SKI ActiveX Control

Version 3.1

User's Guide

Issue 1.8 **March 2007**

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

Purpose

The purpose of this document is to explain how to use the SKI ActiveX Control. It describes the properties, methods and events, plus general information about the use of the control.

www.ScreenKeys.com

SKI ActiveX Control

Information in this document is subject to change without notice.

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

Web: www.ScreenKeys.com

Technical Support is available

via Email support@ScreenKeys.com

via Web: www.ScreenKeys.com

DISCLAIMER:

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

Foreword

The SKI ActiveX Control is a COM-control to be used to interface an application written in a COM-enabled programming language with a ScreenKey console, using a development environment utilizing the qualities of COM (ActiveX) controls, such as IntelliSense® and parameter enumeration. It is designed to work with languages like MS Visual Basic and MS Visual C++, using MS Visual Studio 6.0.

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 (*SKI ActiveX Wrapper User's Guide.pdf*).

Previous versions of the SKI ActiveX Control supported a multiple application interface. This interface is discontinued release version 2.0 onwards. This feature was not beneficial to users and so was removed to make the control more compact and robust.

This document will describe how to use the control from the most popular programming languages.

SKI – **S**creen**K**ey **I**nterface

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 consoles. 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.

SKI implements a "SAC" (ScreenKey Active Control) approach to the control of a ScreenKey console. SAC files define how to control a ScreenKey console and include text, graphics, menus, menu navigations and keystroke return values. The SAC Engine and SAC Controller allow developers to integrate a ScreenKey console with minimal programming effort using SAC files.

Alternatively, the SKI ActiveX Control (described in this document) provides developers with the ability to directly control each key on a ScreenKey console. This is suitable for a user interface that does not change significantly and/or ScreenKeys that have to display dynamic data, e.g. equipment temperature or an alarm condition.

Table of Contents

SKI ACTIVEX CONTROL OVERVIEW 7

What is ScreenKey Console ? 7
What is SKI ActiveX Control ? 7
Who should read this document ? 7
Requirements 8
Cross reference 8

INSTALLATION PROCEDURE 9

Hardware Installation 9
Software Installation 9
Uninstallation of Software 10
Redistributable Components / Files 11
Console Interface (Handshake) Specification 12
Downloadable Console Firmware 12
DCOM & Windows 95 12

TECHNICAL DESIGN 13

SKI ActiveX Control 13
Single Application ActiveX Interface 14
Direct ActiveX Control Access 15

USAGE 16

ScreenKey display 16
Text 16
What if the text doesn't fit? 17
LC16 Graphics 18
LC24 Graphics 18
Text and Graphics Combined 19
SimpleText 19
International character sets 19
Key Presses 20
Single Key Presses 20
KeyDown/KeyUp events 20
Repeat keys 21
Key return 21
Key beeps 21
Disable ScreenKey 21
KeyLock turns 22
MSR swipes 22
Methods, Properties & Events 23
Categories 24

METHODS 27

OpenKeyboard 27
CloseKeyboard 29

DefineLargeKeytop *29*
RedefineKey *30*
DisplayText *32*
DisplayGraphics *33*
DisplayTextOnGraphics *34*
DisplaySimpleText *35*
DisplayText24 ..... *36*
DisplayGraphics24 ..... *37*
DisplayTextOnGraphics24 ..... *38*
SetLED *39*
SetKeyAttribute *40*
Sound *41*
ConfigSleep *41*
ConfigFlash *41*
ConfigRepeat *42*
SetCharSet *44*
SendSimpleCommand *45*
WriteTextODA *46*
WriteTextTDA *47*
WriteTextCDA *48*
SelectCDA *49*
SelectTDA *49*
WriteGraphicTDA *50*
WriteTextODA *51*

PROPERTIES 52

ErrorMode *52*
ErrorNumber *53*
ErrorDescription *53*
KeyPress *53*
KeyPressRow *54*
KeyPressCol *54*
KeyLockPos *54*
KbdLayout *55*
KeyType *55*
MSRTracks *56*
MSRTrack1Data *56*
MSRTrack2Data *56*
MSRTrack3Data *56*

EVENTS 57

KeyPressEvent *57*
KeyLockTurnEvent *58*
MSRSwipeEvent *58*
ErrorEvent *58*

APPENDICES

KEY NUMBERS AND NUMBERING DIAGRAMS 60

The SK-2000 Console 60
The SK-5400 Console 62

The SK-6000 Console 62
The SK-7000 Console *63*

SCREENKEY COLORS **64**

SCREENKEY ATTRIBUTES **66**

ERROR HANDLING **67**

Console errors and MSR errors *68*
Error numbers and messages *70*

TROUBLESHOOTING **73**

PC errors (SkAxCtl) 73
ScreenKey console errors 84

LANGUAGE INTERFACES **93**

Visual Basic 93
Visual C++ 95
Using MFC 96
Delphi 5 98

DOCUMENTATION CONTROL **99**

A.1 Change Control *99*
A.2 Abbreviations Used/Terms of Reference *99*
A.3 Historical Change Reference *99*
A.4 Change Summary *100*

I N T R O D U C T I O N

SKI ActiveX Control Overview

What is ScreenKey Console ?

The ScreenKey console is an Input/Output device that is suitable for use in many diverse markets and specialized applications, such as process control, point-of-sale, 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 two different resolutions: 32 pixels across by 16 pixels down (LC16) and 36 pixels across by 24 pixels down (LC24) and with two different LED backlighting schemes: RG (red-green) and RGB (red-green-blue).

The ScreenKey console connects to the PC via the Serial RS232 port.

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 like Visual C++, Visual Basic etc. 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 integrate the ScreenKey console with an application. 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, with Microsoft Windows and Windows commands.

Requirements

Most applications should be suitable for use with SKI ActiveX Control 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 SKI ActiveX Control does not require any additional memory than that required for running the operating system.

Cross reference

[1] ScreenKey Getting Started & Installation Guide

[2] ScreenKey Console Technical Reference Manual

[3] ScreenKey Low-Level Interface Programmer's Guide

[4] SKI ActiveX Wrapper User's Guide

I N S T A L L A T I O N

Installation Procedure

Hardware Installation

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

Software Installation

The SKI ActiveX 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 SKI ActiveX Control sub-folder using Windows explorer from the root of the CD and run Setup.exe from this folder.

Start the **Setup.exe** program, and follow the on-screen instructions.

- 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,** but this may be changed by clicking on the Browse button. 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.
- When the control has been installed, it (three files) will register in the registry. If registration goes well, the user has to click OK for the files SkAxCtlps.dll and SkAxCtl.dll (problems registering SkAxCtl.exe will be reported by the Setup).

- If errors occur at this stage, i.e. the registering did not succeed, the Control will not be functioning. In this case, try the following:

  1. Uninstall earlier versions of the control, 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 supplied, Register.bat, which can be run from a DOS shell when errors have been corrected.

- After registration of the 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 SKI ActiveX Control do the following:

1. Terminate any application using 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 **SKI ActiveX Control** (or ScreenKey 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 SKI ActiveX Control an all of its components.
6. The SKI ActiveX 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.
7. If parts of the control could not be removed, a button named Details… will be visible, and the user can 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 SKI ActiveX Control in an application some files have to be redistributed with the application. They should all reside in the same folder to avoid problems. The application may be distributed with the following files:

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. This file is necessary if SkAxCtl.dll is in use.	YES
SkAxCtl.dll	SKI ActiveX 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
*.skf	ScreenKey Font files. These files are distributed for CodePages other than 1252. They should be distributed with the control, residing in the same directory as SkAxCtl.exe.	N/A

Some of the EXE and DLL files have to be registered in the registry. SkAxCtl.exe can register itself, 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
```

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.

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

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.

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 SKI ActiveX Control 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\PLL4_2.bin" for products such as the OEM-5400

or

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

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 SKI Installation CD.

Technical Design

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

- SKI ActiveX Control Interface
- SKI ActiveX Control

The **SKI ActiveX Control Interface** runs in the application's own process space, and therefore is accessible from Visual Basic. This is a DLL file, called **SkAxCtl.dll**.

The **SKI ActiveX Control** itself runs in a separate process space, and therefore cannot be accessed directly from Visual Basic. It is accessible from C++ when it is created as an object. The control is an EXE file, called **SkAxCtl.exe**.

A Proxy Stub DLL is also necessary, this is called **SkAxCtlps.dll**.

SKI ActiveX Control

The SKI ActiveX Control consists of two modules, a client and a server. There may only be one instance each of the client and the server. The server handles the actual interface to the ScreenKey console.

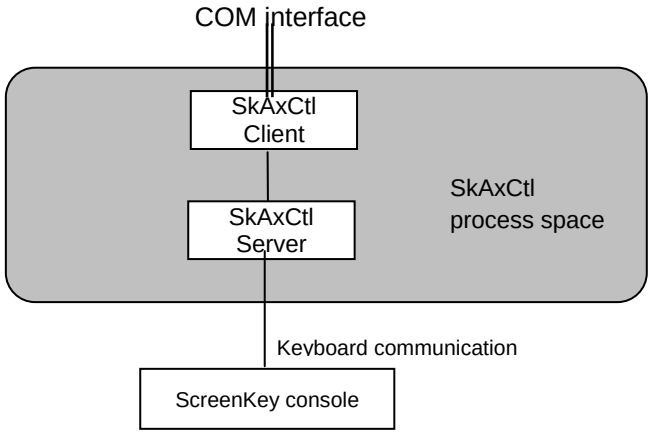


Figure 1 – SKI ActiveX Control

The use of a client and a server was intended to provide the opportunity to manage multiple applications interfacing to the same ScreenKey console, i.e. multiple SkAxCtl clients interfacing to one SkAxCtl server. However, this feature was removed in release 2.0 as it proved excessive to user requirements and increased the overall complexity and footprint of the SKI ActiveX Control unnecessarily.

Single Application ActiveX Interface

The control can only be used by a single application. What must be considered is if the programming language allows use of EXE controls or not. Visual Basic does not allow this, and must therefore use the SKI ActiveX Control Interface, the DLL file. This introduces an extra interface, and will have some performance hit to the system.

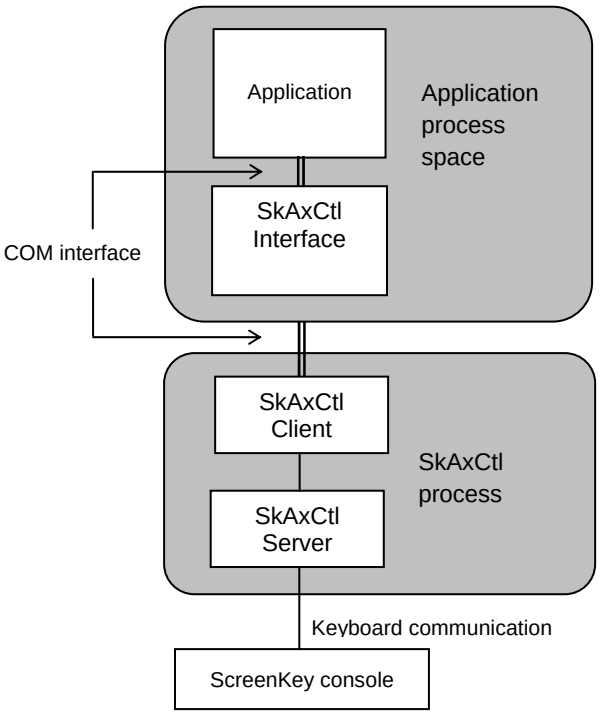


Figure 2 – Single Application with ActiveX Control Interface used from e.g. Visual Basic

An application, written e.g. in C++, can also access the SKI ActiveX Control directly. See Direct ActiveX Control Access, page 15.

Direct ActiveX Control Access

From e.g. C++ (Visual C++ etc.) it is also possible to access the SKI ActiveX Control Client/Server directly by creating an object "on the fly" (not dropping the object onto a form), since C++ allows accessing COM objects in a different process space. This way it is possible to bypass the SkAxCtl Client Interface, and therefore get better performance, but this approach takes more coding.

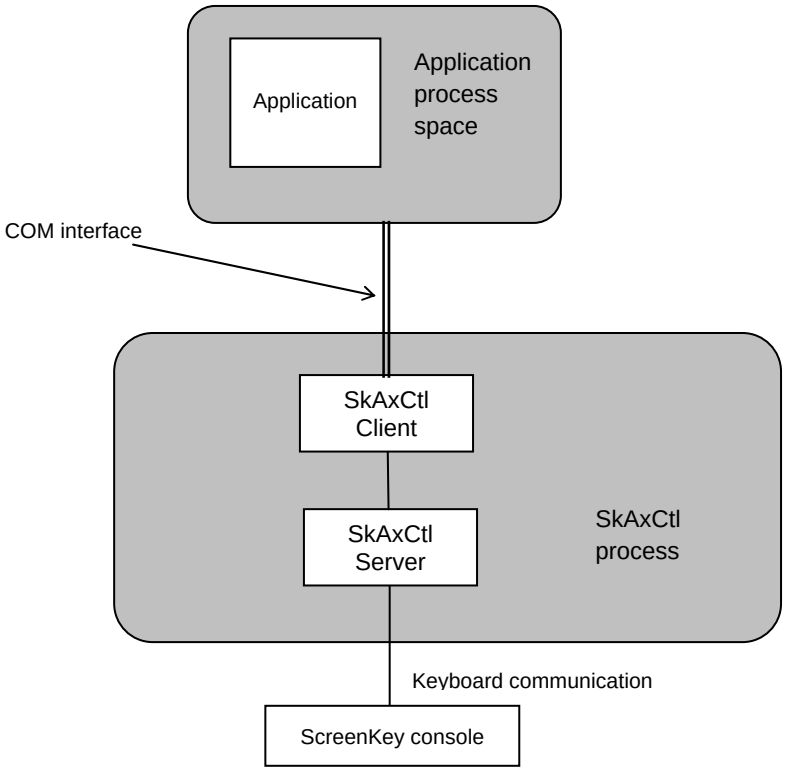


Figure 3 – Direct control access, from e.g. C++

NOTE:

Be aware that when using the SKI ActiveX Control (SkAxCtl.exe) directly, all the enumeration definitions are named as for the SKI ActiveX Control Interface, but prefixed with "S_" (EXCLUSIVE_USE will be S_EXCLUSIVE_USE).

U S A G E

Usage

The SKI ActiveX Control has different categories of properties, methods and events, and some of these have to be executed in a certain sequence. The detailed description of each method, property and event will be described later.

ScreenKey display

The main advantage of the ScreenKey console is the ability to display information on the key. A ScreenKey can display text, graphics or a combination of text and graphics.

Text

ScreenKey consoles include a default 7x5 pixel font. With this inbuilt mono-spaced font it is possible to display two lines of five characters on an LC16 ScreenKey and three lines of six characters on an LC24.

The SKI ActiveX Control offers a feature that can display more than the standard default number of characters per line, with text adjustment control, by converting the text to graphical images before sending them to the console. The methods using this feature are; **DisplayText** and **DisplayTextOnGraphics** for LC16 and **DisplayText24** and **DisplayTextOnGraphics24**. These methods are self-contained, and don't need any properties to be set.

When more than the default number of characters are attempted to be displayed on a line, narrower fonts are used, and the gaps between the characters are removed (similar to proportional spaced printing). This way it is possible to display up to approximately 10 characters, although normally around eight, on each line, depending on the nature of the characters.

When the default number of characters or less is displayed, the mono-spaced font is used by default, unless the attribute CONDENSED is specified. If this is the case, the second widest (where this is a good representation of the character) is chosen, and all gaps are removed, giving the text a proportional look.

The DisplayText(24) and DisplayTextOnGraphics(24) methods also offer a range of text justification, the text can be justified both horizontally and vertically. Available text justification attributes, which are parts of the DISP_ATTRIBUTE enumeration structure, are:

Definition	Val	Description
NO_JUSTIFY	00h	The same as HOR_LEFT if not combined with CONDENCED
HOR_LEFT	01h	The text (both lines) is left adjusted.
HOR_CENTRE	02h	The text is centered horizontally on the key.
HOR_RIGHT	04h	The text is aligned to the right.
VER_TOP	08h	If only one line of text is specified, this is displayed at the topmost line.
VER_CENTRE	10h	If only one line of text is specified, this is displayed vertically centered on the key.
VER_TOP	20h	If only one line of text is specified, this is displayed at the bottom line.
CONDENSED	40h	The second widest font is chosen, and all gaps (one pixel is left) are removed. This applies only to texts containing five or less character on one line.

What if the text doesn't fit?

If the specified text doesn't fit onto a ScreenKey, there are a few things that can be done:

FAIL_TEXT_TOO_LONG

One can either choose to display the text anyway (clipped), or not to display it, and get an error instead. This mode, FAIL_TEXT_TOO_LONG, is set globally with the OpenKeyboard method, or as an attribute (parts of DISP_ATTRIBUTE) with the methods DisplayText and DisplayTextOnGraphics. These can be FAIL_TOO_LONG or NOFAIL_TOO_LONG, and will override what is specified with the OpenKeyboard method. The default setting for the OpenKeyboard method is NOT to report a too long text as an error.

Using the FAIL_TEXT_TOO_LONG mode, it is possible to loop through an algorithm, e.g. removing the last character or vowels after each failure, until the text fits on the key.

