Engine Control Interface, it must always be installed and registered.

SAC Engine Control

The SAC Engine Control is an EXE server, which means it runs in its own process space as a separate application. This has a few advantages, one being that it is safer because the Engine cannot crash the hosting application, another being that it will be able to process even if the hosting application is busy doing other tasks.

The SAC Engine uses the SKI ActiveX Control as a low-level control for communicating with the ScreenKey console. The SKI ActiveX Control Interface is NOT used, since the SAC Engine is written in C++, a language that can connect to an out-of-process-space COM control directly.

Depending on the application hosting the SAC Engine, the SAC Engine Control Interface may be used, as shown in the figure below, or the application can use the SAC Engine Control directly e.g. if used from C++.



**Figure 1 – SAC Engine Control, with SAC Engine Control Interface.
E.g. Visual Basic**

Direct SAC Engine Control Access

As mentioned above, the SAC Engine Control (the EXE file) can be accessed directly from languages like C++.

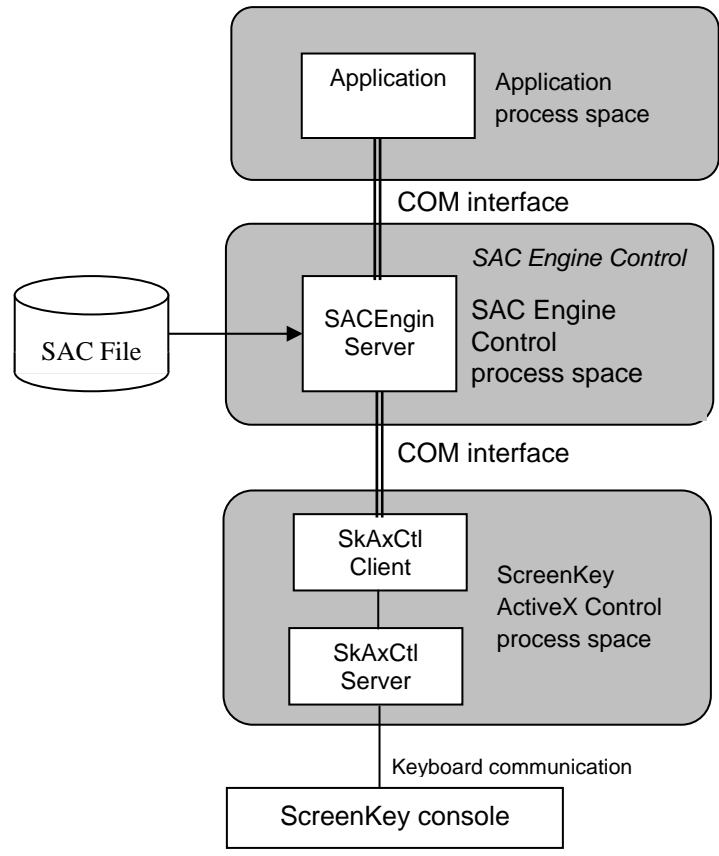


Figure 2 – Direct SAC Engine Control Access. E.g from C++

SAC Engine using PlayBack

The SAC Engine Control can be used as an ordinary COM control, but can also work with other applications using the PlayBack function. PlayBack means the target application does not have to be changed, and will be operated by the ScreenKey console sending keystroke messages directly to the application, using a hook and the JournalPlayback functionality in Windows. The control still needs an application to host it, but all output is fed to the target application using the above mentioned playback function, SAC_PLAYBACK_MODE instead of SAC_COM_MODE. The **SAC Controller** is such a product, and is available as a separate product. See the download area at www.ScreenKeys.com.

A combination of the two modes can also be used, and can be very powerful.

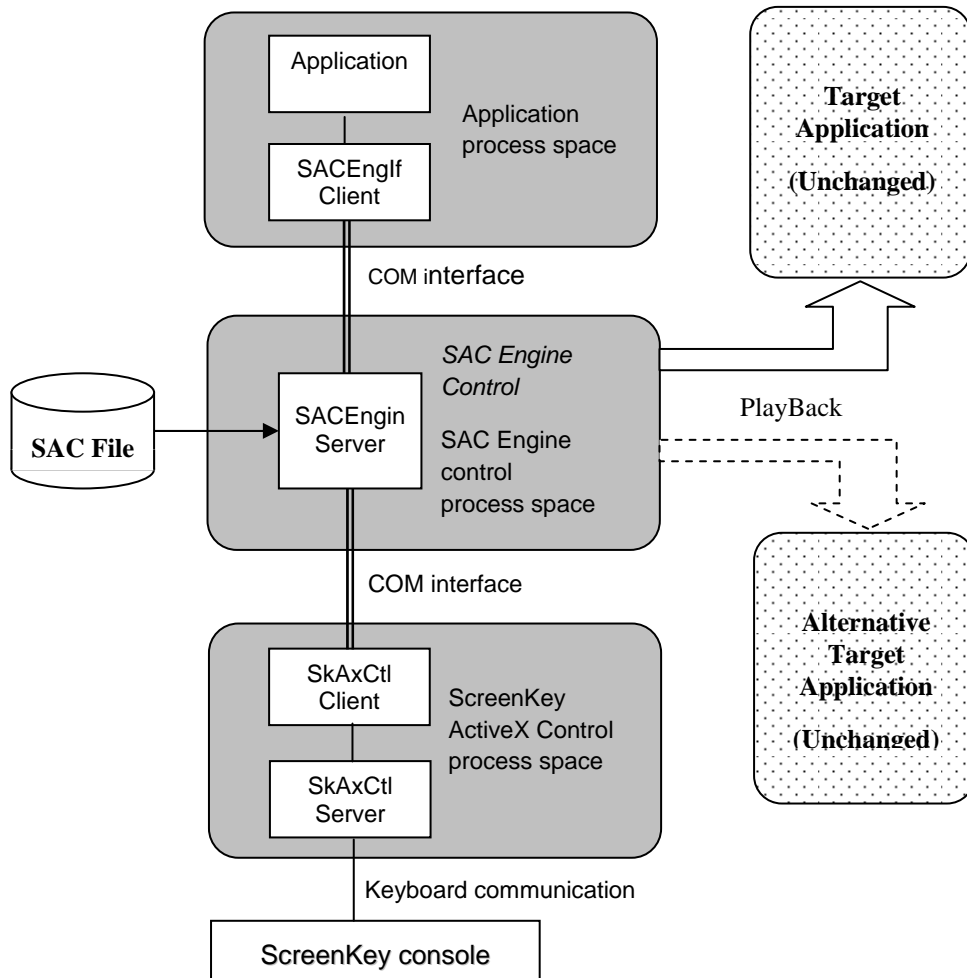
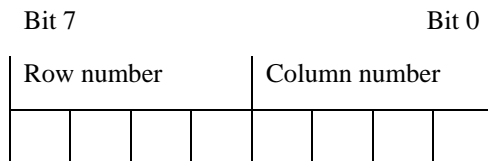


Figure 3 – Application hosting SAC Engine Control, using SAC Engine Control Interface, playback to one or more unchanged applications.

Key Numbering

To use certain methods in the SAC Engine Control, individual keys may have to be addressed. Each key has a key number, defined by the SKI ActiveX Control. The numbering system used is a simple numbering system, counting row and columns from the upper left corner of the console.

The keys are addressed as Row and Column, but key numbers reported from the SAC Engine Control combines these two numbers into one number. The system is based on a maximum of 16 keys per row, and 8 rows. The Column number forms the lower four bits of a byte, and the row number the upper four bits as follows:



The rows and columns are counted from the upper left key, where this keys has Row = 0 and Column = 0. The key numbers will therefore vary from keyboard type to keyboard type, and from keyboard layout to keyboard layout.

The SK-2000 Console

This console has been discontinued but the information is contained here for legacy users and for general information.

This example is for a SK-2012 keyboard with the 12 ScreenKeys on the right. The keys are numbered as rows and columns, from top left to bottom right as follows:

Fixed keys								ScreenKeys		
R 0	R 0	R 0	R 0	R 0	R 0	R 0	R 0	R 0	R 0	R 0
C 0	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10
R 1	R 1	R 1	R 1	R 1	R 1	R 1	R 1	R 1	R 1	R 1
C 0	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10
R 2	R 2	R 2	R 2	R 2	R 2	R 2	R 2	R 2	R 2	R 2
C 0	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10
R 3	R 3	R 3	R 3	R 3	R 3	R 3	R 3	R 3	R 3	R 3
C 0	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10
R 4	R 4	R 4	R 4	R 4	R 4	R 4	R 4	R 3	R 3	R 3
C 0	C 1	C 2	C 3	C 4	C 5	C 6	C 7	C 8	C 9	C 10

This key will be reported as key number 2A hex

Note that since the fixed keys and the ScreenKeys don't align, row 4 of the fixed keys is by side of row 3 of the ScreenKeys. There are no missing rows.

The OEM-5400 Console

This example is for a OEM-5400 console, which only has 12 ScreenKeys and no fixed keys. The keys are numbered as rows and columns, from top left to bottom right as follows:

R 0	R 0	R 0
C 0	C 1	C 2
R 1	R 1	R 1
C 0	C 1	C 2
R 2	R 2	R 2
C 0	C 1	C 2
R 3	R 3	R 3
C 0	C 1	C 2

This key will be reported as key number 12 hex

The SK-6000 Keyboard

This console has been discontinued but the information is contained here for legacy users and for general information.

This example is for a SK-6000 keyboard, which has 2 panels of 12 ScreenKeys on the left and one panel of 12 fixed keys on the right. The keys are numbered as rows and columns, from top left to bottom right as follows:

R 0	R 0	R 0	R 0	R 0	R 0
C 0	C 1	C 2	C 3	C 4	C 5
R 1	R 1	R 1	R 1	R 1	R 1
C 0	C 1	C 2	C 3	C 4	C 5

R 0	R 0	R 0	R 0	R 0	R 0
C 6	C 7	C 8	C 9	C 10	C 11
R 1	R 1	R 1	R 1	R 1	R 1
C 6	C 7	C 8	C 9	C 10	C 11

R 0	R 0	R 0	R 0
C 12	C 13	C 14	C 15
R 1	R 1	R 1	R 1
C 12	C 13	C 14	C 15
R 2	R 1	R 1	R 1
C 12	C 13	C 14	C 15

This key will be reported as key number 1C hex

The SK-7000 Console

The SK-7000 has 12 ScreenKeys and 37 fixed keys in the order shown below. The keys are numbered as rows and columns, from top left to bottom right as follows:

ScreenKeys			Fixed keys										
R 0 C 0	R 0 C 1	R 0 C 2	R 0 C 3	R 0 C 4	R 0 C 5	R 0 C 6	R 0 C 7						
R 1 C 1	R 1 C 1	R 1 C 2	R 1 C 3	R 1 C 4	R 1 C 5	R 1 C 6	R 1 C 7	R 1 C 8	R 1 C 9	R 1 C 10			
R 2 C 2	R 2 C 1	R 2 C 2	R 2 C 3	R 2 C 4	R 2 C 5	R 2 C 6	R 2 C 7	R 2 C 8	R 2 C 9	R 2 C 10			
R 3 C 3	R 3 C 1	R 3 C 2	R 3 C 3	R 3 C 4	R 3 C 5	R 3 C 6	R 3 C 7	R 3 C 8	R 3 C 9	R 3 C 10			
			R 4 C 3	R 4 C 4	R 4 C 5	R 4 C 6	R 4 C 7	R 4 C 8	R 4 C 9	R 4 C 10			

This key will be reported as key number 43 hex

U S A G E

Usage

There are different ways of using the SAC Engine Control. The basis of the software is an out-of-process COM control, and it has to be considered as a library type of software. Playback mode offers a quick and simple method for integrating ScreenKey console control. Using the COM interface provides a mechanism to provide greater functionality and better performance from the ScreenKey console.

It is easy to integrate it into a separate application for Playback mode. It only takes two calls to the control, a call to StartSAC and a call to StopSAC, to make it a standalone application. It will then work in PlayBack mode only, but will be a fully functional application, controlling the ScreenKey console from the information in the SAC file, and sending keystrokes to the currently active application. This solution exists as a product named **SAC Controller**, and is available from the download area at www.ScreenKeys.com.

It is important to bear in mind that the control is mostly driven from the ScreenKey console, and therefore is not dependent on being driven from an application. Added value can be achieved by adding functionality to the application by using the consoles potential, by controlling it partly from the application. In this case one has to integrate the SAC Engine Control more or less tightly into the application, or a separate side-application can be designed, which again can communicate with the main application.

The SAC framework provides a comprehensive and integrated approach to incorporating ScreenKey technology into a target application. The SAC Editor allows the design of a ScreenKey user interface with actions (keystrokes) offline from the runtime system. The SAC file is self-contained and holds the different ScreenKey menus (ScreenKey text, graphics and colours), the relationships between the menus, and keystrokes defined for each key. The SAC Engine Control provides a Windows server that processes this SAC file. The application can leave all the processing to the SAC engine or take control of the keyboard at any point, i.e. intercept the SAC processing and modify any actions to be taken.

*The SAC framework does **not** include support for the large LCD display integrated with the SK-7000 ScreenKey console. The SAC Engine provides dedicated methods for handling this component directly. The SAC Controller does **not** support this display.*

SAC Engine Modes

There are five main ways to use the SAC Engine Control:

1. As a COM control where the methods are used to control the console, the properties are used to retrieve data from the keyboard, and the events to inform that actions have taken place. The SAC Engine will still process the console events according to the instructions in the SAC file, so the application does not have to do that part. This can be called 'pure COM' mode.
2. As a COM module where the only interaction with the control is to Start the system, and Stop it. The keystrokes produced will then be sent to the target application using the Journal Playback system that Windows offers. The SAC Engine pretends to behave like the QWERTY keyboard, inserting keyboard messages into the application's message queue. This can be called 'pure PlayBack' mode.
3. As a combination of the two approaches above. This can be a very powerful solution for existing programs, where functionality can be added using the events and methods of the SAC Engine Control. This would be called 'combined' mode.
4. As a separate side-application. With a tiny program to start and stop the control, it can be turned into a self-contained program feeding keystrokes into the active application. This can be useful when the source code to the application is not available, or one doesn't want to change the application. This can also be called 'pure PlayBack' mode. The **SAC Controller** is such a product and is available from SKI as a separate product.
5. As a side-application with special features built into it. The side application would be a program dedicated to control the ScreenKey console, but it would probably also have an interface to the main application. This approach is similar to approach 3 but with a special interface to the main application. Such a side application would probably use the control in a 'combined' mode.

These approaches will be discussed in detail later.

The SAC Engine Control can be used from different programming languages as described above under Technical Design, page 16. It can be used from:

- C++ and other programming languages supporting out-of-process COM controls, using the out-of-process control directly, or by using the control interface as an insertable control.
- Visual Basic and other programming languages NOT supporting out-of-process COM controls, but supporting COM modules, by using the control interface as an insertable control.

SAC Engine Advantages

How to integrate a ScreenKey console into an application can be done in many ways, depending on the type of application, e.g. if the application is being designed with this in mind or not, if it is an old-fashioned application or a modern one, and so on.

The approach that gives the tightest control over the console is to use the direct control approach, i.e. the SKI ActiveX Control, which gives all control to the application, and access to all features of the console. To benefit from this can take a lot of programming, although the COM control makes this easier than if it was a standard library, and the application has to allow for such integration. The application has to contain the data to display on the ScreenKeys, all relationships between the different menus, and how to interpret keystrokes etc. This approach, the SKI ActiveX Control will be best if there are few menus or layers of menus, and if the data on the ScreenKeys is very dynamic.