```
OpenKeyboard "COM1", KBD_TYPE_6000, EXCLUSIVE_USE or FAIL_TEXT_TOO_LONG
```

UpperCase/LowerCase

In general, a combination of uppercase and lowercase letters, the first in uppercase and the rest in lowercase, is the easiest to read. A problem is that the lowercase letters are normally more detailed and therefore harder to "squeeze" into the key. It could therefore be a good idea to use UpperCase characters.

Abbreviations/Consonants

Another trick is to use mostly consonants, a form of abbreviation. A word consisting only (or mostly) of consonants is easier to understand than the first part of a longer word.

Pronunciation

Sometimes it can be effective to Display the pronunciation of the work instead of the actual spelling.

LC16 Graphics

The method DisplayGraphics is used to display a graphical image on a LC16 ScreenKey. The resolution of the key is 32 pixels horizontally, and 16 pixels vertically, organized from the upper left corner, with four bytes across. The image is a BSTR string, 64 characters long. The BSTR string is normally used to hold Unicode string, but it is used with these methods because it is handy to hold a byte array. The graphical format is as follows:

Byte	Col	Byte n+0								Byte n+1								Byte n+2								Byte n+3							
	Row	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0-3	0																																
4-7	1																																
8-11	2																																
…	…																																
60-63	15																																

Note for VB users:
In Visual Basic, a BYTE array, 64 bytes long, should be used instead of a 64 long BSTR (Unicode string). This is because if the application is running on a platform with a codepage other than 1252, the characters (Unicode characters) can be converted to zeros if the 7th bit is set. VB users should use a design as described below:

```
Dim Image(63) As Byte   ‘ Define as a 64 byte array
For I = 0 To 63
    Image(i) = 85       ‘ Every other pixel set
Next I
ScreenKey.DisplayGraphics Image, 0, 0, BRIGHT_RED
```

Note: MFC does not support BSTR type, and a special class is available for avoiding problems with MFC applications. See “Using MFC” on page 96

LC24 Graphics

The method DisplayGraphics24 is used to display a graphical image on a LC24 ScreenKey. The resolution of the key is 36 pixels horizontally, and 24 pixels vertically, organized from the upper left corner, with five bytes across. The lower nibble of the fifth byte is not used. The image is a BSTR string, 120 characters long. The graphical format is as follows:

Byte	Col	Byte n+0								Byte n+1								Byte n+2								Byte n+3								Byte n+4							
	Row	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0-4	0																																								
5-9	1																																								
10-14	2																																								
…	…																																								
115-119	24																																								

Bits 3-0 of Byte n+4 are not used

Text and Graphics Combined

The methods DisplayTextOnGraphics (and DisplayTextOnGraphics24) is used to display text mapped onto a graphical image on a ScreenKey. The graphical image is the same as for DisplayGraphics (or DisplayGraphics24), and the text attributes are the same as for DisplayText (or DisplayText24). The INVERT_ , FLIP_ and EXCLUSIVE_OR_MAP functions are performed after the text is mapped into the graphical image. The EXCLUSIVE_OR_MAP flag will invert the text if the background pixels are set, so it will be readable. One has to be careful though, because if the image where a character displayed is a mix between set and unset pixels, the character can still be very difficult to read.

SimpleText

DisplaySimpleText is an obsolete method offered for backward compatibility. Users should use DisplayText instead. This method is used to display mono-spaced characters on an LC16 ScreenKey only. It does not support LC24. It takes a string up to 10 characters long. The first five will be displayed on the topmost line, and the rest on the bottom line. The ScreenKey colors have to be specified using the KeyColor property, and the attributes using the Attribute property. The attribute ATTRIB_CENTRE only applies when a string of five or less characters is specified. Then the text is centered both horizontally and vertically.

International character sets

The SKI ActiveX Control uses Unicode strings, and is therefore able to display international characters. Currently code-page 1252 is supported, which covers most of the European and American languages, but other character sets may be designed on demand. The application designer may also design a custom character set. The character set format may be obtained by contacting SKI. Tools for designing character sets will also be available in the future.

If an application using the SKI ActiveX Control is running on a computer with a different codepage, the control will look for a character set file containing ScreenKey characters for this codepage. If it is not found it will not be able to initialize the console and therefore return an error. The file name has the following format: “SK_?.SKF”, where ? is the codepage number (e.g. SK_932.SKF for Japanese codepage 932).

In some cases one would want the SKI ActiveX Control to use the Western European codepage (1252) independent of the codepage installed on the computer. To do this the control has to be set to use this codepage before attempting to open the console, using the SetCharSet method, see page 44. This method will then pass the specified character set to the OpenKeyboard method, and will be attempted to be loaded when this is invoked. Any defined codepage may be used at this stage, as long as their corresponding font files exist in the ScreenKey Bin folder.

If the flag **NO_FAIL_ON_CODEPAGE** is given as a parameter to the **OpenKeyboard** method, the control will try to load the character set given by the codepage retrieved from the operating system. If this is not found, it will load the built-in character set, which is 1252.

If the application has not implemented either of the above-mentioned approaches, and it is run on a computer with a codepage that is not supported, it will fail when attempting to initialize the console. To work around this problem, the file SK-1252.SKF, which is included in the install, can be renamed to the currently loaded codepage (e.g. to SK_932.SKF). This will trick the control to think it is working with the current codepage, while in fact it will use the 1252 codepage character set for the ScreenKeys. This approach should only be used if the application cannot be changed to use the appropriate approach.

Key Presses

The application will be notified of key presses through the **KeyPressEvent** event handler. Every time a key is pressed on the ScreenKey console; this event handler is called. The event handler has one parameter; the key number of the key pressed. The key number received is according to the figures in the chapter 'Key numbers and Numbering diagrams', unless the key numbers are redefined using the RedefineKey method (See RedefineKey method, page 30). The event is normally sent when the key is pressed, but the console can be configured to also send an event when the key is released. A key can also be configured to send repeated key presses as long as it is being held pressed, and the repeat rate can be configured.

When a key press event occurs the properties KeyPress, KeyPressRow and KeyPressCol will be updated, and they may be accessed from the event handler or from another function at a later stage. Be aware that if another key press event occurs before the previous event is handled in full (outside of the event handler), these properties will be overwritten with the new values. Therefore a key press event should be handled immediately, and if that is not possible, the application should store the key presses locally for later handling.

Single Key Presses

The default operation of a key press is a single key press event sent when the key is pressed, and no event when it is released. There is no special handling necessary in this mode.

KeyDown/KeyUp events

A ScreenKey console can be configured to send events on both **key down** and **key up** globally. This is done by using the method SendSimpleCommand with the parameter ENABLE_KEY_OPENS:

From Visual basic:
ScreenKey.**SendSimpleCommand ENABLE_KEY_OPENS**

The console will then send an event when the key is pressed, with the corresponding key number as parameter. When the key is released, the console will send another event with the same key number, but this time with **bit 31 set to one** (1) as an indication that it is an **up** event. The KeyPressEvent handler in the application must mask out this bit to get the right key number if it is using the parameter.

The properties KeyPress, KeyPressRow and KeyPressCol will hold the key number without the Up-bit set, so these can be used to get hold of the right key number, or row and column numbers.

If repeat is enabled and the key is held pressed for a certain time, the event handler will receive multiple key down events, and one key up event (with the up-bit set) when it is released.

The ENABLE_KEY_OPENS does not apply to the KeyLock switch (KeyLockTurnEvent). These "key presses" are sent from the console as ordinary key presses, both Close and Open (Down and Up), but are filtered in the SKI ActiveX Control so only the new positions are sent to the applications as events.

Repeat keys

Any key on a ScreenKey console can be configured individually to send multiple key press events when it is held pressed (typematic). This can be configured using the method SetKeyAttribute.

From Visual basic:

```
ScreenKey.SetKeyAttribute 0, 0, ENABLE_KEY_REPEAT
```

which configures the key in row=0, col=0 to send multiple key press events (down) when it is held pressed.

The rate can be adjusted globally by using the ConfigRepeat method:

From Visual basic:

```
ScreenKey.ConfigRepeat 500, 16
```

which configures all keys to delay for 500 ms before starting repeating, and then sends a key press event at a rate of 10 characters a second (SpeedVal=16).

Key return

Any key on a ScreenKey console can be configured individually to send a key press event or not when it is pressed. This can be configured using the method SetKeyAttribute, default is key return enabled. Disabling the key return will make a key inactive.

From Visual basic:

```
ScreenKey.SetKeyAttribute 0, 0, DISABLE_KEY_RETURN
```

which configures the key in row=0, col=0 NOT to send a key press event when it is pressed.

Key beeps

Any key on a ScreenKey console can be configured individually to give an audio indication (key beep) or not when it is pressed. This can be configured using the method SetKeyAttribute, default is key beep enabled.

From Visual basic:

```
ScreenKey.SetKeyAttribute 0, 0, DISABLE_KEY_BEEP
```

which configures the key in row=0, col=0 NOT to beep when it is pressed.

Disable ScreenKey

The parameter DISABLE_SCREENKEY with the SetKeyAttribute method will disable a particular ScreenKey switch, but only for updating the LCD screen in the key. It does NOT stop it from sending an event when it is pressed.

KeyLock turns

The application will be notified of KeyLock turns through the **KeyLockTurnEvent** event handler, when the KeyLock is turned on the ScreenKey console. The event handler has one parameter, KeyPos, which can be 0, 1, 2, 3 or 4, unless the key numbers (positions) are redefined using the RedefineKey method (See RedefineKey method, page 30).

When a KeyLock turn event occurs the property KeyLockPos will be updated, and it may be accessed from the event handler or from another function at a later stage. Be aware that if another KeyLock turn event occurs before the previous event is handled in full (outside of the event handler), these property will be overwritten with the new value. Therefore a KeyLockTurn event should be handled immediately, and if that is not possible, the application should store the key position locally for later handling.

The KeyLock does not send any Close or Open events, only the new position of the KeyLock.

MSR swipes

A ScreenKey console may be equipped with a MSR device (Magnetic Stripe Reader). When a magnetic card is swiped, an event is sent to the application through the **MSRSwipeEvent** event handler. The event handler has one parameter, NoTracks, which tells the application how many tracks were read. Which tracks that were read can be obtained using the property **MSRTracks**, and the track contents can be obtained using the properties **MSRTrack1Data**, **MSRTrack2Data** and **MSRTrack3Data**.

Certain tracks can be made mandatory using the method SendSimpleCommand with the parameters:
T1_MANDATORY, T2_MANDATORY, T3_MANDATORY, T1_T2_MANDATORY, T2_T3_MANDATORY, T1_T3_MANDATORY or T1_T2_T3_MANDATORY.

If a good card is swiped with the track defined as mandatory is swiped, a “good card beep” (a short beep) will be issued, and the event sent to the application. The application may then obtain the data through the properties mentioned above.

If a bad card is swiped, a card containing an unsupported format, or a card without the track defined as mandatory, a “bad card beep” (a longer beep) will be issued, and NO event will be sent to the application. If the console has been configured to return MSR errors (via the OpenKeyboard method) an error message will be sent to the application. Other types of MSR errors will also be sent to the application if this option is chosen.

The data in track-data is converted to readable ASCII characters after it’s read by the console, before it is presented in the MSRTrack?Data properties. To the track 1 data it is added 20 hex, and to track 2 and 3 data it is added 30 hex to get readable ASCII data.

The MSR device in the ScreenKey console can only read tracks that follow the Visa standard (number of bits per byte, start and end sentinel, and the LRC code), while the layout of the data is not essential (separators etc.). It is up to the application to interpret the data right.

Methods, Properties & Events

The methods, properties and events are the following:

Methods	Properties	Events
OpenKeyboard	KbdLayout	KeyPressEvent
CloseKeyboard	ErrorMode	KeyLockTurnEvent
DefineLargeKeytop	ErrorNumber	MSRSwipeEvent
RedefineKey	ErrorDescription	ErrorEvent
ConfigSleep	KeyPress	
ConfigFlash	KeyPressRow	
ConfigScroll	KeyPressCol	
ConfigRepeat	KeyLockPos	
SetKeyAttribute	MSRTracks	
SetCharSet	MSRTrack1Data	
DisplayText	MSRTrack2Data	
DisplayGraphics	MSRTrack3Data	
DisplayTextOnGraphics	KeyType	
DisplaySimpleText		
DisplayText24		
DisplayGraphics24		
DisplayTextOnGraphics24		
SetLED		
Sound		
SendSimpleCommand		
RequestKbdData		
WriteTextODA		
WriteTextTDA		
WriteTextCDA		
SelectCDA		
SelectTDA		
WriteGraphicTDA		

Categories

The interface listed per category, Methods are marked (m), properties (p), Events (e) and ordinary functions (f):

Initialization	
OpenKeyboard (m)	Opens / Initializes the ScreenKey console
KbdLayout (p)	Holds the ScreenKey console key layout after being initialized.
CloseKeyboard (m)	Closes the ScreenKey console
KeyType (p)	Holds the type of ScreenKeys currently attached to the console, i.e. resolution and LED colors

Setup	
DefineLargeKeytop (m)	Defines a large keytop.
RedefineKey (m)	Redefines a key's key number and the number returned when pressed
ConfigSleep (m)	Defines the Sleep mode, enable, disable and sets the sleep timeout.
ConfigFlash (m)	Defines the flashing sequence.
ConfigScroll (m)	Defines the scrolling speed and delay before scrolling starts. *This method only works with OEM firmware versions 4.3 or earlier. It is **not** supported by SK-7000 firmware versions.*
ConfigRepeat (m)	Defines the repeat rate and the delay before repeat starts for a key.
SetKeyAttribute (m)	Defines key attributes like key beep, return, repeat etc.
ErrorMode (p)	Defines the error mode, how control shall notify application about errors occurred.
SetCharSet (m)	Sets the requested code-page, and/or downloads a user-defined character set to the console.

Errors & Debugging	
ErrorMode (p)	Defines the error mode, how control shall notify application about errors occurred.
ErrorEvent (e)	Notifies an application that an error has occurred.
ErrorNumber (p)	Holds the error code from the most recent method executed.
ErrorDescription (p)	Holds the error message from the most recent method executed.
RequestKbdData (m)	Requests a code/data dump from the console.
SendSimpleCommand (m)	Method to send simple commands to the console. These are typically rarely used commands.

Key activity	
KeyPressEvent (e)	Notifies an application that a key press has arrived.
KeyPress (p)	Holds the latest pressed key's number .
KeyPressRow (p)	Holds the latest pressed key's row number.
KeyPressCol (p)	Holds the latest pressed key's column number.
KeyLockTurnEvent (e)	Notifies an application that the KeyLock switch has been turned.
KeyLockPos (p)	Holds the current KeyLock position.

ScreenKey	
DisplayText (m)	Sends text to be displayed on a LC16 ScreenKey.
DisplayText24 (m)	Sends text to be displayed on a LC24 ScreenKey.
DisplayTextOnGraphics (m)	Sends text and graphical image to mapped together and displayed on a LC16 ScreenKey.
DisplayTextOnGraphics 24 (m)	Sends text and graphical image to mapped together and displayed on a LC24 ScreenKey.
DisplayGraphics (m)	Sends graphical image to be displayed on a LC16 ScreenKey.
DisplayGraphics24 (m)	Sends graphical image to be displayed on a LC24 ScreenKey.
DisplaySimpleText (m)	Displays a text string on a ScreenKey, this method is for backward compatibility only.
SetLED (m)	Turns LEDs ON or OFF. ScreenKey-6000 only.
Sound (m)	Request the console to issue a beep.

Magnetic Stripe Reader	
MSRSwipeEvent (e)	Notifies an application that a Magnetic Card has been successfully swept, and how many tracks have been read.
MSRTracks (p)	Holds a description of which tracks have been read.
MSRTrack1Data (p)	Holds the data of track 1
MSRTrack2Data (p)	Holds the data of track 2
MSRTrack3Data (p)	Holds the data of track 3

SK-7000 Integrated Displays	
WriteTextODA (m)	Displays a text string on a line of the ODA
WriteTextTDA (m)	Appends a new text line to a TDA page buffer
WriteTextCDA (m)	Displays a text string on the CDA and turns the CDA on if not already displayed
SelectCDA (m)	Turns the CDA on or off
SelectTDA (m) [1]	Selects a TDA page buffer to be displayed
WriteGraphicTDA (m) [1]	Displays a graphic on the TDA

1 These methods are not implemented in the current release of the SKI ActiveX Control (version 2.4 at time of writing).

CONTROL METHODS

Methods

OpenKeyboard

Description

Call this method to initialize the control and console. If successful, the control is ready to communicate with the console, and the application can start sending commands to the console and receiving key presses etc.

The method opens the communication port, retrieves the console layout from the console and initiates the key translation tables for the specific console.

If the COM server is already running (in it's own process space) under the control of another application (e.g. that application has crashed or did not call the CloseKeyboard method on exit) then this method will take control of the COM server and the console.

Parameters

CommPort
Type: BSTR.
Name of comport to which ScreenKey console is connected, "COM1", "COM2" etc.

KeyBoardModel
Type: KBD_TYPE.
Type of console connected, the following choices are available:

Definition	Val	Description
KBD_TYPE_3000	3	SKI ScreenKey 3000 series
KBD_TYPE_5400	5	SKI ScreenKey 5400 series
KBD_TYPE_7000	7	SKI ScreenKey 7000 series

Exclusive
Type: KBD_OPEN_MODE
Default: EXCLUSIVE_USE

The following modes are available:

Mode Definition	Val
EXCLUSIVE_USE	0000h
FAIL_TEXT_TOO_LONG	0008h
NO_KBD_ERRORS	0010h
RETURN_MSR_ERRORS	0020h
NO_KBD_RESET	0040h
NO_FAIL_ON_CODEPAGE	0100h

EXCLUSIVE_USE is the default mode as the console can only be controlled from this application.

FAIL_TEXT_TOO_LONG means that an error will be generated, and the text not sent to the console, if the text specified with DisplayText or DisplayTextOnGraphics does not fit on the ScreenKey. If this mode is not set, the text will be truncated and sent to the console. This global attribute can be overridden by attributes specified with the DisplayText and DisplayTextOnGraphics methods.

NO_KBD_ERRORS means errors occurring in the ScreenKey console will not be sent to the application.

RETURN_MSR_ERRORS means that MSR (Magnetic Stripe Reader) errors will be sent to the application. By default, if a card could not be read, the console issues a beep to notify the operator that the swipe was not successful, and does not send the error message to the application.

NO_KBD_RESET means the control does NOT send a RESET to the console when opened/initiated. *This mode should be used with great caution, and is meant used only during the development phase.*

NO_FAIL_ON_CODEPAGE means the control will read the codepage number from the operating system. If this is found, it will use it, but if it is not found, it will use the built-in code page, which is 1252. Normally the OpenKeyboard method will fail if the codepage does not exist. Note that the codepage can be set prior to the OpenKeyboard method if a specific character set is desired.

Return Value Type: long

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

Use after: none

Remarks

After a successful initialization of the console the control updates a property called KbdLayout which specifies the console layout. The property KeyType is also updated with the type of ScreenKeys attached to the console, i.e. LC16 or LC24 resolution and whether the keys are RG or RGB type.

CloseKeyboard

Description

Call this method to end communication with the console. If successful, the control will not be able to send commands to the console or receive events from it.

Parameters

None

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

This method should be called before exiting the application. Otherwise the COM server will continue to run in its own process space taking up unnecessary resources.

DefineLargeKeytop

Description

Call this method to inform the console of large keytops.

The console may be equipped with large keytop, either double keys, vertical or horizontal, and/or quad keys (square). To prevent the console from sending key presses for all keys pressed, the console needs to be informed about these keys. The console will then only send the upper left key to the application.

Parameters

Row
Type: short
Row number of the upper left key of the large key.

Column
Type: short
Column number of the upper left key of the large key.

Height
Type: short
The height of the large keytop. To define a quad key Height will be set to 2.

Width
Type: short
The width of the large keytop. To define a quad key Width will be set to 2.

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

The Row and Column can be redefined key numbers.

At present the console does not detect other than the upper left key (the others are disabled), but will be updated to detect any key press and then send only the upper left key.

RedefineKey

Description

Call this method to redefine a key's number. The number used when displaying text on a key or sending other commands to it, may be redefined, so can the number returned when a key is pressed or KeyLock turned. These do not have to be the same numbers, (originally they are). See remarks section and page 60 for more information.

The redefined key numbers are local to the application, and do therefore not affect other applications using the same console at the time (shared).

Parameters

Row
Type: short
Original Row number of the key (zero based).

Column
Type: short
Original Column number of the key (zero based).

NewRow
Type: short
Optional, default is the value of Row.
New Row number of the key (zero based). In the range 0 to 7.

NewColumn
Type: short
Optional, default is the value Column.
New Column number of the key (zero based).
In the range 0 to 15.

NewKeyNo
Type: long
Optional, default is old value.
The new key number given in the KeyPressEvent and KeyLockTurnEvent event procedures.

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

The console does not have to be claimed or owned when using this method.

Original Row/Column are the numbers described in the *Numbering diagrams* paragraph. This numbering system will not be affected when a key is redefined, it may be redefined again using the same key numbers.

New Row/Column are any numbers in the range 0-65536.

With this method one can design ones own virtual numbering system, to suit any purpose. It may be used when an application shall handle ScreenKey consoles with different key panel layout, to make the code more generic.

NewKeyNo is the key number returned when a key is pressed or the KeyLock is turned. The NewKeyNo does NOT have to be unique, multiple keys may have the same key number, e.g. if they have the same functionality.

Row and column numbers may also be specified by setting row to zero and column to "Row * 16 + Column", as long as it does not exceed 65535.

RedefineKey can be used to make this ActiveX Control backwards compatible (can be defined to use the "old" numbering scheme).

DisplayText

Description

This method displays text on upper and or lower line of the specified LC16 ScreenKey with a specified color, flash color, text justification and bitmap manipulation.

Parameters

Line1
Type BSTR
String to display on upper line of specified ScreenKey.

Line2
Type BSTR
String to display on lower line of specified ScreenKey.