If the user interface will require a menu hierarchy, like a drill down menu system, the SAC Engine Control can be a far better approach. This keeps the menu data out of the application, as if the menus and their corresponding commands/keystrokes are in a separate database. The SAC Engine Control will act as a device driver that also handles the data. The main advantage is that the control of the console can be kept away from the main application and only the necessary commands need be integrated into the application to add value to the application. The slogan for the SAC Engine will be “Maximum gain - minimum effort”.

SAC Engine in COM-Mode

The most obvious way of using the SAC Engine Control is to use it as an ordinary COM control. This gives the application the very best control over the ScreenKey console, without “bothering” the application with all the details that would have been necessary if the SKI ActiveX Control was used.

The application receives events from the SAC Engine Control only when the application needs to handle actions at the console. These can be keystrokes or strings assigned to the pressed key, KeyLock changes, MSR data, menu changes or errors that occurred. When a button with a link to another menu is pressed, the application does not have to act, even if it is notified through the MenuChange event. This way the interaction with the console can be kept to a minimum. It is possible for the application to be notified even if no keystrokes are assigned to the pressed key if desired. This can be useful if one wants to implement special functionality for some keys. The MenuChange event does not have to be handled but can be useful for synchronisation purposes, or if there is need to change the legend of a key. It could be used in situations where the application wants to change a mode, open a window etc. as a result of a new menu displayed on the ScreenKey console.

There are also methods available for controlling the console. A certain menu can be displayed on the ScreenKey console on demand. This can be useful if the application goes to a certain mode, e.g. a tender mode in a POS application. The console can also be ordered back to the previous menu after a diversion as the one mentioned above.

The application can change what a ScreenKey is to display. This will be done when a menu is about to be changed, and it will be in effect until another menu is displayed or the same one is

displayed again without being modified. This is an easy way of getting dynamic data on some ScreenKeys, especially if most of the menus/keys are static.

The application can also interrogate the SAC Engine about available menus or graphical images in the current loaded SAC file.

This mode is ideal for new applications, where the integration of the ScreenKey console support can be designed into the application, but also for well-designed existing applications.

SAC Engine in PlayBack-Mode

The PlayBack mode is an easier way of implementing the ScreenKey console into an application, but limits access to the optimum functionality that the COM interface can provide. Only two small changes have to be made to the application to integrate the ScreenKey console in the simplest implementation. The SAC Engine Control (which again initializes the ScreenKey console) has to be started and initialized, and it has to be stopped when the application is shut down.

The SAC Engine needs some information, like which mode to operate in, which comm-port to use, which SAC file to use, and a keystroke definition file is necessary. It may also be necessary to specify a window handle or application name for the application it is going to work with.

When the SAC Engine is successfully initialized, it operates as a stand-alone application, feeding keystrokes to the target application through its keyboard message channel, until it is stopped. The SAC Engine makes sure all keystrokes are sent to the right application, unless it is directed to feed to another application. This can be done using the SetWindow and TrackWindow methods, which are only available in PlayBack mode.

The SAC Engine can also be configured to feed the keystrokes to the currently active application if desired.

This mode is ideal for existing applications or old-fashioned applications, where it is not possible or practical to integrate the ScreenKey console tightly into the application.

SAC Engine in combined mode (COM & PlayBack)

A powerful and easy solution is when the COM-mode and the PlayBack-mode is combined. This gives you the advantages of the feedback from the control, necessary to be able to control the keyboard tightly and the easy way of feeding keystrokes into the application.

This mode is ideal for maximum gain with minimum effort. It is suitable for both old and new applications.

More detailed examples are given later in this document.

SAC Engine as standalone application – SAC Controller

The last solution is running the SAC Engine Control as a separate stand-alone application – the **SAC Controller**. It is actually not a COM-module anymore, but a separate application, using the COM control. The application will start the SAC file processing at startup, and stop it when the application closes. Any keystrokes from the keyboard, after being processed by the SAC Engine, will be fed to an application. This can be the currently active application if no specific application is defined, or a specifically defined application or a specific window in an application, if an application/window has been defined. The SAC Controller will keep track of the specified window, and will stop feeding keystrokes if the application/window is closed, and will start feeding the keystrokes (again) when the application is started (window opened).

This solution is ideal for applications that for some reason cannot be modified.

The SAC Controller is available as a separate product.

Altering ScreenKeys dynamically

The nature of the SAC Engine is that the contents of the menus are static. This normally suits the purpose, because the SAC approach wouldn't be chosen if mainly dynamic menus were required. It could though be an advantage to be able to change the properties of keys within a menu dynamically. With the SAC Engine the ScreenKeys can be updated or altered dynamically. The caption of the keys may be altered, as can the background and flashing color, and the display attributes.

Each key, both ScreenKeys and fixed keys, can also be set up to auto repeat when held pressed, and the security level for each key may be altered. This is done with the SetKeyAttribute method and does not have to be done from an event.

The method UpdateKey can be used to alter the properties of a given ScreenKey. After the keys are updated, the changes are lost, which means that the next time the same menu is displayed, it will be as it originally was designed.

When updating a key, one can decide to update some of the properties only, or update all. E.g. one can update the bottom line of a key if desired, or just change the color.

The method supports TextToGraphics, which makes it possible to "squeeze" more than five characters into each line without having to make up a graphical image first.

The syntax for the UpdateKey method is:

```
HRESULT UpdateKey(  short Row,
                   short Column,
                   BSTR Line1,
                   BSTR Line2,
                   BSTR Graphics,
                   SAC_KEY_COLORS KeyColor,
                   SAC_DISP_ATTRIBUTE DispAttrib )
```

The syntax is almost the same as for the DisplayText method in the SKI ActiveX Control, except for the way to specify that a certain property is not to be altered.

The **Row** and **Column** specification is counted from the upper left corner, starting at zero, including the fixed keys, as specified in the SKI ActiveX Control User's Guide.

Line1, Line2 and Graphics may be altered or left unchanged individually, which means the topmost line may be altered while the bottommost is kept as in the SAC file. To leave a line unchanged, use an empty string. To update a line with a blank string, a string containing at least one blank character must be specified. If the key already has a graphical image to display, this will be mapped with the new text(s), as with the DisplayTextOnGraphics method in the SKI ActiveX Control. To leave the graphical image unchanged (or not use a graphical image at all), use an empty string. To avoid mapping of graphics and text, the Graphics parameter has to be set to a string at least containing one space.

Line1 and **Line2** as mentioned above can contain ordinary text, and it can contain more than 5 characters per line. The method will always use the features of the SKI ActiveX Control when UpdateKey is used, and the default text adjustment will therefore be Centered horizontally and vertically, even if the SAC_DISP_ATTRIBUTE is set to be SAC_NO_ATTR_CHANGE.

Actually, if a key contains the two lines: "RED" and "PEAS", they will be displayed left adjusted because the SAC Editor does not have text adjustment attributes, and this is the default behavior of the console. If the MenuChange event is intercepted and the UpdateKey is invoked with all parameters set to default (Updatekey(Row, Col, "", "", "", SAC_NO_COL_CHANGE, SAC_NO_ATTR_CHANGE)), it will change the two lines to be horizontally centered. This can be used to adjust the default behavior to centered text.

NOTE:

Use "" (no space) as parameter NOT to alter a text string defined in the SAC file.

Use " " (one space) as parameter to erase text string defined in the SAC file.

Use "String" as parameter to replace the text string defined in the SAC file.

Graphics can be specified in different ways. It can use an array containing the bit pattern as specified in the SKI ActiveX Control User's Guide, or it can use a named image contained in the SAC file.

An example (LC16 resolution) of displaying a local graphical image from VB can be:

```
Private Sub SACEngine_MenuChange(MenuName As String)

Dim Image(63) As Byte      ` Set up a 64 byte array
For i = 0 To 63           ` Make up an image, consisting of
    Image (i) = &h33      ` vertical stripes - 2 pixels each
Next i
SACEngine.UpdateKey 0, 8, "", "", Image, SAC_NO_COL_CHANGE, SAC_NO_ATTR_CHANGE

End Sub
```

An image existing in the SAC file can also be used from the MenuChange event. The name of the image must then be specified with the UpdateKey method.

```
SACEngine.UpdateKey 0, 8, "", "", "UPARROW", SAC_NO_COL_CHANGE, . . .
```

The image named UPARROW, residing in the loaded SAC file will be displayed on the ScreenKey in row 0, column 8.

NOTE:

Use "" (no space) as parameter NOT to alter the image defined in the SAC file.

Use " " (one space) as parameter to erase the image defined in the SAC file.

Use Array name as parameter to replace the image defined in the SAC file.

Use "IMAGE" (name if image) as parameter to replace the image defined in the SAC file.

The **KeyColor** parameter specifies both the background color, and the flash-color, as in SKI ActiveX Control. If one wants to use the existing colors set the parameter to SAC_NO_COL_CHANGE. If another color is specified, both the background color and the flash-color are affected. To specify bright red background flashing with dark red, OR the enumeration values SAC_BRIGHT_RED and SAC_FLASH_DARK_RED together. The following color settings are available (depending on ScreenKey backlight type – see note below):

Definition	Val	Backlight	Description
SAC_NO_COL_CHANGE	FFFh	RG, RGB	Don't change the colors
SAC_NO_COLOR	00h	RG, RGB	No color – backlight off
SAC_DARK_RED	01h	RG, RGB	Background colors
SAC_BRIGHT_RED	02h	RG, RGB	“
SAC_DARK_GREEN	03h	RG, RGB	“
SAC_BRIGHT_GREEN	04h	RG, RGB	“
SAC_DARK_ORANGE	05h	RG, RGB	“
SAC_BRIGHT_ORANGE	06h	RG, RGB	“
SAC_GREENISH_ORANGE	07h	RG, RGB	“
SAC_REDDISH_ORANGE	08h	RG, RGB	“
SAC_DARKBLUE	09h	RGB only	“
SAC_BRIGHTBLUE	0Ah	RGB only	“
SAC_PINK	0Bh	RGB only	“
SAC_DARKPURPLE	0Ch	RGB only	“
SAC_BRIGHTPURPLE	0Dh	RGB only	“
SAC_TORQUOISE	0Eh	RGB only	“
SAC_WHITE	0Fh	RGB only	“
SAC_FLASH_DARK_RED	10h	RG, RGB	Flash colors
SAC_FLASH_BRIGHT_RED	20h	RG, RGB	“
SAC_FLASH_DARK_GREEN	30h	RG, RGB	“
SAC_FLASH_BRIGHT_GREEN	40h	RG, RGB	“
SAC_FLASH_DARK_ORANGE	50h	RG, RGB	“
SAC_FLASH_BRIGHT_ORANGE	60h	RG, RGB	“
SAC_FLASH_GREENISH_ORANGE	70h	RG, RGB	“
SAC_FLASH_REDDISH_ORANGE	80h	RG, RGB	“
SAC_FLASH_DARKBLUE	09h	RGB only	“
SAC_FLASH_BRIGHTBLUE	0Ah	RGB only	“
SAC_FLASH_PINK	0Bh	RGB only	“
SAC_FLASH_DARKPURPLE	0Ch	RGB only	“
SAC_FLASH_BRIGHTPURPLE	0Dh	RGB only	“
SAC_FLASH_TORQUOISE	0Eh	RGB only	“
SAC_FLASH_WHITE	0Fh	RGB only	“

NOTES:

The availability of colors listed above depend on the type of LED backlight available on the ScreenKey. Only the first eight colors (plus first eight flash colors) in the table are available when using RG ScreenKeys. All colors are available when using RGB ScreenKeys.

Use SAC_NO_COL_CHANGE or -1 as parameter NOT to alter the colors defined in the SAC file, any other will define a new color. If background color is defined, flash-color will also have to be set again if desired.

The **DispAttrib** parameter specifies how the text is to appear (see SKI ActiveX Control User's Guide). As for the colors, if one attribute is changed, all have to be changed. To leave the existing attribute use SAC_NO_ATTR_CHANGE. The following display attribute settings are available:

Definition	Val	Mutual Exclusive		Description
SAC_NO_ATTR_CHANGE	0000h			No justification or attribute
SAC_NO_JUSTIFY	0000h			No justification or attribute
SAC_HOR_LEFT	0001h	✓		Horizontal justification, LEFT Applies to both lines of text.
SAC_HOR_CENTRE	0002h	✓		Horizontal justification, CENTRE Applies to both lines of text
SAC_HOR_RIGHT	0004h	✓		Horizontal justification, RIGHT Applies to both lines of text
SAC_VERT_TOP	0008h		✓	Vertical justification, TOP Applies only if one line of text is defined. If only bottom line is defined, this will be displayed at the topmost line.
SAC_VERT_CENTRE	0010h		✓	Vertical justification, CENTRE Applies only if one line of text is defined. Any of the lines will be centered vertically.
SAC_VERT_BOTTOM	0020h		✓	Vertical justification, BOTTOM Applies only if one line of text is defined. If only topmost line is defined, this will be displayed at the bottom line.
SAC_CONDENSED	0040h			Display the text condensed, even if there is space enough to use largest font. Normally if there is five or less character on a line, the largest font will be chosen unless this attribute is set. The second largest font will be used, and any extra gap between the characters will be removed.

SAC_INVERT_TOP	0100h				Invert the topmost line
SAC_INVERT_BOTTOM	0200h				Invert the bottom line
SAC_INVERT_KEY	0300h				Invert the whole key
SAC_FLIP_HOR	0400h				Flip the image horizontally
SAC_FLIP_VER	0800h				Flip the image vertically

NOTE:

This method does not support scrolling text or graphics. It does however allow text to be mapped with existing graphics.

NOTE:

Use SAC_NO_ATTR_CHANGE or -1 as parameter NOT to alter the attribute defined in the SAC file, any other will define a new attribute. Exceptions are mentioned above.

Advantages of the PlayBack mode

With the PlayBack mode, the SAC Engine control can send keystrokes defined in the SAC file to the target application. The reason for using this approach is because it may be too difficult or too risky to implement the SAC Engine Control as a COM control. This method still allows additional features to be implemented using the SAC Engine Control. If no additional features are to be added, the SAC Controller can be used, but if additional control of the keyboard is needed, a special integration will be necessary.

There are two main approaches for doing this:

1. Adding functionality to the application using the SAC Engine Control, but leaving the existing modules alone.
2. Make a second application that can host the SAC Engine Control, add the new functionality to this one, and leave the target application completely alone.

In both cases the target application (the existing one) has to be fed keystrokes using the playback. The SAC Engine Control will in these cases have to be opened in combined mode, both COM_MODE and PLAYBACK_MODE.

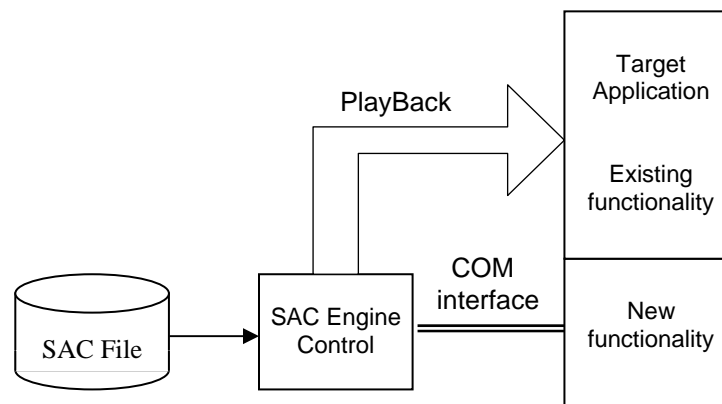


Figure 4 – SAC Engine Control hosted from NEW part of existing application, playback to existing part.