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

Color
Type: KEY_COLORS
Optional, default: BRIGHT_GREEN
Specifies color and flash color of ScreenKey.
The background colors and flashing colors has to be or'ed together to form a complete color specification. See page 64 more information.

DispAttrib
Type: DISP_ATTRIBUTE
Optional, default: NO_JUSTIFY
Specifies text attributes. All available attributes apply to this method.
See page 66 more information.

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

The Row and Column cam be calculated to a key number as follows:

Row * 16 + Column

The functions using row and column specifications can take Row=0 and Column=KeyNumber (Row * 16 + Column). This means the key number received through the KeyPressEvent can be used directly as Column if Row is set to zero.

DisplayGraphics

Description

This method displays Graphics on the specified LC16 ScreenKey. A predefined image can be created and displayed on the key.

Parameters

GraphicData
Type: BSTR
Graphic Data to display on the specified key

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

Color
Type: KEY_COLORS
Optional, default: BRIGHT_GREEN
Specifies color and flash color of ScreenKey.
The background colors and flashing colors has to be or'ed together to form a complete color specification. See page 64 more information.

DispAttrib
Type: DISP_ATTRIBUTE
Optional, default: NO_JUSTIFY
Specifies text attributes. Only some attributes apply to this method.
See page 66 more information.

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

The Row and Column cam be calculated to a key number as follows:

Row * 16 + Column

The functions using row and column specifications can take Row=0 and Column=KeyNumber (Row * 16 + Column). This means the key number received through the KeyPressEvent can be used directly as Column if Row is set to zero.

A BYTE array of 64 bytes should be used to hold the graphical image.
VB users: see page 18. MFC Users: see page 96

DisplayTextOnGraphics

Description

This method displays text on upper and or lower line of the specified LC16 ScreenKey mapped with a graphical image, with a specified color, flash color, text justification and bitmap manipulation.

Parameters

Line1
Type: BSTR
String to display on upper line of specified ScreenKey.

Line2
Type: BSTR
String to display on lower line of specified ScreenKey.

GraphicData
Type: BSTR
Graphic Data to display on the specified key

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

Color
Type: KEY_COLORS
Optional, default: BRIGHT_GREEN
Specifies color and flash color of ScreenKey. The background colors and flashing colors has to be or'ed together to form a complete color specification. See page 64 more information.

DispAttrib
Type: DISP_ATTRIBUTE
Optional, default: NO_JUSTIFY
Specifies text attributes. All available attributes apply to this method.
See page 66 more information.

Return Value Type: long

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

Use after - OpenKeyboard

Remarks

The Row and Column cam be calculated to a key number as follows:
Row * 16 + Column

The functions using row and column specifications can take Row=0 and Column=KeyNumber (Row * 16 + Column). This means the key number received through the KeyPressEvent can be used directly as Column if Row is set to zero.

A BYTE array of 64 bytes should be used to hold the graphical image.
VB users: see page 18. MFC Users: see page 96

DisplaySimpleText

Description

This method displays text on the specified LC16 ScreenKey with a specified color.

This sends a text as ASCII to the console, NO graphics or graphical text. This method uses the *KeyColor* and *Attribute* properties to specify the key's colors and attributes. This method is able to display scrolling texts, as the only one.

Parameters

Text
Type: BSTR
String to be displayed on the specified key. A static text can be up to 10 characters long, five characters on the upper line and five on the lower. If a text longer than 10 character is send, the text will be scrolled from right to left.

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

KeyColor
Type: KEY_COLORS
Specifies the background and flash color for the specified key.

CentreText
Type: BOOL
If set, the text is centered both vertically and horizontally if only one line of text is to be displayed (less than 6 characters specified).

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

This method is a simplified version of the DisplayText method.

When in Text mode, the ScreenKey will display up to five characters on each of the two lines. The text may be up to 10 characters long, the first five will be displayed using the default font (may be downloaded) on the topmost line, and the rest, if any on the bottom-line.

The text "HELLOWORLD" will display "HELLO" on the topmost line and "WORLD" on the bottom-line. The text "HEY JUDE" will display "HEY J" on the topmost line and "UDE " on the bottom-line.

DisplayText24

Description

This method displays text on upper, middle and/or lower lines of the specified LC24 ScreenKey with a specified color, flash color, text justification and bitmap manipulation.

Parameters

Line1
Type BSTR
String to display on upper line of specified ScreenKey.

Line2
Type BSTR
String to display on middle line of specified ScreenKey.

Line3
Type BSTR
String to display on lower line of specified ScreenKey.

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

Color
Type: KEY_COLORS
Optional, default: BRIGHT_GREEN
Specifies color and flash color of ScreenKey.
The background colors and flashing colors has to be or'ed together to form a complete color specification. See page 64 more information.

DispAttrib
Type: DISP_ATTRIBUTE
Optional, default: NO_JUSTIFY
Specifies text attributes. All available attributes apply to this method.
See page 66 more information.

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

The Row and Column cam be calculated to a key number as follows:

Row * 16 + Column

The functions using row and column specifications can take Row=0 and Column=KeyNumber (Row * 16 + Column). This means the key number received through the KeyPressEvent can be used directly as Column if Row is set to zero.

DisplayGraphics24

Description

This method displays Graphics on the specified LC24 ScreenKey. A predefined image can be crested and displayed on the key.

Parameters

GraphicData
Type: BSTR
Graphic Data to display on the specified key

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

Color
Type: KEY_COLORS
Optional, default: BRIGHT_GREEN
Specifies color and flash color of ScreenKey.
The background colors and flashing colors has to be or'ed together to form a complete color specification. See page 64 more information.

DispAttrib
Type: DISP_ATTRIBUTE
Optional, default: NO_JUSTIFY
Specifies text attributes. Only some attributes apply to this method.
See page 66 more information.

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

The Row and Column cam be calculated to a key number as follows:

Row * 16 + Column

The functions using row and column specifications can take Row=0 and Column=KeyNumber (Row * 16 + Column). This means the key number received through the KeyPressEvent can be used directly as Column if Row is set to zero.

A BYTE array of 120 bytes should be used to hold the graphical image.
VB users: see page 18. MFC Users: see page 96

DisplayTextOnGraphics24

Description

This method displays text on upper and or lower line of the specified LC24 ScreenKey mapped with a graphical image, with a specified color, flash color, text justification and bitmap manipulation.

Parameters

Line1
Type: BSTR
String to display on upper line of specified ScreenKey.

Line2
Type: BSTR
String to display on middle line of specified ScreenKey.

Line3
Type: BSTR
String to display on lower line of specified ScreenKey.

GraphicData
Type: BSTR
Graphic Data to display on the specified key

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

Color
Type: KEY_COLORS
Optional, default: BRIGHT_GREEN
Specifies color and flash color of ScreenKey. The background colors and flashing colors has to be or'ed together to form a complete color specification. See page 64 more information.

DispAttrib
Type: DISP_ATTRIBUTE
Optional, default: NO_JUSTIFY
Specifies text attributes. All available attributes apply to this method.
See page 66 more information.

Return Value Type: long
Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after - OpenKeyboard

Remarks

The Row and Column cam be calculated to a key number as follows:
Row * 16 + Column

The functions using row and column specifications can take Row=0 and Column=KeyNumber (Row * 16 + Column). This means the key number received through the KeyPressEvent can be used directly as Column if Row is set to zero.

A BYTE array of 120 bytes should be used to hold the graphical image.
VB users: see page 18. MFC Users: see page 96

SetLED

Description

This method is used to set the LED indicators on the RTI-6000, RTI-3000 and RTI-7000 consoles ON or OFF.

Parameters

LEDNo
Type: LED_CODE
LED indicator no/id.
These can be specified in two different ways, by numbers, counted from the left or by name. One can set one, more than one or all, on or off. A combination of LEDs can be specified by OR'ing the values together. It is not possible to set some ON and others OFF at the same time. The following choices are available:

Definition	Val	Description
LED_1	01h	Leftmost LED
LED_2	02h	…
LED_3	04h	….
LED_4	08h	Rightmost LED
LED_ALL	0F	ALL LEDs

State
Type: LED_STATE

Definition	Val	Description
LED_OFF	00h	Turn LED(s) OFF
LED_ON	01h	Turn LED(s) ON

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

Note: The SK-7000 and SK-3000 do not exist at the moment for Open Systems platforms.

SetKeyAttribute

Description

This method can be used to set certain key attributes, some applying to all keys, others to ScreenKeys only.

Parameters

Row
Type: short
Row number of the key (zero based).

Column
Type: short
Column number of the key (zero based).

Attribute
Type: KEY_ATTRIBUTES
The following attributes are available:

Definition	Val	Description
ENABLE_KEY_BEEP	08h	Console will beep when key is pressed
DISABLE_KEY_BEEP	80h	Console will NOT beep when key is pressed
ENABLE_KEY_RETURN	04h	Console will return key number when key is pressed
DISABLE_KEY_RETURN	40h	Console will NOT return key number when key is pressed
ENABLE_KEY_REPEAT	01h	Console will repeat sending key number when key is held pressed
DISABLE_KEY_REPEAT	10h	Console will only send one key number even when key is held pressed
ENABLE_SCREENKEY	02h	ScreenKey will operate normally
DISABLE_SCREENKEY	20h	ScreenKey will NOT be updated when disabled

Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

This method sends the command to the console, and will therefore apply to all applications connected to the console (shared console). The attributes will be reset to their default state when the console is reset, either hard reset (power-up), or soft reset (receiving reset command).

Sound

Description

This method activates the buzzer/speaker in the ScreenKey console for the specified duration.

Parameters

Duration
Type: long
The number of milliseconds to sound the speaker. The resolution is 100 milliseconds. The minimum duration is 100 ms. If 0 is specified the speaker will **not** sound.

Return Value Type: long
Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **OpenKeyboard**

ConfigSleep

Description

This method enables or disables the Sleep-Mode, and sets the number of seconds after last keypress before console goes into sleep mode.

Parameters

TimeOutSec
Type: short
Optional, default: 0
Number of seconds after last keypress before console goes into sleep mode. If set to 0 the sleep mode is disabled, console never goes to sleep.

Return Value Type: long
Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **OpenKeyboard**

ConfigFlash

Description

This method defines the global flash sequence. The duration the original color shall be held (OffDuration), and the duration the flash color shall be held (OnDuration). The flash configuration applies to all ScreenKeys on the console at any time. The flash configuration may be changed at any time.

Parameters

OnDuration
Type: long
Number of milliseconds (multiple of 50 ms) the flash color shall be ON.

OffDuration
Type: long
Number of milliseconds (multiple of 50 ms) the flash color shall be OFF.

Return Value Type: long
Returns 0 if successful, otherwise the control returns standard Windows error code.

Use after **OpenKeyboard**

Remarks
This command sends the information to the console, and applies to all applications using the console simultaneously.
0 gives the shortest possible duration (50 ms) for both parameters.

ConfigRepeat

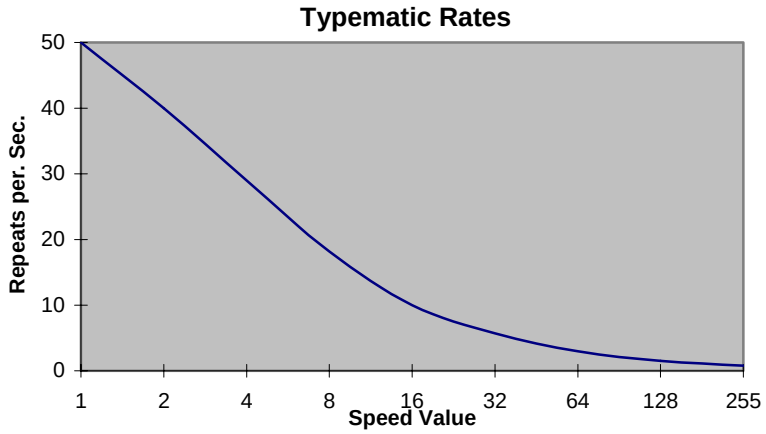
Description
This method defines the global key repeat sequence. The delay before a key starts repeating when held pressed, and the repeat speed to use.

Parameters

Delay
Type: long
Number of milliseconds before the key starts to send repetitive key numbers.
Minimum is 1 ms, maximum is 1275 ms. 0 gives the shortest possible delay (1 ms).

SpeedVal
Type: long
This specifies the repeat rate (Typematic rate), is an non-linear function.
Minimum is 1, maximum is 255.

SpeedVal	Approximately no. of repeats per. Sec.
1	50
2	40
4	30
8	20
16	10
32	6
64	3
128	1.5
255	0.8



Return Value Type: long

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

Use after **OpenKeyboard**

Remarks

This command sends the information to the console, and applies to all applications using the console simultaneously.

SetCharSet

Description

This method defines the font file to use, and downloads the corresponding character set. The font file can be specified by a codepage number, or by a filename.

Parameters

CodePage
Type: long
The CodePage number.
The character set for the desired code page must exist, either as a built in character set, or as a separate file. If the character set is not built in, it may be downloaded to the ScreenKey console using the CharSetFile parameter.

CharSetFile
Type: BSTR
Full or relative (relative to where the ActiveX control resides) pathname of the desired character set.

Return Value Type: long

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

Use after **None**

Remarks

One can either specify a specific code-page to use, or the filename of a font file to be loaded. If a font file is specified, the codepage parameter must be 0 (zero).

If the ScreenKey console is opened, the character set will be loaded immediately. If not, it will be loaded when the OpenKeyboard method is invoked. See “International character sets” on page 19.

The character set format may be obtained by requesting it from SKI Ltd. (See page 2.)

SendSimpleCommand

Description