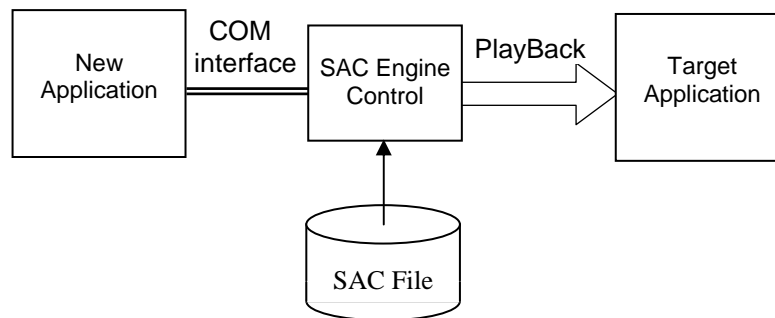


Figure 5 – SAC Engine Control hosted from NEW application, playback to existing, unchanged application.

PlayBackMode and PlayBack methods

With the PlayBackMode method, the application hosting the SAC Engine can enable and disable the keystroke playback in different ways. This is especially useful when some keystroke sequences defined in the SAC file, are not to be sent to the target application for some reason.

The PlayBackMode affects all types of playbacks, keystrokes, MSR swipes, etc.

The PlayBackMode can be set to the following:

- **SAC_PLAYBACK_OFF** Turns playback OFF immediately, and will not play back until it is set to ON again.
- **SAC_PLAYBACK_ON** Turns playback ON immediately, and will be ON until other mode is set.
- **SAC_PLAYBACK_NEXT_OFF** Turns playback OFF **after** the next series of keystrokes have been handled according to current mode, and from then on it will stay OFF until another mode is set.
- **SAC_PLAYBACK_NEXT_ON** Turns playback ON **after** the next series of keystrokes have been handled according to current mode, and from then on it will stay OFF until another mode is set.

- `SAC_PLAYBACK_ONE_OFF` Turns playback OFF for the next series of keystrokes to be played back, and then it will turn playback ON, independent of the current mode.
- `SAC_PLAYBACK_ONE_ON` Turns playback ON for the next series of keystrokes to be played back, and then it will turn playback back OFF, independent of the current mode.

This method may be used when the application enters a mode where keystroke sequences are not allowed, and the application wants to prevent them from being sent.

Note that the Security levels also can be used to prevent keys from being played back to the application, and this is normally a better way to control the keyboard.

PlayBackMode can also be combined with the PlayBack method to obtain certain features.

The Playback method is a method that can be used to send a sequence of keystrokes to the target application. It is supplied so the hosting application can send dynamic data, which the SAC Engine cannot since the keystroke sequences are stored statically in the SAC file. It can also be used for synchronization purposes.

The method has the following syntax:

PlayBack(BSTR KeyCodes)

KeyCodes is a BSTR, because this is an array of 16-bit values. The low byte (first byte) shall hold the ASCII value and the high byte (second byte) shall hold the ScanCode. These two bytes form what in the SAC Editor is called the **Scan Codes** - they are treated as a 16-bit Scan Code, including the ASCII value.

The available scancode, including the corresponding ASCII values can be found in the keyboard definition file, normally named `Kybgenuk.kbd`. The Scan Codes are “old-fashioned” DOS scancodes, and are needed to generate the messages used during playback.

For keystrokes only containing plain ASCII values, the scancode part can be omitted and the SAC Engine will fill these in from the information found in the `.kbd` file. An example of this is shown in the example below where the keystroke sequence is specified as a string. Since this is a BSTR string (Unicode) it is actually a 16-bit array, and therefore the second byte of every character is zero. Great caution has to be shown for special characters, like foreign characters, because these will have information in the second byte too.

For function keys, like F1, Ctrl-F10 and Alt-F, the key is defined as a scancode only and the ASCII value will then be zero. These keys cannot be described as above, the scancode part **MUST** be set, and the ASCII value part must be zero.

A keystroke like Shift-F or Ctrl-F, will have the same scancode as the character F, but with a different ASCII value, and these types of keystrokes have to be specified in full – both scancode and ASCII value.

Example:

The following example uses the PlayBackMode and the Playback methods to implement a macro-feature for the ScreenKey console. In the SAC file some keys are defined with the string `%1, %2` etc. as keystroke sequences. The percent sign means (in this case) that the key is to be treated as a

macro, and the digit is the macro identifier. When a key containing %1, the string “First macro” is to be played back to the target application, and if it contains %2, “Second macro” is to be played back.

This example sends plain strings to the application, which is not really correct. An array of scancodes and ASCII values should have been set up, but since there are no special characters the SAC Engine will fill in scancodes itself.

This playback of dynamic data can be implemented easily.

```
Private Sub SACEngine1_KeyStroke(ByVal NoKeys As Integer)
Dim KeyStrokes As Variant
Dim Keys() As Byte

If NoKeys Then      ` Key has keystrokes assigned to it
  ` Get the keystrokes and convert into a byte array
  KeyStrokes = SACEngine1.KeyCodes(NoKeys)
  Keys = KeyStrokes

  ` If it is a macro . . .
  If Keys(0) = Asc("%") Then
    ` Don't play back this sequence - skip one sequence
    SACEngine1.PlayBackMode SAC_PLAYBACK_ONE_OFF

    ` Which macro????
    Select Case Keys(2)
      Case Asc("1")
        SACEngine1.PlayBack "First macro"      ` PlayBack macro 1
      Case Asc("2")
        SACEngine1.PlayBack "Second macro"
    End Select
  Else
    ` . . .
  End If
End If

End Sub
```

Example of application using the PlayBack method

The ScreenKey console is being integrated with an existing POS application that cannot be changed. The obvious choice will be to choose the SAC Engine, since this does not require changes to the target application. Since the ScreenKey console has potential to do more than being an input device for the POS application, more functionality is wanted.

A new application can be made, hosting the SAC Engine, playing back keystrokes to the existing, unchanged target application. The new application will also contain a Service Management system, which interacts with the cash register operator, and sends messages to the Customer Service Manager. These messages can be messages like, operator running out of change, operator needs a break, running out of paper, credit card authorization, call security etc. Instead of implementing these features on special keys, they will be implemented on a few ScreenKey console menus.

The problem is that the existing POS application cannot be changed to include these features. The new application can contain these features instead, and the PlayBack method can be used in conjunction with other SAC Engine Control methods and events, i.e. COM mode.

The SAC Engine will be started in combined mode, SAC_MODE and PLAYBACK_MODE, and normally the keystrokes assigned to the ScreenKey menus will be played back to the target application. Most menus in the SAC file are for this target application, but some are meant for the Service Management application, also hosting the SAC Engine Control.

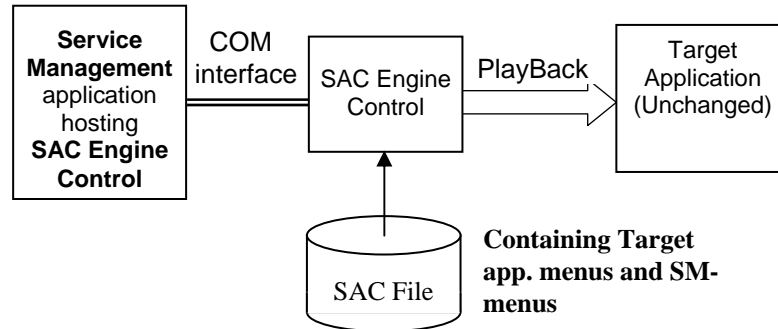


Figure 6 – Service Management application hosting SAC Engine Control, playback to existing unchanged application.

The problem will then be to redirect the output from these menus to the Service Management application. This can be done easily using the KeyStroke event, the MenuChange, or the KeyLockTurn event, and the PlayBackMode method. The keyboard will have one fixed key triggering the SM-system. The KeyStroke event can monitor for this key, and set the application in a certain mode when this is hit, or it can monitor the MenuChange event, and trigger this mode when a certain menu is about to be displayed.

When either is detected, the PlayBackMode method can be invoked to stop playback to the target application, so the SM-application can process the keystrokes until exiting the Service Management menus. This way the SM-application only needs to worry about its “own commands”, and just monitor the keystrokes or the menu changes.

When the Service Management menu has been entered, the application can work away with the SAC Engine Control as a COM Control, using the events, methods and properties, or it can use the PlayBack mode if that is preferred. It may use the GotoMenu, UpdateKey, SetWindow and the TrackWindow methods to enhance the functionality as described elsewhere in this document.

Security levels

Security levels only apply to applications supporting ScreenKey consoles equipped with a KeyLock. Each ScreenKey on each menu, and all fixed keys, have a security level. These levels are set up with the SAC Editor (SAC Editor documentation explains this in detail).

The KeyLock has up to five positions, but not all of these are available with all keys. These positions are the security levels 0 to 4. By turning the key clockwise the level will increase, and vice versa. If a key has a security level less than the current system security level (the KeyLock

position), the SAC Engine will make the key inactive. Inactive keys will not send keystroke events, they will not PlayBack any keystrokes to the target application and they will not activate a linked menu – they are totally inactive. Inactive ScreenKeys will be dark (no background color), but they will still display the assigned text or graphic. If the ScreenKey console is set up to beep when pressed, this beep will still be issued when pressed if the key is inactive.

The security, as described above, is fully taken care of by the SAC Engine. This scenario requires that the keys for the KeyLock be used. If the ScreenKey console is integrated into an application the security can be handled from the application. The SetSecurityLevel method offers three different security modes:

- **Normal Security Mode** This is the mode described above, where the KeyLock is used to control the security level. The security level can still be set from the application, using the SetSecurityLevel method, but the KeyLock will override this when turned. In this mode any keystrokes or GotoMenus defined for the KeyLock in the SAC file will work as specified.
- **Software Only Mode** In this mode the KeyLock will **not** influence the security level at all. When the method is invoked, a security level has to be specified as well. The system security level will change with the same effect as if the KeyLock was turned to the corresponding position, but any keystrokes or GotoMenus defined for the KeyLock in the SAC file will **NOT** be performed. The KeyLockChange event will still receive notifications when the key is turned, and it is up to the application to make use of this.
- **Software Simulation Mode** In this mode the KeyLock will **not** influence the security level at all. When the method is invoked, a security level has to be specified as well. The system security level will change with the same effect as if the KeyLock was turned to the corresponding position, and any keystrokes or GotoMenus defined for the KeyLock in the SAC file **will** be performed. This mode behaves exactly as if the KeyLock was turned. The KeyLockChange event will still receive notifications when the key is turned, and it is up to the application to make use of this.

In addition the SAC Engine can be instructed to restrict the security level to a specified range of levels. This can be done by setting the Minimum and/or Maximum security levels using the SetSecurityLevel method. These limitations apply to all security modes, and attempts to set the levels outside these boundaries will result in an error.

These different modes can be used if the security is set by using log-on passwords, card swipe etc., and these can also be used in combination with the KeyLock.

Modern systems/applications often have their own security system, independent of the KeyLock device, and the SetSecurityLevel method can be used to integrate the SAC Engine's own security system into this.

As mentioned above, each key's security level can be set in the SAC Editor, but if a key for some reason needs a dynamic security level, the level can be changed on a per key basis using the SetKeyAttribute method. For fixed keys, this level is kept until the SAC Engine is stopped, or the level is set again. For ScreenKeys the behavior is different. Since the same key can/will have a different meaning in different menus, they are treated as if they are different keys. The security levels set with the SetKeyAttribute method will therefore only last until another menu is displayed. If a key's security level is to be altered, it should be done in the MenuChange event. This way it will be handled every time the menu is displayed.

If a security level is always to be altered, this should be done in the SAC file, using the SAC Editor.

NOTE: The SetKeyAttribute may also be used to get a key to be auto repeating. This will last until it is reset or till the SAC Engine is stopped. Therefore, if a ScreenKey is set to auto repeat when a certain menu is displayed, and it should not auto repeat in the next menu, the auto repeat has to be reset in the MenuChange event for the next menu.

An example of advanced use of the SAC Engine's security system can be:

- The Operator uses a magnetic card to log on to the system, which sets the security level. The application then sets the SAC Engine's security level according to the operator's profile.
- The manager can then use a key to get access to parts of the system the operator wouldn't have access to. This KeyLock change can be detected in the KeyLockChange event, or by inspecting the KeyLockPos property.
- The manager can then log on to the system using his password or a card swipe as an extra security, and when he is accepted the application can set the security mode to the needed level.
- When the key is removed, which can only be done in position 0 and 1, the security level can be reset to the original level.

Target application – window tracking

When using PlayBack to remotely control an application using the SAC Engine, the target application has to be defined. In some cases the target application would be the application hosting the SAC Engine, and in other cases it would be a different application. In other cases it is necessary to send information to a given window, not just to the application.

There are a two ways to remotely control an application:

- Using the **JournalPlayback** function in windows to send the keystrokes to an application.
- Sending messages directly to a window, using the **PostMessage** API function

JournalPlayback

The JournalPlayback is the safest way. This feature requires that the application is the currently active one – the one receiving keystrokes from the QWERTY keyboard. The JournalPlayback handles shift states well, and it disables the mouse and the QWERTY keyboard while playing back to the application so it doesn't get interrupted or disturbed. JournalPlayback is meant for playing back macros recorded with the JournalRecord function, and therefore it handles the input as if it came from the QWERTY keyboard.

PostMessage

Another method is to use the PostMessage API function to send a message directly to a specific window. The specified window does not have to be the active one, it can remain in the background. When using the PostMessage function the shift state is not handled properly (according to Windows documentation), so this approach is better suited for sending types other than strings, e.g. to simulate clicking of a button, the button can be specified and a "Space" can be sent to the window. This would then be the same as selecting the button and hitting the space bar

to press it. In some cases the PostMessage would be chosen just because the window doesn't have to be the active one.

SetWindow

If the handler of the target window is known, e.g. if the target is the application hosting the SAC Engine itself, the target for the PlayBack or PostMessage can be set using the SetWindow method. With SetWindow there is a separate parameter PostmsgMode (which defaults to false) to specify if PostMessage is to be used instead of JournalPlayback.

The window handler can also be obtained by using the CurrentWindow property, which holds the window handler found with TrackWindow. Further details are described below.

TrackWindow

The TrackWindow method can be very useful, but requires some knowledge about the target window/application.

What TrackWindow does is to find the handler of a window specified with the method, and keeps track of this window. When a key is hit on the ScreenKey console its keystrokes will be fed to this window. If the window is not present when the key is hit, no playback will happen. The window does not have to exist when the method is invoked, but the window tracking functionality in the SAC Engine will detect the window when it is created, and start feeding to it then. If the window closes, feeding will stop until it appears again. Keystrokes will not be buffered, if the window is not present the keystrokes are discarded.

When the specified window is found, an event to the TrackChange will be fired to notify the application, and the same will happen when the window closes (window lost). If the specified window is found, the SAC Engine will verify that the windows is still present every five seconds, and if it is not found it will look for it every second.

The application will also be notified with an error message if the window is not found when the method is invoked, so the application can start the application.

The TrackWindow can also be used to find the handle of a window for later use. The application can invoke TrackWindow with a specification, and if the window is found, the handle can be retrieved using the CurrentWindow property. The handle can then be stored locally, and the TrackWindow can be invoked again to find another window. The SetWindow function can then be used to set the target window.

The advantage of this approach is that it is very efficient, but the disadvantage is that the window may have been closed in the meantime. There is no real danger using this approach because the SetWindow will also give an error if the window is not found – so the process of finding the window handlers can be repeated if it happens.

This approach is suitable if many controls (windows) within a dialog are to be controlled individually, for instance by activating buttons, lists etc. The window handlers of each control (window) can then be stored locally, and if one of them is still valid, the rest probably are too. The approach is very suitable in conjunction with the PlayBack method.

How to find a window

TrackWindow takes a string as parameter, which should be used to describe the window to feed keystrokes to. It can be a problem to specify the right window, because it is not obvious which window is owned by the application that actually is processing the input. TrackWindow has some default rules built into it that can make it easier.

TrackWindow has the following syntax:

```
TrackWindow( WindowSpec )
```

Where WindowSpec is a string for specifying the window.

WindowSpec MUST start with an application name, and it can have one or more options following this. The application name may be the filename only, a filename with extension, or the full path of the file. Relative paths are not allowed because there is no working folder defined.

The options are:

Option syntax	Description
/CF:Caption	Caption Full. The target window must have a caption containing the specified text entirely, no more – no less.
/CS:Subcaption	Caption Substring. The target window must have a caption containing the specified text.
/CL:classname	Class name. The target window must have the specified class name (full name required)
/CA	Caption Any. The target window must have a caption.
/HP	Has parent. The target window must have a parent window.
/NP	No parent. The target window CANNOT have a parent window.
/HC	Has child. The target window must have at least one child window.
/NC	No child. The target window CANNOT have a child window.
/SO:n	Start Order. The target application shall be the one started as the n'th one. These are applications with the same name. Ex: If three instances of notepad is running, this option can specify to use the one started second as the target application. Note: The Start Order is always in the present, which means that if an application that has Start Order 1 is being tracked, is closed, the application that has start order 2 becomes start order 1 (since it is the oldest), and this will then be the target application.
/RM	Relaxed mode. This mode allows the SAC Engine to use the first window that matches the specification. Normally if more than one window matches (non-relaxed mode), an error will be issued.
/PM	PostMessage mode. As described above for SetWindow.

NOTE: *The window tracking functionality will be enhanced in the next version of the SAC Engine Control and more options will be added to offer a more powerful window specification method. Because the approach will be a bit different, the syntax will be altered slightly. The existing options will be the same but for backward compatibility the user should enclose the options in brackets, which will be the future syntax when using more than one option to specify a certain window.*

Because Windows applications do not have a defined MAIN window, it can be hard to find the right window handler. It can often take some trial and error before the right one is found. If there

is such a thing as a typical Windows application, it will have a “main” window that has a caption, it has no parent, and it probably has one child window. For this reason default options are used when an application name is specified with no other options:

```
(/CA /NP /HC)
```

Except for options not specifying windows (/SO, /PM and /RM).

Typical examples are as follows:

```
SACEngine1.TrackWindow "Notepad.exe /SO:2"
```

- Tracks the second instance of notepad.

```
SACEngine1.TrackWindow "calc.exe /CF:bin"
```

```
SACEngine1.PlayBack " "
```

- Sets the Windows calculator to Binary mode (Scientific mode).

The EventStatus mask

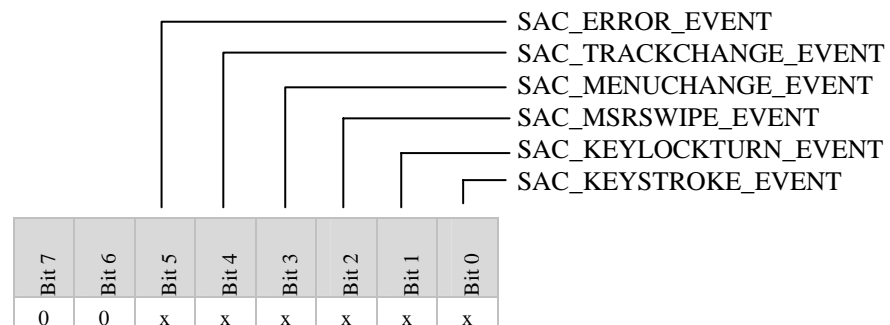
The EventStatus property is a read/write property holding information about the last events that has happened. It is provided for applications that want to poll for events instead of using the event handler. The reason for doing this may be that the programming language or framework used, or the existing application makes it difficult or impossible to implement an event handler.

Each event has a bit in the mask that will signal the event has occurred. When an event happens, either in Playback mode or in COM-mode, the corresponding bit in the mask is set.

Each bit must be cleared by the application when handled. If multiple bits are set, and the events are handled in different places, each bit must be reset individually. If all events are handled, the whole mask may be cleared by setting the property to zero.

The bits should be reset as soon as the corresponding data is read, to make sure no events and data are lost. As an example, if the bit for MSR events is set, the MSRTracks and MSRTrack properties should be read immediately, and then the MSR bit should be reset, before processing of the data starts.

The layout of the mask is as illustrated below:



Methods, Properties & Events

The methods, properties and events are the following:

Methods	Properties	Events
StartSAC	KeyCodes	KeyStroke
StopSAC	KeyNum	KeyLockTurn
GotoMenu	KeyLockPos	MSRSwipe
UpdateKey	MSRTrack	MenuChange
PlayBackMode	MSRTracks	TrackChange
PlayBack	CurrentMenu	SACError
SetSecurityLevel	CurrentWindow	
SetKeyAttribute	MenuName	
SetWindow	GraphicName	
TrackWindow	EventStatus	
WriteTextODA ¹	ErrorNumber	
WriteTextTDA ¹	ErrorDescription	
WriteTextCDA ¹	LLErrorNumber	
SelectTDA ²	LLErrorDescription	
SelectCDA ¹		
AboutBox (design time)		

Categories

The interface listed per category, Methods are marked (m), properties (p) and events (e):

Initialization/Configuration	
StartSAC (m)	Opens / Initializes the SAC Engine, which again opens and initializes the ScreenKey console.
StopSAC (m)	Shuts down the SAC Engine and closes the ScreenKey console.
MenuName (p)	Returns a specific name of a menu in the SAC file. Can also be used to enumerate menu names
GraphicName (p)	Returns a specific name of a graphic image in the SAC file. Can also be used to enumerate graphic names.
SetWindow (m)	Sets the Window handler to send the keystrokes to.
TrackWindow (m)	Sets the Application/window to track, for sending the keystrokes to.
CurrentWindow (p)	Returns the handler of the window currently the target window
WriteTextODA (m) ¹	Write supplied text to the specified ODA line number
WriteTextTDA (m) ¹	Write supplied text to the TDA (append new line)
WriteTextCDA (m) ¹	Write supplied text to the CDA. This does not automatically show the CDA display.
SelectCDA (m) ¹	Turns the CDA display on/off as specified
SelectTDA (m) ²	Selects between 1 of 4 buffers to display on the TDA
AboutBox	Displays the AboutBox containing version number. (Design time)

¹ These methods are only applicable when used with the SK-7000 console.

² This feature is not implemented in the current release of the SK-7000 firmware (1.0).

Data/Events	
KeyStroke (e)	Notifies the application that a KeyStroke has arrived., and that key codes are available.
KeyLockTurn (e)	Notifies the application that the Key Lock has been turned.
MSRSwipe (e)	Notifies the application that a magnetic card has been swiped, and that MSR data is available.
KeyCodes (p)	Property holding the key codes for the last pressed key.
KeyNum (p)	Holds the key number of last pressed key notified to the application by KeyStroke.
KeyLockPos (p)	Property holding the current KeyLock position.
MSRTrack (p)	Property holding the swiped MSR track data.
MSRTracks (p)	Property holding information about the available MSR data (which tracks).
CurrentMenu (p)	Property holding the name of the current menu displayed on the ScreenKey console.
PlayBackMode (m)	Enables or disables playback of keystrokes.
PlayBack (m)	Sends a key sequence from the hosting application to the target application
TrackChange (e)	Notifies the application that a change has happened in tracking a window.
EventStatus (p)	Property holding a bit mask describing events occurred.

Keyboard Control	
MenuChange (e)	Notifies the application that a menu is about to be displayed.
UpdateKey (m)	Enables the application to alter the contents and properties of a ScreenKey.
GotoMenu (m)	Allows the application to command a specific menu to be displayed on the ScreenKey console.
SetSecurityLevel (m)	Sets the current keyboard security level
SetKeyAttribute (m)	Sets the security level and repeat property for individual keys.

Errors/Debugging	
SACError (e)	Event triggered when asynchronous error occurs, when no method or property is invoked.
ErrorNumber (p)	Error number of last error.
ErrorDescription (p)	Description of last error.
LErrorNumber (p)	Error number of last low-level error.
LErrorDescription (p)	Description of last low-level error.

C O N T R O L M E T H O D S

Methods

StartSAC

Description

This method initiates the SAC processing, loads the controls, DLL, and reads all necessary files. When this method returns successfully, SAC processing is fully engaged and will continue to operate without any need for further interaction from the application.

Parameters

CommPort

Type: BSTR.

Name of comport to which ScreenKey console is connected, "COM1", "COM2" etc.

SACFile

Type: BSTR

Name of SAC file to be used. (*.pkf)

KbdFile

Type: BSTR

Name of keyboard definition file to use for playback. (*.kbd)

Mode

Type: SAC_OPENMODE

Mode of operation, and other initial settings.

Definition	Val	Description
SAC_COM_MODE	0001h	Run in COM mode. (Default)
SAC_PLAYBACK_MODE	0002h	Run in PlayBack mode
SAC_EVENT_ALL_KEYS	0004h	Send KeyStroke event also if key does not have keystrokes assigned to it.
SAC_REQ_INIT_KEYLOCK	0008h	Instigate the keyboard to send a KeyLockTurn event at startup, so application can "synchronize"
SAC_SHARED_MODE	0010h	Discontinued.
SAC_NO_LOWLEV_ERR	0020h	Don't report Low-level errors (except MSR errors)
SAC_NO_MSR_ERR	0040h	Don't report MSR errors
SAC_REPORT_ERRDESC	0080h	When sending errors as playback, also send error description.
SAC_NO_EXCEPTION	0100h	Don't raise exceptions when errors occur, only return HRESULTS.
SAC_DISCARD_ALL_CLIENTS	0200h	Discard all clients at start.
SAC_DISABLE_SCREEN_SAVER	0400h	Disable the screen saver when starting the SAC Engine.

Definition	Val	Description
SAC_ANSI_KEY_CODES	0800h	Return keystrokes for MFC/ANSI applications

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after: none**Remarks**

Description of the StartSAC modes and flags:

SAC_COM_MODE This mode is the normal mode of operation, the Control will behave like an ordinary COM control. See SAC Engine in COM-Mode, page 25. Default if no mode is given.

SAC_PLAYBACK_MODE This mode is when the keystrokes defined for the keys shall be played back to the target application through the keyboard message channel. See SAC Engine in PlayBack-Mode, page 26.

SAC_EVENT_ALL_KEYS If for some reason, all key presses are wanted and should be reported as KeyStroke events, not only those with keystrokes assigned to them, this flag should be set.

SAC_REQ_INIT_KEYLOCK If this flag is set, a KeyLockTurn event will be instigated at startup, so the application can use the event for synchronization purposes etc.

SAC_NO_LOWLEV_ERR If for some reason the application does not want to know about the low-level errors occurring this flag may be set. MSR errors will still be sent, and if these also shall be suppressed, the SAC_NO_MSR_ERR has to also be given.

SAC_NO_MSR_ERR The MSR errors will not be forwarded if this flag is set.

SAC_REPORT_ERRDESC When in PlayBack mode, and the "Error Stuffing Active" flag is set in the SAC Editor, the SAC Engine will playback the error numbers in the format: "ERRxxxxxxx", where x is the error number in 8 digit hexadecimal format. If this flag is set, the corresponding error description will be concatenated to the error number. The error numbers are the same numbers described under Error numbers and messages.

Example: "ERR00004002 Could not open ScreenKey console".

SAC_NO_EXCEPTION Normally, when using a COM control, it will raise an exception if an error occurs, and this will be captured by the exception handling in the programming language (run-time system). For Visual Basic, this can be controlled with the On Error statements. If the exception should not be raised, this flag can be set. The errors can instead be detected using the return value from the methods, and/or the ErrorNumber/ErrorDescription methods.

SAC_DISCARD_ALL_CLIENTS If the application using the SAC Engine has been terminated accidentally, the SAC Engine might not have been able to close the SKI ActiveX Control. This may then remain open and prevent the SAC Engine connecting to it. To prevent this situation, especially during development, this flag can be set and all clients of the SkAxCtl control will be discarded (by force), resulting in the control allowing the SAC Engine to connect to it. This should not be done in release versions, especially if the SAC_SHARED_MODE flag is in use. *Note that SHARED MODE access is discontinued in release 2.0 (current at time of writing) and later.*

SAC_DISABLE_SCREEN_SAVER When this flag is specified, the SAC Engine will disable the screen saver when starting, and enable it again when terminating. The reason for this is that the computer can hang if PlayBack is attempted while the screensaver is active (has kicked in). These problems are documented by Microsoft, and there is no indication that they will be resolved.

SAC_ANSI_KEY_CODES For use when hosting application is an MFC application using ANSI strings. Normally when the scancodes are returned, they can be retrieved as an array of shorts, where scancode value and ASCII value each takes one byte. The COM control returns a BSTR, which is an array of shorts, but when using ANSI strings, each BSTR element (a short) will be converted to a one-byte character. Since the short contains two bytes that are a Unicode character, this conversion will fail and the data will be meaningless. By using this flag, the array will be converted to a Unicode string before it is returned, and therefore it can be safely converted back to an ANSI string by the COM wrapper supplied by the MFC Wizard. See Using MFC on page 93.

StopSAC

Description

Call this method to stop the SAC file processing, and to end communication with the keyboard. If successful the application will not be able to send commands to the SAC Engine, nor receive events from it.

Parameters

None

Return Value

 Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after

StartSAC

Remarks

This method **MUST** be called when the application does not need the SAC Engine to process anymore. It will stop the SAC Engine Server (the .exe server), and also release the SKI ActiveX Control Server (.exe server) so it can be used by other applications.

GotoMenu

Description

With this method a named menu can be displayed on the ScreenKey console.

Parameters

MenuName

Type: BSTR

The name of the menu to be displayed on the ScreenKey console. The name is not case sensitive.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after StartSAC

Remarks

The GotoMenu command will trigger a MenuChange event, as any other change of menu. From the MenuChange event, the application may alter the menu.

This method may be used from any event except from MenuChange if the specified menu is the current menu as this would be a never ending recursive loop. It is however allowed used from the MenuChange event if a different menu is to be displayed.

UpdateKey

Description

This method can be used to change the contents and the characteristics of a specified ScreenKey.

Parameters

Row

Type: short
Row number of ScreenKey to update.

Column

Type: short
Column number of ScreenKey to update.

Line1

Type: BSTR
Text to appear on the topmost line of the ScreenKey.
If the text is specified as an empty string, the current text on the upper line will not be replaced, but if it is defined, either as a specific text or a text containing one or more blanks, the text will be replaced.

Line2

Type: BSTR
Text to appear on the bottom line of the ScreenKey. See Line1.

Graphics

Type: BSTR
Graphical image to appear on the ScreenKey. This can be combined with Line1 text and Line2 text, and will then be mapped (OR'ed) together with these.
The image can be specified two ways
* either as a name of a graphical image residing in the SAC file, or
* as a graphic array of bytes, or as a BSTR, see Altering ScreenKeys dynamically, page 28.

KeyColor

Type: SAC_KEY_COLORS
The background color and flash color of the ScreenKey. See Altering ScreenKeys dynamically, page 28.

DispAttrib

Type: SAC_DISP_ATTRIBUTE
The display attribute of the ScreenKey. See Altering ScreenKeys dynamically, page 28.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after StartSAC and MenuChange

Remarks

See separate description of this method, Altering ScreenKeys dynamically, page 28.

To verify the names of the graphical images, the property GraphicName can be used. It can enumerate all image names in the SAC file.

To use an array containing the image, an array of exact number of bytes for an LC16 or LC24 must be used. See SKI ActiveX Control User's Guide for description of the format. For LC24 images, the data may be sent as compiled (108 ytes) or uncompiled (120 bytes).

This checks if the Row and Col parameters are legal, but does NOT check KeyColor, nor the DispAttribute parameter.

PlayBackMode

Description

Enables or disables the keystroke playback. When using playback of keystrokes to an application, the playback can be disabled using this method to prevent the SAC Engine from sending keystrokes to the application.