This method may be used to send simple commands (one byte commands) to the ScreenKey console, not available via other methods. The low-level commands are described in “ScreenKey Low-Level Interface” document.

Parameters

Command
Type: SK_LOWLEVEL_CMD
Available commands are:

Definition	Val	Description
NO_COMMAND	00h	Debugging
RESET_KBD	01h	SW-reset of console, CAUTION
REQUEST_KEYLOCK_EVENT	04h	Requests KeyLock position to be sent
DISABLE_KEY_OPENS	05h	Send only key down events
ENABLE_KEY_OPENS	06h	Send key up and down events
DISABLE_KEY_BLIP	07h	Disables key-blips for all keys
ENABLE_KEY_BLIP	08h	Enables key-blips for all keys
ENABLE_DEBUG	09h	Run in Debug mode, CAUTION
DISABLE_ DEBUG	0Ah	Stops Debug mode
DISABLE_LUHN_TEST	0Bh	Default MSR
ENABLE_LUHN_TEST	0Ch	Turns on LUHN checking for MSR
T1_MANDATORY	0Dh	Defines mandatory tracks
T2_MANDATORY	0Eh	‘’
T3_MANDATORY	0Fh	‘’
T1_T2_MANDATORY	10h	‘’
T1_T3_MANDATORY	11h	‘’
T2_T3_MANDATORY	12h	‘’
T1_T2_T3_MANDATORY	13h	‘’
REQUEST_BEEP	14h	Same as Sound method
ENABLE_DEFAULT_CHARSET	23h	Cancel downloaded character set
CLEAR_MCR_BIT_BUFFERS	24h	For debugging of MSR

Return Value

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

Use after

OpenKeyboard

Remarks

Most of these commands are for **special use only**; be **very careful** if they need to be used.

This command sends the information to the console, and applies to all applications using the console simultaneously (NOT local to client).

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 **OpenKeyboard**

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 **OpenKeyboard**

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 **OpenKeyboard**

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 **OpenKeyboard**

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 **OpenKeyboard**

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 2.4).

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 **OpenKeyboard**

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 2.4).

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 **OpenKeyboard**

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.

CONTROL PROPERTIES

Properties

ErrorMode

Type ERROR_MODE

Get property Supported

Put Property Supported

Available Always

Description ErrorMode defines the behavior when errors occur in the ActiveX Control. There are four levels of error severity:

No error reporting
Fatal errors
Critical errors
Non-Critical errors.

VB is only triggered if an ERROR (HRESULT) has occurred, and this property can therefore be used to define an error (the severity levels) to appear as ERROR or NOT, per default all categories are ERRORS.

When an error occurs, it depends on the current setting of ErrorMode if a FATAL error is triggered.

If ErrorMode is set to the error's severity level or higher, the ERROR bit will be set in the HRESULT, and an error will be triggered in VB. If it is lower it will not have any effect in VB.

See page 67 for more information.

Available options

Definition	Val	Description
ERRMODE_NONE	0	Never generate errors
ERRMODE_FATAL	1	Generate errors for fatal errors only
ERRMODE_CRITICAL	2	Generate errors for fatal errors and critical errors only.
ERRMODE_NONCRITICAL	3	Always generate errors

Remarks None

ErrorNumber

Type ERROR_NUM

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 E_NO_ERROR (0).

This property can hold both error codes generated by SKI ActiveX Control and errors generated by Windows. See page 67 for definitions of the 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 latest 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)

KeyPress

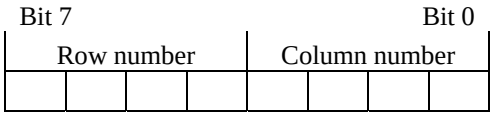
Type long

Get property Supported

Put Property Not supported

Available After first key press event.
Updated after each key press event received. Initially set to –1 (before any key has been pressed).

Description KeyPress holds the last pressed key number. This data is only updated if a key event occurs from console. The data is coded in the following format:



The number is according to the current definition of key numbers. See RedefineKey method (page 29). If the key return has been redefined, the format shown above does not necessarily apply.

KeyPressRow

Type	short
Get property	Supported
Put Property	Not supported
Available	After first key press event. Updated after each keypress event received. Initially set to –1 (before any key has been pressed).
Description	KeyPressRow holds the row number of the last pressed key. The number is according to the current definition of key numbers. See RedefineKey method (page 29).

KeyPressCol

Type	short
Get property	Supported
Put Property	Not supported
Available	After first key press event. Updated after each keypress event received. Initially set to 15 (before any key has been pressed).
Description	KeyPressCol holds the column number of the last pressed key. The number is according to the current definition of key numbers. See RedefineKey method (page 29).
Remarks	Since the initial value (before any key is pressed) is 15 this property should not be used to check for a key press, use KeyPressRow instead.

KeyLockPos

Type	long
Get property	Supported
Put Property	Not supported
Available	Always Updated after each key lock turn event received. Initially set to the current position of the key lock.
Description	KeyLockPos holds the key number of the current KeyLock position. Initially these are numbered from 0 – 4. The number is according to the current definition of key numbers. See RedefineKey method (page 29).

KbdLayout

Type KBD_LAYOUT

Get property Supported

Put Property Not supported

Available After the ScreenKey console has been successfully opened/initiated.

Description KbdLayout holds the layout configuration of the currently opened ScreenKey console. The console is divided into panels of fixed keys or ScreenKeys. A Fixed key panel can hold up to 20 fixed keys, and a ScreenKey panel up to 12 ScreenKeys.
The panels are listed from left to right, Left, center, right.

Available options

Definition	Val	Description
LAYOUT_UNDEF	-1	No configuration obtained from console
FIX_FIX_FIX	0	Three Fixed key panels
FIX_FIX_SCR	1	Fixed-Fixed-ScreenKey
FIX_SCR_FIX	2	Fixed-ScreenKey-Fixed
FIX_SCR_SCR	3	Fixed-ScreenKey-ScreenKey
SCR_FIX_FIX	4	ScreenKey-Fixed-Fixed
SCR_FIX_SCR	5	ScreenKey-Fixed-ScreenKey
SCR_SCR_FIX	6	ScreenKey-ScreenKey-Fixed
SCR_SCR_SCR	7	ScreenKey-ScreenKey-ScreenKey

The initial value of this property and the value if initialization fails, is LAYOUT_UNDEF.

KeyType

Type KEY_TYPE

Get property Supported

Put Property Not supported

Available After the ScreenKey console has been successfully opened/initiated.

Description KeyType holds the ScreenKey type as currently attached to opened ScreenKey console. Currently this supports two ScreenKey resolutions and two backlight color versions.

Available options

Definition	Val	Description
KEYS_UNDEF	-1	No or invalid configuration obtained from console
KEYS_LC24RG	0	LC16 (32x16 pixels) Red-Green backlight
KEYS_LC16RG	1	LC24 (36x24 pixels) Red-Green backlight
KEYS_LC24RGB	2	LC16 (32x16 pixels) Red-Green-Blue backlight
KEYS_LC16RGB	3	LC24 (36x24 pixels) Red-Green-Blue backlight

The initial value of this property and the value if initialization fails, is KEYS_UNDEF.

MSRTracks

Type MSR_TRACKS

Get property Supported

Put Property Not supported

Description MSRTracks describe the tracks read from the most recent MSR swipe. This property may be used to decide which tracks was read and where to find the data. The property can hold one of the following values:

Definition	Val	Description
MSR_NO_TRACKS	0	No track read
MSR_TRACK_1	1	Track 1 read, data in MSRTrack1Data
MSR_TRACK_2	2	Track 2 read, data in MSRTrack2Data
MSR_TRACK_3	4	Track 3 read, data in MSRTrack3Data
MSR_TRACK_1_2	3	Track 1 and track 2 read, data in MSRTrack1Data and MSRTrack2Data
MSR_TRACK_1_3	5	Track 1 and track 3 read, data in MSRTrack1Data and MSRTrack3Data
MSR_TRACK_2_3	6	Track 2 and track 3 read, data in MSRTrack2Data and MSRTrack3Data
MSR_TRACK_1_2_3	7	Track 1, track 2 and track 3 read, data in MSRTrack1Data, MSRTrack2Data and MSRTrack3Data

MSRTrack1Data

Type BSTR

Get property Supported

Put Property Not supported

Description Track1Data property holds the last Magnetic Strip Reader track number one data. The data from MSR will be translated by the control into ASCII format.

MSRTrack2Data

Type BSTR

Get property Supported

Put Property Not supported

Description Track2Data property holds the last Magnetic Strip Reader track number one data. The data from MSR will be translated by the control into ASCII format.

MSRTrack3Data

Type BSTR

Get property Supported

Put Property Not supported

Description Track3Data property holds the last Magnetic Strip Reader track number one data. The data from MSR will be translated by the control into ASCII format.

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

Events

KeyPressEvent

Description

This event occurs when a key is pressed on the ScreenKey console.

Parameters

KeyNo
Type: long
Key pressed.

Remarks

The KeyNo parameter is coded in the following format:

Bit 31				Bit 7							Bit 0
Key Up bit				Row number				Column number			
X	……										

The row and column of the most recent key press can be obtained through the properties KeyPressRow and KeyPressCol, and the key number without the Key Up bit can be obtained from the KeyPress property

The number is according to the current definition of key numbers. The key numbers returned can be redefine using the RedefineKey method (see RedefineKey method, page 30). If the key return has been redefined, the format shown above does not necessarily apply.

If the console is configured to send both key Close and key Open (Down and Up) events, bit 31 will be set to one (1) when an up-event (Open) occurs. See Key Presses, page 20.

KeyLockTurnEvent

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 KeyLockTurnEvent may be triggered manually (from an application) by using the method *SendSimpleCommand* with the parameter *REQUEST_KEYLOCK_EVENT*.

Parameters

KeyPos
Type: long
New KeyLock position, 0-4.

Remarks

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

The KeyLock position number returned can be redefine using the RedefineKey method.

MSRSwipeEvent

Description

This event occurs when a magnetic card has been swiped successfully. The MSR data are accessible through the MSR properties (MSR??????).

Parameters

NoTracks
Type: long
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 MSRTracks, MSRTrack1Data, MSRTrack2Data and MSRTrack3Data.

ErrorEvent

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. Errors occurring in a method, before execution is returned to the invoking application will not be reported through this even, only through an exception and through the properties ErrorNumber and ErrorDescription.

The enumeration type ERROR_NUM contains definitions of all SkAxCtl generated error messages.

Parameters

ErrorID
Type: long
SKI ActiveX Control error numbers, and standard Windows error numbers

ErrorText
Type: BSTR
Error in text format.

K E Y N U M B E R S

Key numbers and Numbering diagrams

To operate the ScreenKey console from an application one needs to know the key numbers the control uses. The SKI ActiveX Control uses a simple numbering system, counting row and columns from the upper left corner of the console. The key number is made up by multiplying the row number by 16 and adding the column number, which means the row number will be in the high nibble and the column number in the low nibble of a byte.

Key numbering system

Upper Left key ⇒

Row:	0	Row:	0	Row:	0
Col:	0	Col:	1	Col:	2
Keynumber:	0x00	Keynumber:	0x01	Keynumber:	0x02
Row:	1	Row:	1	Row:	1
Col:	0	Col:	1	Col:	2
Keynumber:	0x10	Keynumber:	0x11	Keynumber:	0x12

The key numbers may be redefined using the method RedefineKey. There is no preferred way of numbering the keys, it is up to the application programmer, but if more than one console type or layout is supported by the application, it can be useful to redefine the keys, so the algorithms can stay as simple as possible.

The default key numbers for some console types are described below:

The SK-2000 Console

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

The keys are numbered as rows and columns, from top left to bottom right as follows on the 2012 (Fixed-Fixed-ScreenKey):

Fixed keys ScreenKeys

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

When receiving key-press events, the 2012 (Fixed-Fixed-ScreenKey) console uses the following key numbers:

Fixed keys								ScreenKeys		
00h	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah
10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	1Ah
20h	21h	22h	23h	24h	25h	26h	27h	28h	29h	2Ah
30h	31h	32h	33h	34h	35h	36h	37h	38h	39h	3Ah
40h	41h	42h	43h	44h	45h	46h	47h			

The keys are numbered as rows and columns, from top left to bottom right as follows on the 2024 (ScreenKey-ScreenKey-Fixed):

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 0 C 8	R 0 C 9
R 1 C 0	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 2 C 0	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 3 C 0	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 4 C 6	R 4 C 7	R 4 C 8	R 4 C 9

When receiving key-press events, the 2024 (ScreenKey-ScreenKey-Fixed) console uses the following key numbers:

ScreenKeys						Fixed keys			
00h	01h	02h	03h	04h	05h	06h	07h	08h	09h
10h	11h	12h	13h	14h	15h	16h	17h	18h	19h
20h	21h	22h	23h	24h	25h	26h	27h	28h	29h
30h	31h	32h	33h	34h	35h	36h	37h	38h	39h
						46h	47h	48h	49h

The SK-5400 Console

The keys are numbered as rows and columns, from top left to bottom right as in the leftmost figure. Then receiving key-press events the 5400 console uses the key numbers shown in the rightmost figure:

ScreenKeys

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

ScreenKeys

00h	01h	02h
10h	11h	12h
20h	21h	22h
30h	31h	32h

The SK-6000 Console

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

The keys are numbered as rows and columns, from top left to bottom right as follows:

ScreenKeys

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

ScreenKeys

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

Fixed keys

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

When receiving key-press events the 6000 console uses the following key numbers:

ScreenKeys

00h	01h	02h	03h	04h	05h
10h	11h	12h	13h	14h	15h

ScreenKeys

06h	07h	08h	09h	0Ah	0Bh
16h	17h	18h	19h	1Ah	1Bh

Fixed keys

0Ch	0Dh	0Eh	0Fh
1Ch	1Dh	1Eh	1Fh
2Ch	2Dh	2Eh	2Fh

The SK-7000 Console

The keys are numbered as rows and columns, from top left to bottom right as follows on the 7000 (ScreenKey-Fixed-Fixed):

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 0 C 8	R 0 C 9	R 0 C 10
R 1 C 0	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 0	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 0	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