Parameters

Mode

Type: SAC_PLAYBACK

Specifies the playback mode. The following choices are available:

Definition	Val	Description
SAC_PLAYBACK_OFF	0	Turns playback OFF immediately.
SAC_PLAYBACK_ON	1	Turns playback ON immediately.
SAC_PLAYBACK_NEXT_OFF	2	Turns playback OFF after the next series of keystrokes have been handled according to current mode (normally played back if we assume playback is on).
SAC_PLAYBACK_NEXT_ON	3	Turns playback ON after the next series of keystrokes have been handled according to current mode (normally rejected if we assume playback is off).
SAC_PLAYBACK_ONE_OFF	4	Turns playback OFF for the next series of keystrokes to be played back. When this series of keystrokes have been rejected, playback is turned ON.
SAC_PLAYBACK_ONE_ON	5	Turns playback ON for the next series of keystrokes to be played back. When this series of keystrokes have been played back, playback is turned OFF.

The following table shows the state of the playback mode before and after a new mode has been set:

Mode	Before PlayBackMode command	After PlayBackMode command	At first PlayBack	At second PlayBack
SAC_PLAYBACK_OFF	OFF	OFF	OFF	OFF
	ON	OFF	OFF	OFF
SAC_PLAYBACK_ON	OFF	ON	ON	ON
	ON	ON	ON	ON
SAC_PLAYBACK_NEXT_OFF	OFF	OFF	OFF	OFF
	ON	ON	ON	OFF
SAC_PLAYBACK_NEXT_ON	OFF	OFF	OFF	ON
	ON	ON	ON	ON
SAC_PLAYBACK_ONE_OFF	OFF	OFF	OFF	ON
	ON	OFF	OFF	ON
SAC_PLAYBACK_ONE_ON	OFF	ON	ON	OFF
	ON	ON	ON	OFF

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **StartSAC** (in PLAYBACK_MODE only)

Remarks

This method is only applicable when PlayBack mode is in operation. It enables or disables the playback in the specified way. It will typically be used from an event handler such as KeyStroke, KeyLockTurn, MSRSwipe or SACError, but may also be used from outside the event handlers.

See page 33 for detailed description and example.

PlayBack

Description

This method will send a series of keystrokes to the target application.

Parameters

KeyStrokes

Type: BSTR

Specifies the playback mode. The following choices are available:

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **StartSAC** (in PLAYBACK_MODE only)

Remarks

PlayBack allows the SAC Engine hosting application to send keystrokes to the target application. The reason for using this method would be if there is dynamic data needed that cannot be defined in the SAC file. It will typically be used from an event handler such as KeyStroke, KeyLockTurn, MSRSwipe or SACError, or from outside the event handlers, typically in conjunction with a GotoMenu command.

See PlayBackMode and PlayBack methods, page 33, for detailed description and example.

SetSecurityLevel

Description

Sets the SAC processing security level and/or mode. The security levels will normally be set automatically by the KeyLock.

Parameters

SecurityLevel

Type: SAC_SECURITYLEVEL

The security level or security mode to be set, see Security levels, page 36.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **StartSAC**

Remarks

This method offers software control of the keyboards security level.

The following options are available:

Definition	Val	Mutual Exclusive		Description
SAC_SEC_LEVEL_0	0000h	✓		Security level 0
SAC_SEC_LEVEL_1	0001h	✓		Security level 1
SAC_SEC_LEVEL_2	0002h	✓		Security level 2
SAC_SEC_LEVEL_3	0003h	✓		Security level 3
SAC_SEC_LEVEL_4	0004h	✓		Security level 4
SAC_SEC_SW_ONLY	0010h		✓	Set to software control only. Must be combined with a security level, see above
SAC_SEC_SW_SIM	0020h		✓	Set to software control, but imitates KeyLock functionality, it will send any keystrokes defined and/or goes to the specified menu. Must be combined with a security level, see above
SAC_SEC_MIN	0040h		✓	Sets the minimum security level allowed. The KeyLock will still work, but not below the minimum set. Must be combined with a security level, see above
SAC_SEC_MAX	0080h		✓	Sets the maximum security level allowed. The KeyLock will still work, but not below the maximum set. Must be combined with a security level, see above
SAC_SEC_NORMAL	0100h		✓	Set the security control back to normal, the KeyLock will work as normal. The system will get the security level corresponding to the KeyLock position when set to normal.

SAC_SEC_MIN and SAC_SEC_MAX may be combined, but not with the same command. They can be issued as separate commands to change both the minimum and maximum levels allowed.

SetKeyAttribute

Description

This is a special purpose method that can be used to set certain key attributes. It can be used to set the security level or the repeat configuration for individual keys

Parameters

Row

Type: short

Row number of ScreenKey to set security level for.

Column

Type: short

Column number of ScreenKey to set security level for.

KeyAttribute

Type: SAC_KEY_ATTRIBUTE

The security level or repeat attribute to be set. See below.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after StartSAC

Remarks

It can be used to set a specific key's or a group of key's security level, or it can enable or disable the repeat attribute for a key or a group of keys permanently.

The following options are available:

Definition	Val	Mutual Exclusive		Description
SAC_SEC_KEY_LEVEL_0	0000h	✓		Security level 0
SAC_SEC_KEY_LEVEL_1	0001h	✓		Security level 1
SAC_SEC_KEY_LEVEL_2	0002h	✓		Security level 2
SAC_SEC_KEY_LEVEL_3	0003h	✓		Security level 3
SAC_SEC_KEY_LEVEL_4	0004h	✓		Security level 4
SAC_SEC_SCREENKEYS	0010h		✓	Set security level for all ScreenKeys
SAC_SEC_FIXEDKEYS	0020h		✓	Set security level for all fixed keys
SAC_SEC_ALLKEYS	0030h		✓	Set security level for all keys
SAC_ENABLE_REPEAT	0100h		✓	Enable the key repeat for the specified key(s)
SAC_DISABLE_REPEAT	0200h		✓	Disable the key repeat for the specified key(s)

Setting the Security Level for a Fixed key will remain until the SAC Engine is stopped, while the Security Level set for a ScreenKey remains valid only until the menu is changed. The setting of the repeat attribute will remain until the SAC Engine is stopped, this goes for both Fixed keys and ScreenKeys.

SetWindow

Description

Sets the Window handler, the main window of the application to send the keystrokes to in PlayBack mode.

Parameters

WindowHandle

Type: long

Handle of target applications main window.

PostMsgMode

Type: BOOLEAN

If TRUE, the keystrokes will be fed to the window using the PostMessage function, instead of the JournalPlayback.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after StartSAC

Remarks

This method is only applicable when PlayBack mode is in operation. The target window handler has to be set when starting the SAC Engine, and this method can be used to redefine it at a later stage.

As an example; if an application is launching another application to do special tasks, it can grab the handler of this applications main window and tell the SAC controller to feed the keystrokes to this application. When the main application again gets activated, it can tell the SAC Engine to feed to this application again.

Note that if *WindowHandle* is NULL, the SAC Engine will feed keystrokes to the active application. NULL is the default Window handler, therefore if neither SetWindow nor TrackWindow is used the keystrokes will be fed to the active application.

JournalPlayback is normally easier to use, but using PostMessage, the keystrokes can be fed to the window without having to make it the topmost/active window. Windows controls may be operated directly by using PostMessage. See Target application – window tracking, page 38, for further information.

TrackWindow

Description

Describes the window/application to where the keystrokes are to be sent in PlayBack mode. The specification is based on a description of the target window, not the window handler as with SetWindow.

Parameters

WindowSpec

Type: BSTR

Text string describing the window to feed keystrokes to.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after StartSAC

Remarks

This method is not fully implemented.

The WindowSpec MUST start with the name of the application to feed keystrokes to, filename and extension (MyApp.exe).

See Target application – window tracking, page 38. for further information.

NOTE:

*The window tracking functionality will be enhanced in the next version of the SAC Engine Control and more options will be added to offer a more powerful window specification method. Because the approach will be a bit different, the syntax will be altered slightly. The existing options will be the same, but for backward compatibility the user should enclose the options in brackets, which will be the future syntax when using more than one option to specify a certain window. The application name part of the specification shall **not** be inside the brackets.*

Example: TrackWindow("Myapp.exe (/HP /HC /CL:Myclass)");

WriteTextODA

*Note: This method **only** applies to the SK-7000 console.*

Description

Send text to be displayed on the ODA.

Parameters

LineNo

Type: short

Line number of the ODA to display supplied text (1 = line 1, 2 = line 2, 0 = both lines).

Text

Type: BSTR

Text string containing the text to be displayed. The maximum string size is 20 characters. If *Text* contains NULL (i.e. 00hex) then the ODA line(s) is blanked.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after StartSAC

Remarks

This method is used to display text on the ODA of the SK-7000 console.

The ODA (Operator Display Area) is the upper segment of the SK-7000 large LCD display. It shows 2 lines of 20 characters each using a "large character" font.

The actual font is stored in the SK-7000 memory by means of an options file download.

This method will likely be used from a suitable point (or points) in an application where information for display purposes is available.

Typically the ODA is used to provide realtime console operator information, e.g. operator messages in a POS environment.

WriteTextTDA

*Note: This method **only** applies to the SK-7000 console.*

Description

Write text to a specified TDA buffer page.

Parameters

PageNo

Type: short

Page (buffer) number of the TDA to append the supplied text (0 = currently selected page, n = specified page).

Text

Type: BSTR

Text string containing the text to be appended. The maximum string size is 40 characters. If *Text* contains NULL (i.e. 00hex) then the TDA buffer is blanked.

Return Value

Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after

StartSAC

Remarks

This method is used to put text on the TDA of the SK-7000 console.

The TDA (Transaction Display Area) is the lower segment of the SK-7000 large LCD display. It shows 12 lines of 40 characters each using a “small character” font. The actual font is stored in the SK-7000 memory by means of an options file download.

The TDA is effectively a “window” for displaying information contained in the SK-7000 memory. The console maintains 1 or more buffers (pages) that can be written to using this method. The console operator scrolls up and down through the text on a selected page using the scroll buttons below the TDA.

Each time this method is called, the supplied text is appended as a new line to the specified page.

The default page size is 200 lines but this may be changed via the options file.

The method will likely be used from a suitable point (or points) in an application where information for display or print purposes is available.

Typically, the TDA is used to provide “transactional” data to the operator, e.g. in a retail POS environment, the TDA would be used to show receipt information. It can also be used to display discrete operator messages that the customer should not see, e.g. special checks to be performed.

NOTE:

*The selection and maintenance of multiple buffer pages for the TDA is **not** implemented in the SAC Engine Control at the time of writing (version 2.0). Please use PageNo = 0 for all uses of this method to always select the current page number.*

WriteTextCDA

*Note: This method **only** applies to the SK-7000 console.*

Description

Send text to be displayed on the CDA.

Parameters

Text

Type: BSTR

Text string containing the text to be displayed. The maximum string size is 20 characters. If *Text* contains NULL (i.e. 00hex) then the CDA display is blanked.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after StartSAC

Remarks

This method is used to display text on the CDA of the SK-7000 console.

The CDA (Control Display Area) is a “pop-up” display that appears on the lowest part of the SK-7000 large LCD display. It shows a single line of 20 characters using the same font as the ODA. The CDA is turned off by default. When displayed, it reduces the operating size of the TDA to 8 lines (instead of its normal 12). The TDA returns to normal when the CDA is turned off.

The CDA is automatically displayed when this method is called.

Typically, the CDA is used to provide “alert” or “warning” messages to the operator, e.g. in a retail POS environment, the CDA may be used to instruct the operator to check the bottom of the trolley at the end of a transaction.

SelectCDA

*Note: This method **only** applies to the SK-7000 console.*

Description

Turn the CDA on or off.

Parameters

OnOff

Type: BOOLEAN

If TRUE (1), the CDA is displayed. If FALSE (0) the CDA is turned off.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **StartSAC**

Remarks

This method is used to turn the display of the CDA on or off on the SK-7000 console.

The CDA is turned off by default. When turned on it displays the text last written to the CDA using the *WriteTextCDA* method.

SelectTDA

*Note: This method **only** applies to the SK-7000 console – see note below.*

Description

Select which TDA page buffer to display on the TDA.

Parameters

PageNo

Type: short

Page (buffer) number to display on the TDA.

Return Value Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **StartSAC**

Remarks

This method is used to select which page buffer to display on the TDA on the SK-7000 console.

This method can be useful for compiling two separate data pages and flipping them quickly. For example, in a retail POS environment one page (the default) can have receipt information being collected and displayed. Another TDA page could be created with different information (e.g. help info or product code listings). Then it is possible to quickly display this other page and then return to the main “receipt” page to continue with the transaction.

NOTE:

This method has not been implemented at time of writing (SAC Engine release 2.0).

WriteGraphicTDA

*Note: This method **only** applies to the SK-7000 console – see **note below**.*

Description

Send a graphic to be displayed on the TDA.

Parameters

Graphic

Type: BSTR

A monochrome (black and white) graphic image (dimensions 240 x 97 pixels).

Return Value

Type: long

Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after

StartSAC

Remarks

This method is used to display a graphic on the TDA of the SK-7000 console.

The TDA (Transaction Display Area) is 240 pixels wide by 97 pixels deep. Usually it displays text of 12 lines by 40 characters per line.

It can be used to display a graphic image supplied by this method. The graphic **must** conform to the correct dimensions (240x97 pixels) and must be monochrome (i.e. black and white only) as the SK-7000 display is a monochrome only display.

The TDA can only display either text **or** graphics. It will follow the last method called. If the CDA is turned on when this method is called, the CDA will automatically be turned off. The CDA will remain off if the TDA is changed back to showing text.

Typically, this method can be used to show graphic data in a “standby mode”, e.g. in a retail POS environment, display the store logo when the operator has signed off.

NOTE:

This method has not been implemented at time of writing (SAC Engine release 2.0).

Properties

KeyCodes

Type	BSTR
Parameter	short NoCodes
Get property	Supported
Put Property	Not supported
Available	After KeyStroke event Updated after each KeyStroke event received. Initially set to zero length string (before any key has been pressed that has keystrokes assigned).
Description	KeyCodes holds the sequence for the last pressed key that has keystrokes assigned to it. This data is only updated if a key event occurs from the keyboard. The BSTR string contains an array of 16-bit words, and in this case the result is to be handled as 16-bit words, where the Scancode is in the upper byte and the corresponding ASCII-value, if any, is in the lower byte. See SAC Editor documentation for further explanation.

The property can return a specific number of key codes, specified by NoCodes. NoCodes is the parameter in the KeyStroke event, which notifies the application about new data available. If NoCodes is set to -1, the full array is returned (the number given in the KeyStroke event). Care must be taken to ensure the array receiving the data is large enough to accommodate this. This property should be used from the KeyStroke event handler, to make sure data is not lost (new event before data is retrieved).

The property returns a BSTR, to get an array of words (16-bit). The problem receiving the data as a BSTR is that it will be converted depending on the current language setting. This BSTR array can easily be converted to a byte array and the array can then be used, without any risk of being converted.

Example from VB:

```
Dim KeyStrokes As Variant
Dim Keys() As Byte
```

```
KeyStrokes = SACEm.KeyCodes(NoKeys) ' Get the keystrokes
Keys = KeyStrokes ' Convert to byte array
```

```
' Check if first keystroke is an 'A' (65 in ASCII part)
```

```
If Keys(0) = 65 Then
```

```
...
```

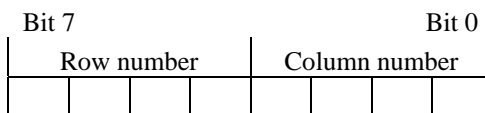
```
' Check if first keystroke is Alt-F10 (71h in Scancode part)
```

```
If Keys(1) = &H71 Then
```

```
...
```

KeyNum

Type	short
Get property	Supported
Put Property	Not supported
Available	After KeyStroke event (See Description)
Description	<p>Holds the number of last key pressed.</p> <p>Under normal operation, the key number is only updated if a key event occurs and this key has keystrokes associated with it. If the flag SAC_EVENT_ALL_KEYS is given when starting the SAC Engine, events will be fired and this property will be updated, also for keys not having keystrokes associated with it and no menu linked to it (dead keys). The data is coded in the following format:</p>



This property should be used from the KeyStroke event handler, to make sure data is not lost (new event before data is retrieved).
Initially, before any key is pressed, the property contains -1.

KeyLockPos

Type	short
Get property	Supported
Put Property	Not supported
Available	Always, updated after KeyLockTurn event
Description	<p>Holds the current KeyLock position.</p> <p>This property should be used from the KeyLockTurn event handler, to make sure data is not lost (new event before data is retrieved).</p> <p>Initially, before any changes have occurred, the property contains the initial KeyLock position. Using the flag SAC_REQ_INIT_KEYLOCK when starting the SAC Engine, StartSAC method will instigate a fake KeyLock change, that can be handled by the application, but one must be aware that it is not a real KeyLock turn happening.</p>

MSRTracks

Type	SAC_MSR_TRACKS
Get property	Supported
Put Property	Not supported
Available	After the MSRSwipe event.
Description	Holds a description of which tracks has been received from the keyboard. The following values are available:

Definition	Val	Description
SAC_MSR_NO_TRACKS	0	No track read
SAC_MSR_TRACK_1	1	Track 1 read, data in MSRTrack1Data
SAC_MSR_TRACK_2	2	Track 2 read, data in MSRTrack2Data
SAC_MSR_TRACK_3	4	Track 3 read, data in MSRTrack3Data
SAC_MSR_TRACK_1_2	3	Track 1 and track 2 read, data in MSRTrack1Data and MSRTrack2Data
SAC_MSR_TRACK_1_3	5	Track 1 and track 3 read, data in MSRTrack1Data and MSRTrack3Data
SAC_MSR_TRACK_2_3	6	Track 2 and track 3 read, data in MSRTrack2Data and MSRTrack3Data
SAC_MSR_TRACK_1_2_3	7	Track 1, track 2 and track 3 read, data in MSRTrack1Data, MSRTrack2Data and MSRTrack3Data (Not available)

Remarks	Note that from the SAC file the keyboard can be set up to return one track or multiple tracks. If the “Return Multiple Tracks” is not ticked, the keyboard will only return one track, which one depending on the setting of the “Preference”. See the “MCR Settings” dialog in the SAC Editor.
----------------	---

MSRTrack

Type	BSTR
Parameter	short TrackNo
Get property	Supported
Put Property	Not supported
Available	After MSRSwipe event
Description	The MSRTrack property holds the last Magnetic Strip Reader tracks. The data from MSR will be translated by the control into ASCII format. The property is an array of BSTR strings, it takes one parameter, which is the track number wanted, 1 for track 1, 2 for track 2, and 3 for track 3. To get all tracks, the property has to be invoked a number of times with the different track numbers as parameters.
Remarks	See MSRTracks.

CurrentMenu

Type	BSTR
Get property	Supported
Put Property	Not supported
Available	After first MenuChange event, which will occur when the initial menu is displayed.
Description	Holds the name of the current menu displayed.

CurrentWindow

Type	long
Get property	Supported
Put Property	Not supported
Available	After first SetWindow or TrackWindow.
Description	Holds the handler to the current target window. If zero, the current target window will be the currently active application.

MenuName

Type	BSTR
Parameter	short MenuInd
Get property	Supported
Put Property	Not supported
Available	Always
Description	This property offers a way to enumerate names of the menus in the currently loaded SAC file. It behaves like an array of strings. It takes one parameter, the index of the menu name wanted. To enumerate through all menu names, start at index zero, and work your way upwards until the property returns a blank string. The number is then the index where the property returned a blank string the first time.

GraphicName

Type	BSTR
Parameter	short GraphicInd
Get property	Supported
Put Property	Not supported
Available	Always
Description	This property offers a way to enumerate names of the graphics in the currently loaded SAC file. It behaves like an array of strings. It takes one parameter, the index of the menu name wanted. To enumerate through all menu names, start at index zero, and work your way upwards until the property returns a blank string. The number is then the index where the property returned a blank string the first time.

EventStatus

Type	SAC_EVENT_STATUS
Get property	Supported
Put Property	Supported
Available	Always
Description	<p>This property holds a bit mask describing the event that has occurred since it was last cleared. It is meant to be used by applications that, for any reason, do not implement or use the events.</p> <p>The property has to be polled, and whenever an event occurs the corresponding bit will be set in the mask.</p> <p>When the given event is handled by the application, it MUST clear the eventmask, or preferably the specific bit. If every event is handled immediately, the property may be set to zero.</p>

Make sure to reset the bit as soon as the properties necessary to process the event have been retrieved, so that new events can be detected if they occur while processing the current event.

Definition of the event:

Definition	Val	Description
SAC_NO_EVENTS	00h	No event has occurred
SAC_KEYSTROKE_EVENT	01h	KeyStroke event has occurred
SAC_KEYLOCKTURN_EVENT	02h	KeyLockTurn event has occurred
SAC_MSRSWIPE_EVENT	04h	MSRSwipe event has occurred
SAC_MENUCHANGE_EVENT	08h	MenuChange event has occurred
SAC_TRACKCHANGE_EVENT	10h	TrackChange event has occurred
SAC_ERROR_EVENT	20h	SACError event has occurred

NOTE: Using this property to detect and handle event is not the preferred approach, because it can be unreliable (events may be lost).

ErrorNumber

Type	SAC_ERRORS
Get property	Supported
Put Property	Not supported
Available	Always, updated after each method invoked, and event received.
Description	Holds the error number from the latest method invoked. If no error occurred the value is SAC_ERR_NONE (0).

This property can hold error codes generated by both the SAC Engine Control and errors generated by Windows. If an error occurred in the SKI ActiveX Control or in the keyboard, the error number can be obtained using the LLErrorNumber property. The low-level error codes (from SKI ActiveX Control) is described in the SKI ActiveX Control User's Guide. The corresponding error descriptions can be obtained using LLErrorDescription. See Error numbers and messages, page 74, for definitions of SAC Engine Control error codes.

ErrorDescription

Type	BSTR
Get property	Supported
Put Property	Not supported
Available	Always, updated after each method invoked, and event received.
Description	Holds the description of any error occurred from the last method invoked. If no error occurred the string is blank. (After an error has occurred, and another method is invoked, this error message is cleared. The same applies to ErrorNumber)

LLErrorNumber

Type	long
Get property	Supported
Put Property	Not supported
Available	Always, updated after each method invoked, and event received.
Description	Holds the error number of any error occurred in the Low-Level control (SKI ActiveX Control) or in the ScreenKey console, from the last method invoked, or from asynchronous error events. If no error occurred the value is 0.

See SKI ActiveX Control User's Guide for further details.

LLErrorDescription

Type	BSTR
Get property	Supported
Put Property	Not supported
Available	Always, updated after each method invoked, and event received.
Description	Holds the description of any error occurred in the Low-Level control (SKI ActiveX Control) or in the ScreenKey console, from the last method invoked, or from asynchronous error events. If no error occurred the value is 0.

C O N T R O L E V E N T S

Events

KeyStroke

Description

This event occurs when a key is pressed on the ScreenKey console, and the key has keystrokes assigned to it. It will also be triggered if the SAC Engine is started with the SAC_EVENT_ALL_KEYS flag.

Parameters

NoKey

Type: short

Number of keystrokes stored in the KeyCodes property. This is the number of keystrokes assigned to the key pressed.

Remarks

The property KeyCodes holds the keycodes/keystrokes assigned to the newly pressed key. The keystrokes should be retrieved immediately in the event handler to prevent the keystrokes from being overwritten if another KeyStroke event occurs immediately afterwards.

KeyLockTurn

Description

This event occurs when the KeyLock is turned on the ScreenKey console. The event passes a key lock position, which may be from 0 to 4. The KeyLockTurn event may be triggered in software (from an application) by using the method *SendSimpleCommand* with the parameter *REQUEST_KEYLOCK_EVENT*.

Parameters

KeyPos

Type: short

New KeyLock position, 0-4.

Remarks

The KeyLock position can also be obtained through the KeyLockPos property.

MSRSwipe

Description

This event occurs when a magnetic card has been swiped successfully.

Parameters

NoTracks

Type: short

Number of tracks received.

Remarks

The event only notifies the application that a card has been swiped, and the data may be retrieved accessing the MSR-properties. These properties are through MSRTracks, to retrieve which tracks have been read, and MSRTracks to read the data for each track.

MenuChange

Description

This event occurs when a menu is about to be displayed. The processing of the SAC file (displaying of the menu) does not resume until execution has returned from the event handler. This event is to inform the application that a new menu is about to be displayed, so the application can synchronize itself to the menus on the keyboard.

The event is also meant to give the application a chance to generate it's own menu, or change the contents of selected keys.

Parameters

MenuName

Type: BSTR

Name of the menu about to be displayed.

Remarks

For the application to change the caption of a ScreenKey, it can use the method *UpdateKey*. The MenuChange event handler is the only place the *UpdateKey* method can be used. See *Altering ScreenKeys dynamically*, page 28, for further details.

TrackChange

Description

This event occurs when the window tracking status has changed. The application will be notified so it can take necessary actions.

Parameters

WinState

Type: SAC_TRKSTAT

Enumeration value of the tracking state:

Definition	Val	Error description
SAC_WIN_NOT_FOUND	0000h	Specified window was not found
SAC_WIN_FOUND	0001h	Specified window was found
SAC_WIN_LOST	0002h	Tracked window has been closed
SAC_WIN_MULTI	0003h	Multiple windows matching specification was found

Remarks

This event will be fired when the TrackWindow method is invoked. It will also be sent when a tracked window has been closed, either because the SAC Engine has detected that it has been closed while monitoring it, or because it could not be found when a keystroke sequence is attempted sent to it. When a window has been found, the control will check for its existence every second, and when a window has not been found, or has been lost, it will look for it every 5 seconds.

SACError

Description

This event occurs when an error has occurred. These are errors occurred in the ScreenKey console or in the communication part of the SKI ActiveX Control Server. In addition, asynchronous errors occurring in the SAC Engine Control will be reported via this event. Errors occurring in a method, before execution is returned to the invoking application, and will not be reported through this event, only through an exception and through the properties ErrorNumber and ErrorDescription. The error codes reported from the low-level software or from the keyboard can be retrieved using the property LLErrorNumber, and the corresponding description using LLErrorDescription.

Parameters

ErrorCode

Type: SAC_ERRORS

SAC error numbers or standard Windows error numbers

ErrorDesc

Type: BSTR

Error description in text format, if non-Windows errors.

E R R O R S

Error Handling

The SAC Engine Control reports errors through runtime system (exceptions) and four properties in the SAC Engine Control itself. In addition it reports asynchronous errors via the SACError event. The control can be set up to trigger errors depending on the current setting. The SAC Engine Control properties will always contain the error codes if an error has occurred (independent of error mode), while the exception will only contain the error code if an exception is raised. This will appear differently in the different programming languages.

A parameter in the StartSAC method describes the desired error-reporting mode. The flags are:

Definition	Val	Description
SAC_NO_LOWLEV_ERR	0x0020	No low-level errors (errors occurred in the SKI ActiveX Control) will be reported. (Except for MSR errors)
SAC_NO_MSR_ERR	0x0040	No MSR errors will be reported.
SAC_NO_EXCEPTION	0x0100	Exceptions will not be raised when errors occur. Errors have to be detected by analyzing the return value of methods.
SAC_REPORT_ERRDESC	0x0080	A description of the error occurred will be reported in addition to the error code. This only applies when errors shall be sent to the application using PlayBack.

Error numbers and messages

The following errors may occur. These error codes are defined in the enumeration type `SAC_ERRORS`.

The following origin specifications apply;

C: SAC Engine Control

I: SAC Engine Control Interface

X: SKI ActiveX Control

K: ScreenKey console.

If LL column shows Y(es), it means that more detailed error codes may be obtained using the Low-Level error properties (`LLErrorNumber`).

If the Evt column shows Y(es), the error message can sent as an event because it then is an error not caused when invoking a method or property. If it is a result from invoking a method or a property, it raises an exception (if applicable) or is reported through the error properties only.

If the Evt column shows O(nly), the error can only happen as a result of a low-level or keyboard error.

Error definition	Error #	Origin	LL	Evt	Error description
<code>SAC_ERR_NONE</code>	0000h	All			None (No error occurred)
<code>SAC_ERR_OPENKBD</code>	4001h	C	Y		Error opening ScreenKey console
<code>SAC_ERR_LAYOUT</code>	4002h	C	Y		Error obtaining ScreenKey console layout
<code>SAC_ERR_KEYNUM</code>	4003h	C	Y		Undefined key number
<code>SAC_ERR_UPDATE</code>	4004h	C	Y		Error updating ScreenKey menu
<code>SAC_ERR_CFGMATCH</code>	4005h	C			SAC file does not match ScreenKey console
<code>SAC_ERR_CONFIG</code>	4006h	C	Y		Error configuring ScreenKey console
<code>SAC_ERR_PLAYBACKDLL</code>	4007h	C			Error loading playback DLL (SKPlayBack.DLL)
<code>SAC_ERR_PLAYBACKADDR</code>	4008h	C			Error getting playback ProcAddr
<code>SAC_ERR_KBDFILEOPEN</code>	4009h	C			Error loading .kbd file
<code>SAC_ERR_SACOPEN</code>	400Ah	C			Failed to open SAC file
<code>SAC_ERR_SACCORRUPT</code>	400Bh	C			SAC file seems to be corrupt
<code>SAC_ERR_NOMENUNAME</code>	400Ch	C			Menu name not found
<code>SAC_ERR_NOGRAPHICNAME</code>	400Dh	C			Graphic name not found
<code>SAC_ERR_MSRRERROR</code>	400Eh	C			No. of MSR tracks read is 0
<code>SAC_ERR_NO_SAC_ENGINE</code>	400Fh	C			SAC Engine not started
<code>SAC_ERR_NO_PLAYBACK</code>	4010h	C			Not in PlayBack mode
<code>SAC_ERR_UNDEF_KEY</code>	4011h	C			Undefined key

Error definition	Error #	Origin	LL	Evt	Error description
SAC_ERR_CLAIM	4012H	C			Failed to claim keyboard
SAC_ERR_KBD_LOST	4013h	C			Keyboard lost
SAC_ERR_ILL_SECURITY	4014h	C			Illegal security level/mode
SAC_ERR_PB_TIMEOUT	4015h	C			Playback timed out
SAC_ERR_NOT_FROM_EVT	4016h	C			Not allowed from this event
SAC_ERR_NOT_FROM_APP	4017h	C			Not allowed from application
SAC_ERR_ILL_PAR	4018h	C			Illegal parameter/combination
SAC_ERR_GET_OSVER	4024h	C			Error retrieving OS Version
SAC_ERR_UNKNOWN_OS	4025h	C			Unknown operating system/version
SAC_ERR_PSAPIDLL	4026h	C			Error loading PSApi DLL (PSAPI.DLL)
SAC_ERR_PSAPIADDR	4027h	C			Error getting PSApi ProcAddr's
SAC_ERR_PSMEMALLOC	4028h	C		Y	Failed to allocate memory during window tracking.
SAC_ERR_ENUM_PROC	4029h	C		Y	Failed to enumerate processes
SAC_ERR_ENUM_WIN	402Ah	C		Y	Failed to enumerate windows
SAC_ERR_WRONG_PAR	402Bh	C		Y	Illegal window specification parameter
SAC_ERR_SNAPSHOT	402Ch	C		Y	SnapShot function failed
SAC_ERR_PAR_TOO_LONG	402Dh	C			Parameter string too long (256)
SAC_ERR_NO_PROCESS	4031h	C		Y	Could not find process/application; no window
SAC_ERR_NO_WINDOW	4032h	C		Y	Could not find window
SAC_ERR_WIN_MULTI	4033h	C		Y	Multiple windows found in strict mode
SAC_ERR_TRACK_THREAD	4034h	C			Could not create window-tracking thread.
SAC_ERR_SRV_STARTED	4041h	I			SAC Engine server is already started
SAC_ERR_CREATE_SRV	4042h	I			Failed to create SAC Engine server
SAC_ERR_NO_SERVER	4043h	I			SAC Engine server not started
SAC_ERR_SKAXCTL	40A1h	X	Y	O	Low-level error (SkAxCtl)
SAC_ERR_KEYBOARD	40A2h	K	Y	O	Low-level error (Keyboard)

In addition to SAC_ERR_SKAXCTL and SAC_ERR_KEYBOARD, ordinary Windows error codes can be sent as events to the application, and these have to be interpreted as normal errors occurring in an application.