When receiving key-press events, the 7000 (ScreenKey-Fixed-Fixed) console uses the following key numbers:

ScreenKeys			Fixed keys							
00h	01h	02h	03h	04h	05h	06h	07h	08h	09	0Ah
10h	11h	12h	13h	14h	15h	16h	17h	18h	19h	1Ah
20h	21h	22h	23h	24h	25h	26h	27h	28h	29h	2Ah
30h	31h	32h	33h	34h	35h	36h	37h	38h	39h	3Ah
			43h	44h	45h	46h	47h	48h	49h	4Ah

S C R E E N K E Y C O L O R S

ScreenKey Colors

The ScreenKey colors are defined by combining the background color and the flashing color (OR'ing them together).

The following colors are available with **all** ScreenKey types (i.e. both RG and RGB versions):

Definition	Val	Description
NO_COLOR	00h	No color, the key will appear very dark green
DARK_RED	01h	
BRIGHT_RED	02h	
DARK_GREEN	03h	
BRIGHT_GREEN	04h	
DARK_ORANGE	05h	
BRIGHT_ORANGE	06h	
GREENISH_ORANGE	07h	
REDDISH_ORANGE	08h	

RGB ScreenKeys also offer the following addiitonal colors:

Definition	Val	Description
DARK_BLUE	09h	
BRIGHT_BLUE	0Ah	
PINK	0Bh	
DARK_PURPLE	0Ch	
BRIGHT_PURPLE	0Dh	
TORQUOISE	0Eh	
WHITE	0Fh	

The flashing colors/definitions usable with both RG and RGB keys are the following:

Definition	Val	Description
FLASH_DARK_RED	10h	
FLASH_BRIGHT_RED	20h	
FLASH_DARK_GREEN	30h	
FLASH_BRIGHT_GREEN	40h	
FLASH_DARK_ORANGE	50h	
FLASH_BRIGHT_ORANGE	60h	
FLASH_GREENISH_ORANGE	70h	
FLASH_REDDISH_ORANGE	80h	

RGB ScreenKeys also offer the following addiitonal flash colors:

Definition	Val	Description
DARK_BLUE	90h	
BRIGHT_BLUE	A0h	
PINK	B0h	
DARK_PURPLE	C0h	
BRIGHT_PURPLE	D0h	
TORQUOISE	E0h	
WHITE	F0h	

S C R E E N K E Y A T T I B U T E S

ScreenKey Attributes

The ScreenKey attributes are defined by combining one or more attribute definitions from the enumeration structure DISP_ATTRIBUTE:

Definition	Val	Mutual Exclusive			Description
NO_JUSTIFY	0000h				No justification or attribute
HOR_LEFT	0001h	✓			Horizontal justification, LEFT Applies to both lines of text.
HOR_CENTRE	0002h	✓			Horizontal justification, CENTRE Applies to both lines of text
HOR_RIGHT	0004h	✓			Horizontal justification, RIGHT Applies to both lines of text
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.
VERT_CENTRE	0010h		✓		Vertical justification, CENTRE Applies only if one line of text is defined. Any of the lines will be centered vertically.
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.
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 character will be removed.
INVERT_TOP	0100h				Invert the topmost line
INVERT_BOTTOM	0200h				Invert the bottom line
INVERT_KEY	0300h				Invert the whole key
FLIP_HOR	0400h				Flip the image horizontally
FLIP_VER	0800h				Flip the image vertically
EXCLUSIVE_OR_MAP	1000h				Map text and graphics by using XOR Applies to DisplayTextOnGraphics
FAIL_TOO_LONG	4000h			✓	Make the method fail if text is too long
NOFAIL_TOO_LONG	8000h			✓	Displays the truncated text if it is too long

E R R O R S

Error Handling

The ActiveX Control reports errors through runtime system (exceptions) and two properties in the SKI ActiveX Control itself. The control can be set up to trigger errors depending on the current error level (error mode). The SKI ActiveX Control properties will always contain the error codes if an error has occurred (independent of error level), while the exception only will contain the error code if an exception is raised. This will appear differently in the different programming languages.

The error levels are set using the ErrorMode property, and are the following:

Definition	Val	Description
ERRMODE_NONE	0	Report NO errors When ErrorMode is set to this level, no errors will be reported through the error handling in the run-time system. The execution will continue as if no error occurred.
ERRMODE_FATAL	1	Report Fatal errors When ErrorMode is set to this level, only fatal errors will be reported through the error handling in the run-time system.
ERRMODE_CRITICAL	2	Report Critical errors When ErrorMode is set to this level, both fatal errors and critical errors will be reported through the error handling in the run-time system.
ERRMODE_NONCRITICAL	3	Report Non-Critical errors When ErrorMode is set to this level, all errors will be reported through the error handling in the run-time system.

The possible errors and their categories are listed on the following pages.

The error codes are split into different categories, depending what part of the system generated the error (where the error occurred). Errors occurring in the control interface part (SkAxCtl.dll) are in the range from 1000h to 10FFh. Errors occurring in the control itself (SkAxCtl.exe) are in the range from 2000h to 20FFh, and errors reported from the console are in the range from 3000h to 30FFh. These numbers will be held in the property ErrorNumber. The corresponding error message will be held in ErrorDescription. In e.g. VB the built-in object **Err** will also hold the error number, and the error description if it is an error generated by the SKI ActiveX Control. If it is an error generated by the run-time system or by Windows itself, the **Err.Description** will normally contain the string "**Unknown error**".

The **Err.Number** holds the error number as in the **ErrorNumber** property in the control, but with the 31st bit set (80002067h if it failed to open COM-port). This bit triggers the exception that reports the error.

The SKI ActiveX Control properties (ErrorNumber and ErrorDescription) will be cleared every time a method is invoked, so it is possible to check the properties after returning to the application. The **Err** object will NOT be cleared when a method is invoked, and will hold the contents until another error occurs, or the application itself clears it, which can be done with the **Err.Clear** statement.

In C++, errors will generate exceptions, and may be caught using the **try** and **catch** statements, using the Cexception class. The error number and description will also be available through the properties ErrorNumber and ErrorDescription. The error information may be retrieved immediately after a method has been invoked, if it succeeds the error number will be E_NO_ERROR (0), and the description string will be blank. Note that if the error level (ErrorMode) is set to a level where an exception is raised, the function needs a try and catch construction, if not, the execution of that function will be interrupted and the exception will be passed on to the calling function.

See the Language Interface section for more information.

Console errors and MSR errors

The default setting is to report console error, but not to report MSR errors. This setting can be changed by using the NO_KBD_ERRORS and RETURN_MSR_ERRORS flags when opening the console. Any combination of these may be used. To handle these error messages, which will be reported through the **ErrorEvent** handler, can be useful to avoid situation where the console get out of synch with the application because an error occurred in the console.

NO_KBD_ERRORS means no errors reported from the console, possibly except MSR errors, will be reported to the application. Errors falling into this category are:

E_KBD_ERR01	E_KBD_ERR21
E_KBD_ERR02	E_KBD_ERR22
E_KBD_ERR03	E_KBD_ERR23
E_KBD_ERR04	E_KBD_ERR24
E_KBD_ERR05	E_KBD_ERR26
E_KBD_ERR07	E_KBD_ERRKYB
E_KBD_ERR20	

RETURN_MSR_ERRORS means that errors occurred when a magnetic card is swiped will be reported to the application. Errors falling into this category are:

E_KBD_ERR08	E_KBD_ERR14
E_KBD_ERR09	E_KBD_ERR15
E_KBD_ERR10	E_KBD_ERR16
E_KBD_ERR11	E_KBD_ERR17
E_KBD_ERR12	E_KBD_ERR18
E_KBD_ERR13	E_KBD_ERR19

NOTE:
The setting of **ErrorMode** DOES apply to there errors also, which means that if some categories of errors are filtered because of the setting of ErrorMode, these categories of errors will also be filtered if they are generated by the console. All errors reported from the console are categorized as CRITICAL.

Error numbers and messages

The following errors may occur. Level F indicates FATAL, C indicates CRITICAL and NC indicates NON-CRITICAL. If the Event column contains 'Y', the error message is sent as an event because it is an error not cause when invoking a method or property, else it raises an exception or reported through the error properties only.

Error definition	Error #	Level	Event	Error description
E_NO_ERROR	0000h	-		None (No error occurred)
E_NO_SERVER	1001h	F		Console not opened (server not started).
E_CREATE_SERVER	2065h	F		Failed to start server.
E_CONNECT_SERVER	2066h	F		Failed to connect to server. OBSOLETE
E_OPEN_COMPORT	2067h	F		Failed to open COM-port.
E_INIT_COMPORT	2068h	F		Failed to initialize COM-port.
E_WRONG_KBD_TYPE	2069h	F		Wrong console type or configuration.
E_ALLOC_TRTABLE	206Ah	F		Failed to allocate memory for translation tables.
E_FAILED_CONFIG	206Bh	F		Failed to initialize ScreenKey console.
E_IN_EXCLUSIVE	206Ch	F		Console already opened in Exclusive mode.
E_NOT_EXCLUSIVE	206Dh	F		Console attempted opened in Exclusive mode while in Shared mode. OBSOLETE
E_KBD_NOT_CLAIMED	206Eh	C		Console not claimed by the calling application. OBSOLETE
E_CANNOT_CLAIM	206Fh	C		Could not claim console. OBSOLETE
E_ALREADY_CLAIMED	2070h	C		Console already claimed. OBSOLETE
E_CANNOT_RELEASE	2071h	C		Could not release console. OBSOLETE
E_SEND_FAIL	2072h	C		Failed to send command.
E_CLAIMED_BYYOU	2073h	NC		Already claimed by application. OBSOLETE
E_UNDEF_KEY	2074h	NC		Undefined key.
E_NOT_CLAIMED_BYYOU	2075h	NC		Not claimed by application. OBSOLETE
E_DISCARD_CLIENT_FAIL	2076h	C		Failed to discard all clients. OBSOLETE
E_TOP_TEXT_LONG	2077h	F		Top line did not fit on key.
E_BOT_TEXT_LONG	2078h	F		Bottom line did not fit on key.
E_BOTH_TEXT_LONG	2079h	F		Neither lines did fit on key.
E_KDB_ALREADY_OPENED	107Ah	NC		Console is already opened, server exists.
E_NOT_SUPPORTED	107Bh	NC		Function is NOT supported.
E_KBD_DUMP_FAILED	107Ch	C		Console dump failed (timeout).
E_CHARSET_NOT_FOUND	107Dh	C		ScreenKey character set could not be found/loaded.
E_MSR_DECODE_ERROR	207Eh	NC	Y	Could not decode MSR data.
E_ILLEGAL_MSR_FORMAT	207Fh	NC	Y	Illegal MSR track format.
E_COMM_OUTP_FAIL1	2091h	NC	Y	Failed sending data to port. (Init)
E_COMM_OUTP_FAIL2	2092h	NC	Y	Failed sending data to port. (Comm 1)
E_COMM_OUTP_FAIL3	2093h	NC	Y	Failed sending data to port. (Comm 2)

Error definition	Error #	Level	Event	Error description
E_COMM_OUTP_FAIL4	2094h	NC	Y	Failed sending data to port. (Comm 3)
E_COMM_OUTP_FAIL5	2095h	NC	Y	Failed sending data to port. (RomDnl 1)
E_COMM_OUTP_FAIL6	2096h	NC	Y	Failed sending data to port. (RomDnl 2)
E_COMM_OUTP_FAIL7	2097h	NC	Y	Failed sending data to port. (RomDnl 3)
E_COMM_INP_FAIL1	2098h	C	Y	Failed receiving data from port. (ProcChar)
E_COMM_INP_FAIL2	2099h	C	Y	Failed receiving data from port. (ProcChar 1)
E_COMM_INP_FAIL3	209Ah	C	Y	Failed receiving data from port. (ProcChar 2)
E_COMM_DNL_FAIL	209Bh	C	Y	Failed during ROM download, Failed.
E_COMM_DNL_ABND	209Ch	C	Y	Failed during ROM download, Abandoned.
E_COMM_DNL_TOUT	209Dh	C	Y	Failed during ROM download, Timed out.
E_COMM_DNL_UNKNOWN	209Eh	C	Y	Failed during ROM download, Unknown error.
E_COMM_DNL_PROB	209Fh	NC	Y	Failed during ROM download, Unknown problem.
E_COMM_ERROR_EVENT	20A0h	C	Y	Comm event error (Framing error or Overrun).
E_COMM_FUNCT_ERROR	20A1h	F	Y	Comm function error.
E_KBD_ERR02	3041h	F	Y	CPU POST Failed.
E_KBD_ERR01	3042h	F	Y	RAM POST failed.
E_KBD_ERR04	3062h	C	Y	Stack overflow.
E_KBD_ERR08	3063h	C	Y	MSR – Incomplete byte at Card-End.
E_KBD_ERR03	3064h	F	Y	PROM POST failed.
E_KBD_ERR07	3065h	C	Y	Invalid Command-code from PC.
E_KBD_ERR05	3066h	C	Y	Internal buffers are corrupt.
E_KBD_ERR16	3081h	C	Y	MSR – Timeout on card reader, Track 1.
E_KBD_ERR12	3082h	C	Y	MSR – Check digit error.
E_KBD_ERR20	3083h	F	Y	Invalid combination of key panels.
E_KBD_ERR10	3084h	C	Y	MSR – Second card swipe, while data buffered.
E_KBD_ERR18	3085h	C	Y	MSR – CLS signal but no data read.
E_KBD_ERR14	3086h	C	Y	MSR – Parity error on card bytes.
E_KBD_ERR22	3087h	C	Y	PC not ready (Parallel port only) OBSOLETE
E_KBD_ERR09	3088h	C	Y	MSR – End-of-Card, but NO data.
E_KBD_ERR17	3089h	C	Y	MSR – Timeout on card reader, Track 2.
E_KBD_ERR13	308Ah	C	Y	MSR – Interrupt Buffer full – bit(s) lost.
E_KBD_ERR21	308Bh	F	Y	Cannot establish CPU speed.
E_KBD_ERR11	308Ch	C	Y	MSR – LRC Error on card.
E_KBD_ERR19	308Dh	C	Y	MSR – Mandatory track missing on card.
E_KBD_ERR15	308Eh	C	Y	MSR – Processing Buffer full – Byte(s) lost.
E_KBD_ERR26	30A4h	C	Y	Serial In-Buffer full – byte(s) lost.

Error definition	Error #	Level	Event	Error description
E_KBD_ERR24	30A8h	C	Y	Checksum error on data from PC.
E_KBD_ERR23	30B0h	F	Y	Timer 0 has stopped.
E_KBD_ERRKYB	30B3h	C	Y	Bad/lost keystroke (buffer full).

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 ScreenKey console 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. Errors may also happen when the application is not using the control directly, like console errors and communication errors (because communications with the console is asynchronous). These will be reported through the ErrorEvent handler. It is important to handle all errors properly, also during application development. This way errors and design problems can be exposed, identified and corrected at an early stage. See also “Error Handling”, page 68.

There are two main types of errors reported to the PC; those occurring in the PC (SKI ActiveX Control), and the ones occurring in the console.

PC errors (SkAxCtl)

E_NO_ERROR **0h**

(blank)

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

E_NO_SERVER **1001h**

Console not opened (server not started).

This error message will be given when a method or property is invoked before the console is initialized. It will normally have been proceeded by an error when attempting to open the console, using the OpenKeyboard method, without success. This will then give the error message E_CREATE_SERVER.

To correct this problem, make sure the ScreenKey console has been successfully opened before invoking any other methods or properties.

E_CREATE_SERVER **2065h**

Failed to start server.

This error may appear if the SKI ActiveX Control Interface module is not able to create the SKI ActiveX Control server. This can happen if the control (SkAxCtl.exe) or the proxy stub (SkAxCtlPS.dll) does not exist or if it has not been registered properly. It can also happen if the computer does not have sufficient resources.

To correct this problem, reinstall the SKI ActiveX Control and make sure all files are successfully registered. Also make sure the computer has sufficient resources.

E_CONNECT_SERVER **2066h**

Failed to connect to server.

Obsolete, should not appear.

E_OPEN_COMPORT **2067h**

Failed to open COM-port.