T R O U B L E S H O O T I N G

Troubleshooting

This chapter describes how to troubleshoot when integrating the SAC Engine Control into an application.

Installation troubleshooting is discussed under "Software Installation", page 9.

Errors may occur at different stages of the development process and when in normal use of the application. Errors may occur when a method or property is invoked, and these errors may be trapped by using exception handling or by checking the `ErrorNumber / ErrorDescription` property. If an error has occurred at a lower level, in the SKI ActiveX Control, the error number and description can be obtained using the properties `LLErrorNumber` and `LLErrorDescription`. Errors may also happen when the application is not using the control directly, like keyboard errors and communication errors (because the communication with the keyboard is asynchronous), or in the processing of the SAC file. These will be reported through the `SACError` Event handler. It is important to handle all errors properly, including during application development. This way errors and design problems can be exposed, identified and corrected at an early stage. See also Error numbers and messages, page 74.

There are two main types of errors reported to the PC. Those occurring in the SAC Engine Control, and those occurring in the keyboard or in the SKI ActiveX Control. The errors ticked in the LL column in the table above are errors that occur outside the SAC Engine Control, and their real error numbers and description can be obtained by using the properties `LLErrorNumber` and `LLErrorDescription`, see SKI ActiveX Control User's Guide for further information.

SAC Engine Errors

SAC_ERR_NONE

0h

(blank)

Contents of `ErrorNumber` when a method or property has been invoked successfully.

SAC_ERR_OPENKBD **4001h****Error opening ScreenKey console**

This error message will be given when the SAC Engine Control cannot open and initialize the ScreenKey console successfully.

This error is a local SKI ActiveX Control error and can have numerous reasons. A more detailed error message can be obtained through the LLErrorNumber and LLErrorDescription properties. Refer to the SKI ActiveX Control User's Guide for details.

SAC_ERR_LAYOUT **4002h****Error obtaining ScreenKey console layout**

This error message will be given by the SAC Engine Control when it cannot obtain the keyboard layout from the SKI ActiveX Control.

This error may be an error in the SKI ActiveX Control, or in the ScreenKey console. It could also be a problem related to the creation of the control, in that case see SAC_ERR_OPENKBD.

A more detailed error message may be obtained through the LLErrorNumber and LLErrorDescription properties. Refer to the SKI ActiveX Control User's Guide for details.

SAC_ERR_KEYNUM **4003h****Undefined key number**

This error message will be given by the SAC Engine Control when renumbering the keys on a ScreenKey console fails.

If this is the first error that occurs, verify that the type of keyboard your SAC file is meant for matches the ScreenKey console attached to the specified comm-port. If nothing wrong is found, please contact your ScreenKey console distributor or support.products@ScreenKeys.com. If other errors occur before this, sort out these errors first.

A more detailed error message may be obtained through the LLErrorNumber and LLErrorDescription properties. Refer to the SKI ActiveX Control User's Guide for details.

SAC_ERR_UPDATE**4004h****Error updating ScreenKey menu**

This error message will be given by the SAC Engine Control when it fails to update the menu on the ScreenKey console.

This error would normally occur in the SKI ActiveX Control, and a more detailed error message can be obtained through the LLErrorNumber and LLErrorDescription properties. Refer to the SKI ActiveX Control User's Guide for details.

SAC_ERR_CFGMATCH**4005h****SAC file does not match ScreenKey console**

This error message will be given by the SAC Engine Control if the ScreenKey console connected does not match the SAC file's target console. This error is also reported if the SAC file and target console use different ScreenKeys resolutions and/or backlight colors (e.g. RG instead of RGB or LC16 instead of LC24).

To correct this problem, attach the right ScreenKey console or load a SAC file designed for the attached keyboard.

SAC_ERR_CONFIG**4006h****Error configuring ScreenKey console**

This error message will be given by the SAC Engine Control if an error occurs when the control is trying to configure the ScreenKey console

This error may be an error in the SKI ActiveX Control, or in the ScreenKey console. A more detailed error message may be obtained through the LLErrorNumber and LLErrorDescription properties. Refer to the SKI ActiveX Control User's Guide for details.

SAC_ERR_PLAYBACKDLL**4007h****Error loading playback DLL (SKPlayBack.DLL)**

This error message will be given by the SAC Engine Control when it fails to load the playback DLL library.

To correct this problem, make sure the installation of the SAC Engine Control was successful, and reinstall if necessary. If this happens with a third party installation (application redistributing the SAC Engine Control), make sure the installation is installing the SKPlayBack.DLL to a folder where the control can find it.

SAC_ERR_PLAYBACKADDR **4008h****Error getting playback ProcAddr**

This error message will be given by the SAC Engine Control if it cannot retrieve the procedure address of a certain function in the playback DLL library.

To correct this problem, make sure the installation of the SAC Engine Control was successful, and that any previous version has been uninstalled before reinstalling. Follow the procedures for the previous error, and contact support.products@ScreenKeys.com if the problem persists.

SAC_ERR_KBDFILEOPEN **4009h****Error loading .kbd file**

This error message will be given by the SAC Engine Control if it cannot load the template keyboard file (.kbd) properly.

To correct this problem, make sure the installation of the SAC Editor was successful, or that the .kbd file specified with the StartSAC method was installed in the specified folder. Be aware that this file is not a part of either the SKI ActiveX Control, or the SAC Engine Control, but a part of the SAC File Editor. See SAC Editor documentation for further information on the .kbd file.

SAC_ERR_SACOPEN **400Ah****Failed to open SAC file**

This error message will be given by the SAC Engine Control if it cannot find the specified SAC file.

To correct this problem, verify the name of the SAC file, and make sure the file is to be found in the specified folder.

SAC_ERR_SACCORRUPT **400Bh****SAC file seems to be corrupt**

This error message will be given by the SAC Engine Control if it cannot load the SAC file properly.

To correct this problem, try to run application once again. If the error persists, verify the SAC file by loading it into the SAC Editor, and if necessary, save it again. If the error still persists, there could be something wrong with the storage media or the hardware itself.

SAC_ERR_NOMENUNAME**400Ch****Menu name not found**

This error message will be given by the SAC Engine Control if it cannot find the specified menu name in the currently loaded SAC file.

To correct this problem, make sure the spelling of the name is correct, and that the menu exists in the currently loaded SAC file. The property `MenuName` may be used to enumerate the menu names in a SAC file, and the application can then verify that the named menu is present.

SAC_ERR_NOGRAPHICNAME**400Dh****Graphic name not found**

This error message will be given by the SAC Engine Control if it cannot find the specified graphic name in the currently loaded SAC file.

To correct this problem, make sure the spelling of the name is correct, and that the graphical image exists in the currently loaded SAC file. The property `GraphicName` may be used to enumerate the graphic names in a SAC file, and the application can then verify that the named graphic is present.

SAC_ERR_MSRError**400Eh****No. of MSR tracks read is 0**

This error message will be given by the SAC Engine Control if it receives a MSR event, but zero tracks were read.

To correct this problem, verify that the magnetic card can be read properly on another card reader. If it can, it is probably something wrong with the MSR device.

SAC_ERR_NO_SAC_ENGINE**400Fh****SAC Engine not started.**

This error message will be given by the SAC Engine Control when a method is invoked before the SAC Engine Control has been properly started.

This problem can occur in different situations. It may be because the control has not been started, because it failed when trying to start, or the control may have been stopped. Make sure the SAC Engine is started before attempting to invoke other methods.

SAC_ERR_NO_PLAYBACK **4010h****Not in PlayBack mode**

This error message will be given by the SAC Engine Control if a method requiring PlayBack mode is invoked while not in playback mode.

To correct this problem, make sure the control is started in PlayBack mode before such a method is invoked

SAC_ERR_UNDEF_KEY **4011h****Undefined key.**

This error message will be given by the SAC Engine Control if a it is attempted to update a non-existing ScreenKey.

To correct this problem, make sure the key to be updated exists. Also make sure that the key refers to the keyboard attached.

SAC_ERR_CLAIM **4012h****Failed to claim keyboard**

This error message will be given by the SAC Engine Control if the SAC Engine Control was not able to claim the keyboard when the SAC Engine was started. The reason for this error can be that the keyboard has been claimed by an application not complying with the rules, stated by the SAC Engine Control, for inter process communication when sharing a ScreenKey console.

To correct this problem, make sure all applications using the ScreenKey console follows the rules for sharing the keyboard.

SAC_ERR_KBD_LOST **4013h****Keyboard lost**

This error message will be given by the SAC Engine Control if a method sending commands to the keyboard is called after the keyboard control has been lost to another application.

It is not possible to detect if the keyboard control has been lost to another application, and therefore there is nothing that can be done to prevent this error situation. What the application hosting the SAC Engine can do, is to reissue the command when the control is regained.

SAC_ERR_ILL_SECURITY**4014h****Illegal security level/mode**

This error message will be given by the SAC Engine Control if an illegal security level or mode is given.

To correct this problem, make sure that the security level and mode used is legal.

SAC_ERR_PB_TIMEOUT**4015h****Playback timed out**

This error message will be given by the SAC Engine Control when the PlayBack method is used from the hosting application and the timeout value specified (if in use) was not long enough to playback the specified keystroke sequence.

This error would normally occur if there is a problem with the target window specified.

To correct this problem, make sure the target window is available and behaves right, or increase the timeout delay.

AC_ERR_NOT_FROM_EVT**4016h****Not allowed from this event**

This error message will be given by the SAC Engine Control if an illegal method is invoked from an event. Some methods are not allowed from certain events, like StopSAC is only allowed from the application itself, and UpdateKey is only allowed from the MenuChange event.

To correct this problem, the design of the hosting application has to be changed.

SAC_ERR_NOT_FROM_APP**4017h****Not allowed from application**

This error message will be given by the SAC Engine Control if a method only allowed from an event has been invoked from the application itself. UpdateKey is only allowed from an event, and only from the MenuChange event.

To correct this problem, the design of the hosting application has to be changed.

SAC_ERR_ILL_PAR **4018h****Illegal parameter/combination**

This error message will be given by the SAC Engine Control if the parameters given with a method are illegal.

To correct this problem, inspect and correct the parameters/ modes given with the method.

SAC_ERR_GET_OSVER **4024h****Error retrieving OS Version**

This error message will be given by the SAC Engine server when it failed to obtain the operating system version.

The reason for this problem could be the operating system is corrupt, or running out of resources. To correct this problem, try to restart the computer, and start the SAC Engine again.

SAC_ERR_UNKNOWN_OS **4025h****Unknown operating system/version**

This error message will be given by the SAC Engine server if the OS version obtained is unknown or unsupported.

To correct this problem, make sure the operating system is supported. Alternatively, try to restart the computer, and start the SAC Engine again.

SAC_ERR_PSAPIDLL **4026h****Error loading PSApi DLL (PSAPI.DLL)**

This error message will be given by the SAC Engine server if the PSAPI.DLL library could not be loaded.

To correct this problem, make sure the file PSIAPI.DLL is in the working folder, or in the system folder. Try reinstalling the SAC Engine Control.

SAC_ERR_PSAPIADDR **4027h****Error getting PSApi ProcAddr's**

This error message will be given by the SAC Engine server if the functions needed in PSAPI.DLL could not be loaded.

To correct this problem, make sure the file PSIAPI.DLL is not corrupted in any way, and if necessary reinstall the SAC Engine.

SAC_ERR_PSMEMALLOC 4028h**Failed to allocate memory during window tracking.**

This error message will be given by the SAC Engine server if memory from the heap could not be allocated.

To correct this problem, make sure there is enough memory and resources. Try restarting the computer if the problem is persistent.

SAC_ERR_ENUM_PROC 4029h**Failed to enumerate processes.**

This error message will be given by the SAC Engine server if the processes could not be enumerated.

To correct this problem, contact support.products@ScreenKeys.com.

SAC_ERR_ENUM_WIN 402Ah**Failed to enumerate windows.**

This error message will be given by the SAC Engine server if the windows could not be enumerated.

To correct this problem, contact support.products@ScreenKeys.com.

SAC_ERR_WRONG_PAR 402Bh**Illegal window specification parameter.**

This error message will be given by the SAC Engine server if an illegal parameter (option) has been given with the TrackWindow method.

SAC_ERR_SNAPSHOT 402Ch**SnapShot function failed.**

This is a system error that may occur when using TrackWindow on Windows 95 and 98.

To correct this problem, contact support.products@ScreenKeys.com.

SAC_ERR_PAR_TOO_LONG **402Dh****Parameter string too long (256).**

This error message will be given by the SAC Engine server if the parameter string given with TrackWindow is too long, the maximum is 256 characters.

To correct this problem the parameter string has to be shortened.

SAC_ERR_NO_PROCESS **4031h****Could not find process/application; no window.**

This error message will be given by the SAC Engine server if the process/application specified with TrackWindow could not be found.

To correct this problem, change the application specification, or start the specified application.

SAC_ERR_NO_WINDOW **4032h****Could not find window.**

This error message will be given by the SAC Engine server if the window specified with TrackWindow could not be found.

To correct this problem, change the window specification, or start the specified application.

SAC_ERR_WIN_MULTI **4033h****Multiple windows found in strict mode.**

This error message will be given by the SAC Engine server if the multiple windows were found matching the specification given with TrackWindow.

To correct this problem, change the window specification to a unique window description.

SAC_ERR_TRACK_THREAD **4034h****Could not create window tracking thread.**

This error message will be given by the SAC Engine server if the thread tracking a window could not be started.

To correct this problem, restart the computer, and make sure there are enough system resources.

SAC_ERR_SRV_STARTED 4041h**SAC Engine server is already started.**

This error message will be given by the SAC Engine Interface when it is attempted to start the SAC Engine Control more than once.

To correct this problem, make sure the SAC Engine Control only uses StartSAC once, and that any buttons or commands used to start the SAC Engine only can be invoked once.

SAC_ERR_CREATE_SRV 4042h**Failed to create SAC Engine server.**

This error message will be given by the SAC Engine Interface when the control interface cannot create the SAC Engine server.

The reason for this problem can be that the SAC Engine Control or the SAC Engine Proxy Stub is not installed, and registered properly.

SAC_ERR_NO_SERVER 4043h**SAC Engine server not started.**

This error message will be given by the SAC Engine Interface when a method is invoked before the SAC Engine server has been started.

This problem can occur in different situations. The first is when the control has not been started, either because it failed, or because there was no attempt to start it, or the control may have been stopped. If the first case, the StartSAC method should have given the error message SAC_ERR_CREATE_SRV (see above).

SAC_ERR_SKAXCTL 40A1h**Low-level error (SkAxCtl).**

This error message will be given by the SAC Engine Control if an error occurs when the ScreenKey console is updated as a result of operator actions.