This error occurs when the specified com-port could not be opened. The cause of this can be that the port is already opened by another application, the port does not exist in the system or if the port specification is wrong (spelt incorrectly). The right format of the port specification is: “COMx”, where x is the port number, e.g. “COM1”.

To correct this problem, make sure no other application is using the com-port, and that the port exists in the system and that the spelling is correct. It may help to re-boot the computer and start the application using the SKI ActiveX Control before other applications are stared.

E_INIT_COMPORT **2068h**

Failed to initialize COM-port.

This error occurs when the specified com-port could not be initialized (the port has been found and opened). The cause of this can be that the system does not have the required system resources.

To correct this problem, see E_OPEN_COMPORT.

E_WRONG_KBD_TYPE **2069h**

Wrong console type or configuration.

This error occurs if the console connected reports a console panel configuration not available for the type of console specified in the OpenKeyboard method. This error may also be triggered when there is a lack of system resources.

To correct this problem, make sure the console connected is the same as specified in the OpenKeyboard method.

Note: This error message will not appear if a SK-2000 series console is specified and another console is connected, because the 2000 consoles may have all possible combinations of key panels and therefore no error will be issued. In this case the key numbers will not match, and the ScreenKey contents may appear on the wrong keys.

E_ALLOC_TRTABLE **206Ah**

Failed to allocate memory for translation tables.

This error occurs if the computer does not have enough memory for the key press translation tables.

To correct this problem, make sure the computer has enough memory available before starting the application. Close other application to free up memory, or re-boot the computer if necessary. This error message may indicate that an application is leaking memory – **it could be your application**!

E_FAILED_CONFIG **206Bh**

Failed to initialize ScreenKey console.

This error occurs if the ScreenKey console could not be initialized. When this error occurs, the com-port has been successfully opened and initialized, but the ScreenKey console does not report back the console configuration. The reason for this may be that the console is not connected, connected to the wrong com-port, or not powered.

To correct this problem, make sure the console is connected to the right com-port and that it is powered.

E_IN_EXCLUSIVE **206Ch**

Console already opened in Exclusive mode.

This error occurs when attempting to open the ScreenKey console when it is already opened by another application (instance of application) in exclusive mode (EXCLUSIVE_USE). The most likely problem is that another instance of the same application is still running, or that the application terminated without closing the console (CloseKeyboard). This means the SKI ActiveX Control (SkAxCtl.exe) is still running, unaware that the application, including SkAxCtl.dll, has terminated.

To correct this problem, first make sure that no other application using the console in exclusive mode is running. If this is not the case, use TaskManager or equivalent tool to see if SkAxCtl.exe is still running. This can then be terminated from TaskManager by selecting the program and clicking on “End Process”.

During development this may happen quite frequently, especially when debugging the application.

E_NOT_EXCLUSIVE **206Dh**

Console attempted opened in Exclusive mode while in Shared mode.

Obsolete, should not appear.

E_KBD_NOT_CLAIMED **206Eh**

Console not claimed by the calling application.

Obsolete, should not appear.

E_CANNOT_CLAIM **206Fh**

Could not claim console.

Obsolete, should not appear.

E_ALREADY_CLAIMED **2070h**

Console already claimed.

Obsolete, should not appear.

E_CANNOT_RELEASE **2071h**

Could not release console.

Obsolete, should not appear.

E_SEND_FAIL **2072h**

Failed to send command.

This error is given if the control is not able to send a command to the console. It may occur if the console is not taking data from the PC fast enough. Parts of a command will not be sent because the control checks if there is space enough for the whole command, and report this error if this is not the case.

To correct this problem, send the command (invoke the method with the same parameters) again until it is successful.

In some cases the application can be displaying data in real time, and then the console should display the latest data on the ScreenKeys. In that case, do not attempt to re-send the data that failed to be sent to the console, go on to the next set of data.

Note: An application should avoid sending so much data that this error occurs. If the application is updating the console this often, it will be harder to read the contents of the keys.

E_CLAIMED_BYYOU **2073h**

Already claimed by application.

Obsolete, should not appear.

E_UNDEF_KEY **2074h**

Undefined key.

This error will occur when a non-existing (undefined) key is referenced. The operation on the key will not take place.

To correct this problem, use the right key number when operating on the keys. The KeyLock positions may are also referred as keys, and may be referred to as keys, and could then generate this error. The most common reason for this error is when the application is designed for another console type or layout than the console connected, or when the keys have been renumbered and the code does not refer to the new numbering scheme.

E_NOT_CLAIMED_BYYOU **2075h**

Not claimed by application.

Obsolete, should not appear.

E_DISCARD_CLIENT_FAIL **2076h**

Failed to discard all clients.

Obsolete, should not appear.

E_TOP_TEXT_LONG **2077h**

Top line did not fit on key.

E_BOT_TEXT_LONG **2078h**

Bottom line did not fit on key.

E_BOTH_TEXT_LONG **2079h**

Neither lines did fit on key.

These errors will given when attempting to display a text string on a ScreenKey, and it/they are to long to fit on a line. The string(s) will not be sent to the console if this error occurs.

To correct this problem, reduce the number of characters to display on the specific line. See “What if the text doesn’t fit” on page 17. This error can only occur if the flag FAIL_TEXT_TOO_LONG is given with the OpenKeyboard method, or FAIL_TOO_LONG is given with the Display… methods.

This error is meant to give feedback to the application if the whole text was not possible to display on the key. The application should then have some algorithm to reduce the number of characters, and attempt to display the text again – until no error occurs. If the application cannot reduce the length of the string, it can use NOFAIL_ TOO_LONG with the Display… methods on the last attempt. If the application in general accepts that the text is truncated, the flag FAIL_TEXT_TOO_LONG should not be given with the OpenKeyboard.

E_KDB_ALREADY_OPENED **107Ah**

Console is already opened, server exists.

This error is given when an application already has opened the console.

To correct this problem, avoid attempting to open the console multiple times. It could be that a CloseKeyboard failed, and therefore the console is still open, or that OpenKeyboard is called twice.

E_NOT_SUPPORTED **107Bh**

Function is NOT supported.

This error will be given when a non-supported method is invoked. This may happen if the console currently opened does not support the method the application is using.

To correct this problem, check that console type used with the OpenKeyboard method is the intended one, and that this console supports the method/feature being invoked. A typical example may be the SetLED method used with a SK-2000 console, as this console is not equipped with LEDs.

E_KBD_DUMP_FAILED **107Ch**

Console dump failed (timeout).

This error may occur when attempting to dump contents of a console, using the RequestKbdData method. The attempt timed out.

To correct this problem, verify that the console is powered and “alive”. When RequestKbdData is used, it is normally because an error has occurred at an earlier stage, and the console is not able to function properly. Try to provoke the failure again, and attempt to dump the data at an earlier stage.

E_CHARSET_NOT_FOUND **107Dh**

ScreenKey character set could not be found/loaded.

This error may occur if the character set requested is not found. A character set may be chosen by using the method SetCharSet, either by specifying the code page number or the character set file itself. If the SetCharSet method is not in use, the control retrieves the current codepage from the operating system.

To correct this problem, check if SetCharSet is in used, and that the specified character set file is in the specified folder, or in the ScreenKey\Bin folder if no path is specified. If a code page is specified, check that SK_*nnnn*.SKF (where *nnnn* if the code page number) exists in the ScreenKey\Bin folder. If the SetCharSet method is not in use, the current codepage is probably not supported. In either case, check with your distributor if the desired code page is available.
To work around this problem, there are a few possibilities:
- Set up the PC with a different codepage, by changing the language from the Control Panel.
- Use the method SetCharSet to set the current character set to 1252, which is the Western/European character sets.
- Design a new character set – tools may be acquired from RTI.
- or as a temporary solution, use an existing character set file, and rename this to the code page currently in use (this will not display the right characters, it will only allow you to continue running the application).

E_MSR_DECODE_ERROR **207Eh**

Could not decode MSR data.

This error will be given when a magnetic card is swiped and the data on the card cannot be decoded, because the track contained NO data or too much data (overflows the buffer).

To correct this problem; No recommendation, the card is probably faulty, or has a coding not recognized by the console.

E_ILLEGAL_MSR_FORMAT **207Fh**

Illegal MSR track format.

This error is given when a MSR track has an illegal format. The reason for this error could be a special format like Track one data on all tracks, etc.

To correct this problem; No recommendation, the card is probably a card with a coding that’s not recognized by the control.

E_COMM_OUTP_FAIL1 **2091h**

Failed sending data to port. (Init).

E_COMM_OUTP_FAIL2 **2092h**

Failed sending data to port. (Comm 1).

E_COMM_OUTP_FAIL3 **2093h**

Failed sending data to port. (Comm 2).

E_COMM_OUTP_FAIL4 **2094h**

Failed sending data to port. (Comm 3).

E_COMM_OUTP_FAIL5 **2095h**

Failed sending data to port. (RomDnl 1).

E_COMM_OUTP_FAIL6 **2096h**

Failed sending data to port. (RomDnl 2).

E_COMM_OUTP_FAIL7 **2097h**

Failed sending data to port. (RomDnl 3).

These errors are low-level communication failures, and should not normally occur. They are Non-Critical, but they are a sign something is not right.

To correct this problem, a restart of the application, including the SKI ActiveX Control, and a reset of the ScreenKey console may make the problem go away. During development of applications, faulty code may have upset the control, and a restart is therefore required. The errors may also occur if the operating system is corrupt or unstable, then a reboot is required. If the problem is persistent, please contact support@ScreenKeys.com.

E_COMM_INP_FAIL1 **2098h**

Failed receiving data from port. (ProcChar).

E_COMM_INP_FAIL2 **2099h**

Failed receiving data from port. (ProcChar 1).

E_COMM_INP_FAIL3 **209Ah**

Failed receiving data from port. (ProcChar 2).

These errors are low-level communication failures, and should not normally occur. They are Critical, and will prevent the application from receiving input from the console.

To correct this problem, see **E_COMM_OUTP_FAIL1**.

E_COMM_DNL_FAIL **209Bh**

Failed during ROM download, Failed.

E_COMM_DNL_ABND **209Ch**

Failed during ROM download, Abandoned.

E_COMM_DNL_TOUT **209Dh**

Failed during ROM download, Timed out.

E_COMM_DNL_UNKNOWN **209Eh**

Failed during ROM download, Unknown error.

E_COMM_DNL_PROB **209Fh**

Failed during ROM download, Unknown problem.

These errors are low-level communication failures regarding the download of the latest firmware to the console (ROM code), and should not normally occur. They are Critical (except 209F), and will prevent the console from operating normally. The latest firmware will match the functionality of the control, and it is therefore crucial that the code is downloaded to the console successfully.

To correct this problem, see **E_COMM_OUTP_FAIL1**.

ERR_COMM_ERROR_EVENT **20A0h**

Comm event error - 20A0h

This error may occur if the communication between the PC and the console fails, because of a framing error, an overrun etc.

To correct this problem, repeat the operation (method) provoking the error again. If the error is persistent the hardware in the PC or in the console may be faulty, or the cable between them may be faulty.

ERR_COMM_FUNCT_ERROR **20A1h**

Comm function error.

This error may occur when accessing functions in the communication driver. The driver software and/or the operating system may be corrupt or unstable.

To correct this problem, restart the application, and/or reboot the PC. If the error is persistent there is probably some software corrupting the operating system, or the PC is running out of system resources.

ScreenKey console errors

The following error messages may be reported from the console itself. The ScreenKey console will report errors occurring to the application, if possible. Most of these errors have to do with the Magnetic Stripe Reader, but there are also error situations related to the console hardware and communication. The non-MSR errors fall into two categories; Recoverable errors and Non-recoverable errors.

- Non-recoverable errors cause the console to hang (not respond), and therefore no error code is returned to the application. These errors cause the console to either issue a beep sequence continuously (OEM 5400 firmware), or to light the top left ScreenKey continuously with the error code.
- Recoverable errors will be reported to the application. These are divided into two main categories:
  - Serious errors, where an error message will be attempted sent to the application. Since the error is of such a nature that there is no guarantee that it will be successfully sent, the console issues a beep sequence to notify the operator of the error (OEM 5400 firmware). These are described as “Beep sequence”. SK-7000 firmware records this error in the error log which can be accessed via the Diagnostics mode.