This error occurs in the SKI ActiveX Control itself, and a detailed error message can be obtained through the LLErrorNumber and LLErrorDescription properties. Refer to the SKI ActiveX Control User's Guide for details.

SAC_ERR_KEYBOARD**40A2h****Low-level error (Keyboard).**

This error message will be given by the SAC Engine Control if an error occurs when the ScreenKey console is updated as a result of operator actions.

This error occurs in the ScreenKey console itself, and a detailed error message can be obtained through the LLErrorNumber and LLErrorDescription properties. Refer to the SKI ActiveX Control User's Guide for details.

Language Interfaces



The SAC Engine Control may be used from different programming languages. The syntax depends on the language, and a few will be described below.

Visual Basic

Visual Basic supports the COM interface, in its own process space, not in a separate process space. VB therefore has to interface to the SAC Engine Control Interface (SACEngIf.dll), not directly to the SAC Engine Control (SACEngin.exe).

VB (Visual Studio) supports IntelliSense and parameter enumeration, which is implemented in the SAC Engine Control.

To use the SAC Engine Control in a Visual Basic program use the following procedure:

- Start Visual Basic, 6.0
- Create new project or open existing
- Select the **Components** menu item from the **Project** menu.
- Select the **Control** tab, and scroll down the list until you find “**SAC Engine Control**”. Tick the box to the left, and click **OK**.
- The SAC Engine Control icon  will then appear on the ToolBox.
- Click on the icon, and “draw” it on the main form, like any other control. It will appear as an icon, but will be invisible when running the program. 
- The object will by default be named SACEngine1, but it can be renamed by redefining the “(Name)” property.
- The SAC Engine keyboard is now accessible from the application

For invoking methods, there are two different syntaxes available from VB. Which one to use is up to the programmer. Using OpenKeyboard as an example:

```
Call SACEngine1.StartSAC( "COM1", "SACFile.pkf", "Kybgenuk.kbd",
                        SAC_COM_MODE )
```

or

```
SACEngine1.StartSAC "COM1", "SACFile.pkf", "Kybgenuk.kbd", SAC_COM_MODE
```

As an example of use from Visual Basic, we will open the keyboard in the Form load event function:

```

Private Sub Form_Load()
On Error GoTo ErrHandler      ' Take care of errors in handler

    ' Example on use of METHOD
    ' Start the SAC Engine,
    SACEngine1.StartSAC "COM1", SACFile, KbdFile, SAC_COM_MODE

Exit Sub

ErrHandler:      ' Capture errors here
    MsgBox Err.Description      ' Alert user using a message box

End Sub

' Unload the form
Private Sub Form_Unload(Cancel As Integer)
On Error Resume Next      ' Keep going even if error occurs

    ' Shut down the SAC Engine
    SACEngine1.StopSAC

End Sub

' Example on use of EVENT, and a property
' When an event is received through the MSRSwipe event,
' show the user the tracks that has been read from the MSR
Private Sub SACEngine1_MSRSwipe( ByVal NoTracks As Integer )
    Dim Tracks as SAC_MSR_TRACKS
    Dim Track1 as String
    Dim Track2 as String

    ' Get the tracks, only track 1 and 2 can be available
    Tracks = SACEngine1.MSRTracks
    Select Case Tracks
        Case SAC_MSR_TRACK_1
            Track1 = SACEngine1.MSRTrack( 1 )
        Case SAC_MSR_TRACK_2
            Track2 = SACEngine1.MSRTrack( 2 )
        Case SAC_MSR_TRACK_1_2
            Track1 = SACEngine1.MSRTrack( 1 )
            Track2 = SACEngine1.MSRTrack( 2 )
    End Select
    MsgBox "Track1: " + Track1 + chr(13) + "Track2: " + Track2

End Sub

Private Sub SACEngine1_KeyLockTurn ( ByVal KeyLockPos As Integer )

    ' In position 2 the Manager application must be launched
    If KeyLockPos = 2 Then
        LaunchManagerApp()
    Endif

End Sub

```

Visual C++

Visual C++ supports the COM interface, both in its own process space, and in a separate process space. Since different frameworks can be used, this document will not cover how to get going in C++, except for the error handling and some special considerations.

The error handling can be done in two different ways in C++, as in VB. The errors may be trapped by using the try/catch construction, or by disabling the run-time errors and using the `LineNumber` or `ErrorDescription` properties.

Example of catching errors using the exception handling in C++.

```
void CSKCTLTestDlg::OnInit()
{
    CString csComPort;
    TCHAR szErrMsg[255];

    try {
        // Get the COMM-port string from the drop-down box
        GetDlgItemText(IDC_COMPOR, csComPort);
        // Start SAC Engine
        SACEngine.StartSAC(csComPort, SACFile, KbdFile, SAC_COM_MODE, hWnd);
        // Update the status field
        SetDlgItemText( IDC_STATUS, "SAC Engine Started!" );
    }

    // Catch if any error has occurred
    catch ( CException *Err ) {
        // Get the error message string
        Err->GetErrorMessage( szErrMsg, sizeof(szErrMsg) );
        // Display the error message in the status field
        SetDlgItemText( IDC_STATUS, szErrMsg );
    }
}
```

Example of catching errors using the SAC Engine properties.

```
void CSKCTLTestDlg::OnInit( void )
{
    CString csComPort;

    // Start the SAC Engine with COMM port specified as VOM1,
    // this should generate an exception
    SACEngine.StartSAC( "", SACFile, KbdFile, SAC_COM_MODE, hWnd );

    // If error occurred display the error message, else an "all well" msg.
    if (SACEngine.GetErrorNumber() != 0 )
        SetDlgItemText( IDC_STATUS, SACEngine.GetErrorDescription() );
    else
        SetDlgItemText( IDC_STATUS, "Keyboard Initialized!" );
}
```

Using MFC

UpdateKey method

Some special considerations have to be taken when using the SAC Engine Control from an MFC application when dealing with graphical images, the UpdateKey method. When adding the control to an MFC application the wizard generates a wrapper class, normally called *sacengine.cpp*, that forms the interface to the COM control. Wherever a BSTR type is used, the wrapper class uses the LPCTSTR type.

The LPCTSTR can either be a wide string if Unicode is used, or a standard ANSI multi byte string if it is not. The difference between these string and a BSTR type is that they cannot contain zero bytes, they are zero terminated, and will clip the string at the first zero-byte (zero-short for Unicode). The BSTR on the other hand contains information about the length of the string, which makes it suitable for containing zero bytes.

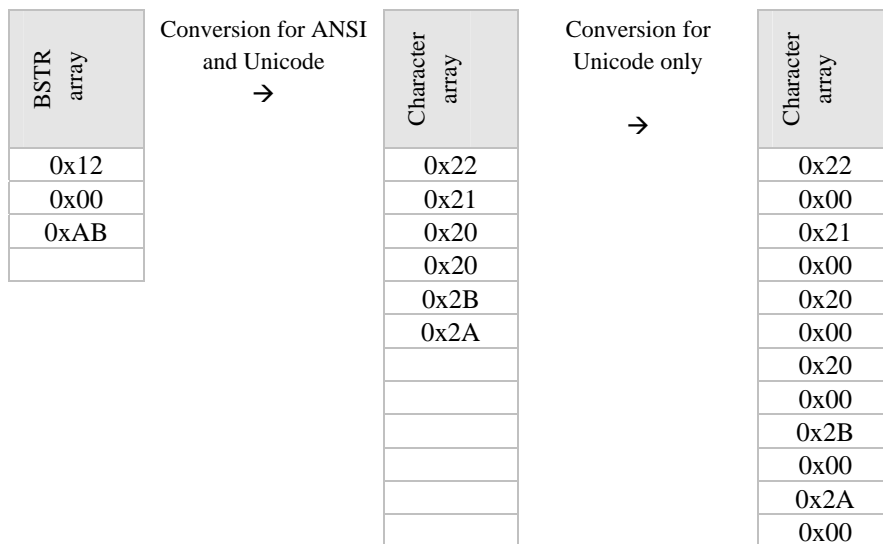
The graphics (images) used by the control may contain zero bytes. This works fine with all programming languages and frameworks supported, except MFC. The problem applies to both ANSI and Unicode versions of MFC applications. For such applications we therefore need to use a different format to avoid the problem, which is really a workaround of the problem.

The control will auto-detect the supplied format. A class containing functions for converting the images to the recognized format is supplied with the development installation of the control. The file is named SkMfcCnv.h.

This file contains a class named CSkMfcCnv, which can be used for converting a byte array containing an image, used by the above mentioned method. The conversion is as follows:

The low nibble of the first byte is store in the first byte and the value 0x20 is added to bring it into the ASCII range. Then the high nibble of the first byte is store in the second byte and the value 0x20 is added to it for the same reasons as above. The reason why the byte is brought into the ASCII range is to avoid any further Unicode conversion.

Example:



When generating code for ANSI, the `InvokeHelper` will convert the array to Unicode, so the byte array (string) is converted as described above, while when generating code for Unicode, the string is assumed to be Unicode already, so no conversion will take place in the helper. For the SKI ActiveX Control to be able to convert the array back again, a zero byte is added after every byte - simulating a Unicode conversion. Since all characters is in the ASCII range this will be right for all codepages.

How to use the class:

1. Insert following line in the MFC C++ -file:


```
#include "SkMfcCnv.h"
```
2. Create an instance of the class, either locally in the function where the above methods are used, or create it globally, e.g. as follows:


```
CskMfcCnv SkImgCnv( ScreenKeyType );
```
3. When using the methods `DisplayGraphics` or `DisplayTextOnGraphics`, insert the `Convert` method from this class, e.g. as follows:


```
m_ScreenKey.UpdateKey( Row, Col, TopText, BotText,
                        SkImgCnv.Convert( Image ),
                        Color, Attr );
```

The class will automatically free any memory used when its destructor is called. The class will automatically adapt to the string type used (ANSII or Unicode/WideChar)

KeyCodes property

A similar problem as described above also applies to the `GetKeyCodes` property. This property returns a BSTR array containing the scancodes defined for the pressed key. These scancodes are made up of two bytes, one containing the real scancode, while the other contains the ASCII value.

When using ANSI, two and two bytes, which represent one scancode, are converted to one ASCII character, and therefore some of the information is lost, and the result is meaningless.

To solve this problem, the array can be converted to a Unicode string by the control before the array is returned, so the wrapper class can convert it back to a CString. The CString will be twice as long as the number of scancodes, because this now is an array of single byte characters.

When using the SAC Engine Control from an ANSI MFC application, the flag **SAC_ANSI_KEY_CODES** must be given with the **StartSAC** method to have the `GetKeyCode` property convert to Unicode (wide characters) before returning the key codes.

Delphi 5

Delphi 5 supports the COM interface in its own process space, but not in a separate process space. Delphi 5 also supports IntelliSense, but not parameter enumeration. To find the parameter definitions, the programmer has to either look up this documentation or look in the file `SACEngine_TLB.pas`, which is generated by Delphi.

Installation in Delphi

To install the SAC Engine Control Interface in Delphi, do the following:

- From the *Component* menu select *Import ActiveX Control...*
- Select *SAC Engine Control Interface* from the list.
- The class will by default be *TSACEmulator*, and the palette will be *ActiveX*. This can be altered if desired.
- Click on the *Install* button to install the control.
- When the *Install* dialog appears, click *OK* to install. The *SAC Engine Control Interface* can be install to an existing package, or to new one.
- When the *Confirm* dialog appears click on *Yes*.

From this moment the *SAC Engine Control Interface* starts to exist and can be used as a regular Delphi component.

The SAC Engine Control icon shall now be available in the ActiveX tab (palette).

Using Control in project

The component is now ready for use. In the *Object Inspector* dialog standard *Properties* can be altered and *Events* handlers can be created.

Example of starting the SAC Engine when a button (Start) is clicked:

```
procedure TForm1.StartClick(Sender: TObject);
begin
    { Start the SAC Engine in COM mode }
    SACEngine1.StartSAC('COM1', SACFile, KbdFile, SAC_COM_MODE );

    { Check for any errors, display error message in error  message field }
    if SACEngine1.ErrorNumber <> E_NO_ERROR then
    begin
        { Error report here. }

        { Error description is held in SACEngine1.ErrorDescription property }
        ErrMsg.Caption := SACEngine1.ErrorDescription;
    end;
end;
```

Event handling

Handling of SAC Engine Control events is as simple as for any regular Delphi component.

In *Object Inspection* dialog choose tab *Events*, and then double click on the empty field on the right from event's description. Delphi will then generate an event handling procedure for this event.

Example of SAC Emulator Control *OnMenuChange* event handler.

```
procedure TForm1.SACEngine1MenuChange(Sender: TObject; var MenuName: WideString);
begin
  { Procedure's body, e.g.: }
  MessageDlg(MenuName, mtError, [mbOK],0);
end;
```

Error handling

Errors from SAC Engine Control can be handled on two ways:

- When SAC Engine Control is started with the flag `SAC_NO_EXCEPTION` (no exception mode), no exceptions will be generated and possible error code can be checked using the *ErrorNumber* property. Example of this is shown above.
- When SAC Engine Control is started without the `SAC_NO_EXCEPTION` flag (exception mode), the SAC Engine Control will generate exception when errors occur. This exception can be handled by using standard try...except statement.

Example of try...except statement.

```
procedure TForm1.StartClick(Sender: TObject);
begin
  try
    { Start the SAC Engine in COM mode }
    SACEngine1.StartSAC('COM1', SACFile, KbdFile, SAC_COM_MODE );
  except
    { Error number and description can be checked and reported here, e.g.: }
    on E: Exception do MessageDlg(E.Message, mtError, [mbOK],0);
  end;
end;
```

A P P E N D I X A

Documentation Control

A.1 Change Control

This document is the responsibility of the author and is subject to formal change control after the initial approved release (i.e. issue 1.0).

A.2 Abbreviations Used/Terms of Reference

SKI	ScreenKey Interface
ScreenKey	Registered Trademarked key-switch with a backlit LCD screen incorporated.
ScreenKey console	SKI's range of keyboards containing ScreenKeys.
SAC	ScreenKey Active Control. Name of SKI's menu file system.
ScreenKey ActiveX Control	Now called SKI ActiveX Control. This is a COM control that forms the low-level interface to a ScreenKey console.
SKI ActiveX Control	A new name for ScreenKey ActiveX Control
PASKeyboard	Former name of ScreenKey console
ActiveX	Name of technology used for communication between applications/modules.
KBD	KeyBoard Definition (file)
COM	Component Object Model. As ActiveX.
LCD	Liquid Crystal Display
BSTR	Visual Basic type string.
MSR	Magnetic Stripe Reader
POS	Point Of Sale - the cash register in a shop.
PWD	Superseded ScreenKey console software toolset.

A.3 Historical Change Reference

Issue	Date	Author	Changes Made
1.0	04/02/01	Bjorn Liane	Initial release
1.1	05/10/01	Bjorn Liane	Updated for version 1.1
1.2	16/03/04	M McDonnell	Updated for version 2.0
1.3	05/05/04	M McDonnell	Updated for version 2.1
1.4	28/01/05	M McDonnell	Updated for version 2.2
1.5	15/12/05	M McDonnell	Updated for version 3.0
1.6	30/11/06	M McDonnell	Updated for version 3.1

A.4 Change Summary

Issue	Change description
1.1	Added paragraph on how to solve problems with BSTR strings in MFC. Minor additions and corrections.
1.2	Added information on SK-7000 and dedicated methods for controlling ODA/TDA/CDA. Removed multi-application interface and COM wrapper information. Formatted doc to use letter page size.
1.3	Included information on interface handshaking setting in registry. Included reference to separate Wrapper module documentation.
1.4	Changed reference to latest version of SAC Engine Control (2.2)
1.5	Added references for LC24 and RGB ScreenKey support
1.6	Changed registry references to use ScreenKeys instead of RTI, removed MSR Utility references. UpdateKey method amended.