  - Non-serious errors, where an error message always will be sent. These errors do not normally generate a beep sequence, to keep the noise level down. Beep sequences can be turned on using “debug mode”. Debug mode can be enabled using the command SendSimpleCommand(ENABLE_DEBUG). These beep sequences are described as “Debug beep sequence”. SK-7000 firmware records this error in the error log which can be accessed via the Diagnostics mode.

MSR related errors will normally not be reported back to the application, unless the flag RETURN_MSR_ERRORS is given with the OpenKeyboard method. All MSR error messages described below states that they are reported back to the application, this is assuming this flag has been given.

Some console errors states they are reported to the application, this assumes that the flag NO_KBD_ERRORS is NOT given with the OpenKeyboard method.

E_KBD_ERR02 **3041h**

CPU POST Failed.

Non-recoverable error – not reported to application.
Beep sequence: ON until reset of console
The ScreenKey console has detected that the CPU’s interrupt system is not working properly during Power On Self Test. Even though interrupts have been set to disable, they are still active. This is unlikely to ever be seen and is a fatal error.

To correct this problem; Replace or repair console.

E_KBD_ERR01 **3042h**

RAM POST failed.

Non-recoverable error – not reported to application.
Beep sequence: ON until reset of console
The ScreenKey console runs a RAM test on its internal and external RAM during Power On Self Test. This has failed for some reason.

To correct this problem; Replace or repair console.

E_KBD_ERR04 **3062h**

Stack overflow.

Non-recoverable error – reported to application continuously.
Beep sequence: Short – Long – Short (continuous).
The ScreenKey console runs a test to see that its stack in internal RAM is OK continuously. It has overflowed.

To correct this problem; Replace or repair console. The problem can also be caused by errors in the ROM code (firmware), and in this case contact support@ScreenKeys.com.

If this error occurs, it shall be trapped in the error event function. A SW-reset (SendSimpleCommand(RESET_KBD)) should be sent to reset the console. The console should then be closed, and it may be attempted re-opened. If the error keeps on appearing, the console should be repaired or replaced.

E_KBD_ERR08 **3063h**

MSR – Incomplete byte at Card-End.

Recoverable error – reported to application.
Debug beep sequence: Long – Long – Short
MSR data is read from the card reader a bit at a time. Depending on which track is being read, a “byte” will be made up of 5 or 7 bits. In this case, there aren’t enough bits to make up a byte.

To correct this problem; No Recommendation, the card is probably faulty.

E_KBD_ERR03 **3064h**

PROM POST failed.

Non-recoverable error – not reported to application.
Beep sequence: ON until reset of console
The ScreenKey console runs a test on its PROM during Power On Self Test. This has failed for some reason.

To correct this problem; Replace or repair console.

E_KBD_ERR07 **3065h**

Invalid Command-code from PC.

Recoverable error – reported to application.
Debug beep sequence: Long – Short – Long
The ScreenKey console is expecting the first byte (Command No.) of a new Command from the PC. However, the byte received is not a valid Command No. The reason for this message could be a problem in the ScreenKey console firmware, but most likely a communication problem, due to an error in the SKI ActiveX Control or a cabling problem.

To correct this problem; Check the cabling from the PC to the console. If the problem is persistent, the problem could be the SKI ActiveX Control – contact support@ScreenKeys.com.

E_KBD_ERR05 **3066h**

Internal buffers are corrupt.

Non-recoverable error – reported to application continuously.
Beep sequence: Short – Long – Long.
The ScreenKey console uses a number of RAM work areas. There is a marker at the end of each area. If this marker is missing it means that the buffer has overflowed. Markers are checked repeatedly.

To correct this problem; Report to support@ScreenKeys.com or check web-site (www.ScreenKeys.com) for update of firmware and or PC software (control).
If this error occurs, it shall be trapped in the error event function. A SW-reset (SendSimpleCommand(RESET_KBD)) should be sent to reset the console. The console should then be closed, and it may be attempted re-opened. If the error keep on appearing, the console should be repaired or replaced.

E_KBD_ERR16 **3081h**

MSR – Timeout on card reader, Track 1.

Recoverable error – reported to application.
Debug beep sequence: Long – Short – Short – Short
Data started to come in from Track-1 but the ScreenKey console was expecting more and gave up waiting after approximately 2 seconds. Probably due to a failure on the part of the operator to swipe the card properly.

To correct this problem; Try swiping the card again.

E_KBD_ERR12 **3082h**

MSR – Check-digit error.

Recoverable error – reported to application.
Debug beep sequence: Short – Long – Short – Short
The ScreenKey console has calculated the Luhn check digit value for the A/C number read from the magnetic card and found that it does not match the last digit of the card. See **Commands 0Bh** and **0Ch** in PLLUM001.doc for more information on Luhn Checks.

To correct this problem;
Use SendSimpleCommand(DISABLE_LUHN_TEST) to disable the Luhn check in the ScreenKey console. Not all A/c numbers read from cards have a Luhn check digit.

E_KBD_ERR20 **3083h**

Invalid combination of key panels.

Non-recoverable error – not reported to application in POST, reported continuously during normal operation.
Beep sequence: Long – Long – Short – Short.
The ScreenKey console has detected that old and new ScreenKey panels are mixed on the same console. This should never occur.

To correct this problem; Contact distributor for repair or replacement console.
There may be a slight possibility to recover by sending a SW-reset command to the console.

E_KBD_ERR10 **3084h**

MSR – Second card swipe, while data buffered.

Recoverable error – reported to application.
Debug beep sequence: Short – Short – Long – Short
The ScreenKey console doesn't allow a second card to be swiped until the current card data has been successfully sent to the PC. The problem is probably that the PC is not taking data from the console due to communication problem. If this message comes through to the PC, the problem has probably been overcome, and the PC is ready to read the card data again.

To correct this problem; Swipe card again.

E_KBD_ERR18 **3085h**

MSR – CLS signal but no data read.

Recoverable error – reported to application.
Debug beep sequence: Long – Short – Long – Short
The CLS is data read from the magnetic card that marks the end of valid data on the card. Here we got this signal but no data was read from the card.

To correct this problem; Card may have been swiped incorrectly – retry. Card may be faulty, try another card.

E_KBD_ERR14 **3086h**

MSR – Parity error on card bytes.

Recoverable error – reported to application.
Debug beep sequence: Short – Long – Long – Short
Parity error on the data read from the Card Reader.

To correct this problem; No recommendation, card is probably faulty.

E_KBD_ERR22 **3087h**

PC not ready. (Parallel port only)

Obsolete, this error should not occur.

E_KBD_ERR09 **3088h**

MSR – End-of-Card, but NO data.

Recoverable error – reported to application.
Debug beep sequence: Short – Short – Short – Long
The sensor says that card is out of the reader but no data was read.

To correct this problem; Swipe card again. It may be that the card was swiped backwards or with the magnetic stripe around the wrong way.

E_KBD_ERR17 **3089h**

MSR – Timeout on card reader, Track 2.

Recoverable error – reported to application.
Debug beep sequence: Long – Short – Short – Long
Data started to come in from Track-2 but the ScreenKey console was expecting more and gave up waiting after approximately 2 seconds. Probably due to a failure on the part of the operator to swipe the card properly.

To correct this problem; Try swiping the card again.

E_KBD_ERR13 **308Ah**

MSR – Interrupt Buffer full – bit(s) lost.

Recoverable error – reported to application.
Debug beep sequence: Short – Long – Short – Long
The ScreenKey console's buffer that holds input from the MSR is full.

To correct this problem; Swipe card again.

E_KBD_ERR21 **308Bh**

Cannot establish CPU speed.

Non-recoverable error – not reported to application.
Beep sequence: Long – Long – Short – Long
Can't work out how fast this ScreenKey console's CPU is.

To correct this problem; Contact distributor for repair or replacement console.

E_KBD_ERR11 **308Ch**

MSR – LRC Error on card.

Recoverable error – reported to application.
Debug beep sequence: Short – Short – Long – Long
The Longitudinal Redundancy Check (LRC) byte is one of the pieces of data read from a Card. In this case this checksum failed.

To correct this problem; No recommendation, card is probably faulty.

E_KBD_ERR19 **308Dh**

MSR – Mandatory track missing on card.

Recoverable error – reported to application.
Debug beep sequence: Long – Short – Long – Long
SendSimpleCommand(<tracks>_MANDATORY) (the commands covers all possible track combinations) are used to tell the ScreenKey console which if any tracks are mandatory. If the data contained on a card doesn't include the specified mandatory track(s) then the ScreenKey console rejects the card; that is no data is sent back even if a track was read successfully.

To correct this problem; No recommendation, because the card swiped did not meet the defined requirements. If any of the other tracks were desired, one should relax restrictions on what tracks are mandatory.

E_KBD_ERR15 **308Eh**

MSR – Processing Buffer full – Byte(s) lost.

Recoverable error – reported to application.
Debug beep sequence: Short – Long – Long – Long
There are two buffers handling MSR input – the input buffer itself that takes raw data from the card reader and the processed buffer which holds the result of analyzing the raw data. The buffer holding the processed data has overflowed. This error should not occur because the buffer is checked for overflowing. If it should happen, some other bug has corrupted the buffer.

To correct this problem; Report to support@ScreenKeys.com or check web-site (www.ScreenKeys.com) for update of firmware and or PC software (control).

E_KBD_ERR26 **30A4h**

Serial In-Buffer full – byte(s) lost.

Recoverable error – reported to application.
Debug beep sequence: Short – Short – Long – Short – Short.
The ScreenKey console's Serial data receive buffer is full. The buffer is 250 bytes. At 170 bytes the ScreenKey console tells the PC that it is busy so the PC should stop sending. Only if the PC continues to send will the buffer ever fill. In that case it is probably an error in the control itself, or the configuration of the serial port or the here-driver has been corrupted.

To correct this problem; Report to support@ScreenKeys.com or check web-site (www.ScreenKeys.com) for update of firmware and or PC software (control). Alternatively reboot the PC, and start the application without any other programs running.

E_KBD_ERR24 **30A8h**

Checksum error on data from PC.

Recoverable error – reported to application.
Debug beep sequence: Short – Short – Short – Long - Short
The PC has sent a Command to the ScreenKey console but the console has decided that the checksum is invalid. This is probably because of communication problems.

To correct this problem; May be due to cables coming loose at the back of the PC or ScreenKey console. Ensure all cables are secure. If the problem is persistent, the problem could be the SKI ActiveX Control – contact support@ScreenKeys.com.

E_KBD_ERR23 **30B0h**

Timer 0 has stopped.

Non-recoverable error – not reported to application.
Beep sequence: ON until reset of console.
The code in the ScreenKey console runs various checks to ensure the interrupts in the console are operating correctly. In this case, Timer-0 should have been active but is not.

To correct this problem; Contact distributor for repair or replacement console. The problem could be a faulty CPU.

E_KBD_ERRKYB **30B3h**

Bad/lost keystroke (buffer full).

Recoverable error – not reported to application.
Debug beep sequence: Long – Long – Short – Short – Short
This condition will be indicated by not issuing key-beeps when the output buffer is full.

The console can buffer up to 10 key presses. Key presses are transmitted to the PC immediately so the buffer will usually never have more than one key press awaiting transmission. However, if the PC is busy and cannot accept transmissions, then the buffer will fill up if keys are pressed. This is not regarded as an error condition, but that the key just pressed has been discarded.

To correct this problem; There is no way to get the lost keystroke back, since it is already discarded.
One reason for this error happening could be that the PC is overloaded, and it is herefore up to the PC administrator to sort that problem out.
Another problem could be faulty cabling. Check the cabling from the PC to the console.
If it is none of the above mentioned reasons, and the problem is persistent, the problem could be the SKI ActiveX Control – contact support@ScreenKeys.com or check web-site (www.ScreenKeys.com) for update of firmware and or PC software (control).

P R O G R A M M I N G L A N G U A G E I N T E R F A C E

Language Interfaces

The ActiveX 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 SKI ActiveX Control Interface (SkAxCtl.dll), not directly to the SKI ActiveX Control (SkAxCtl.exe).

VB (Visual Studio) supports IntelliSense and parameter enumeration, which is implemented in the ActiveX control.

To use the SKI ActiveX 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 **"ScreenKey console ActiveX Control"**. Tick the box to the left, and click **OK**.
- The SKI ActiveX Control icon  will then appear 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 ScreenKey1, but it can be renamed by redefining the "(Name)" property.
- The ScreenKey console 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 ScreenKey1.OpenKeyboard("COM1", KBD_TYPE_2000, EXCLUSIVE_USE)
```

or

```
ScreenKey1.OpenKeyboard "COM1", KBD_TYPE_2000, EXCLUSIVE_USE
```

Examples on use from Visual Basic, in the Form load event function we will open the console:

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

   ' Example on use of METHOD
   ' Initiate the 200 console for COM1
   ScreenKey.OpenKeyboard "COM1", KBD_TYPE_2000, EXCLUSIVE_USE

   ' Example on use of PROPERTY
   ' Warn user and close it if not the right console layout
   If ScreenKey.KbdLayout <> FIX_FIX_SCR Then
      MsgBox "Wrong console configuration!"     ' Alert user
      ScreenKey.CloseKeyboard
   endif

Exit Sub

ErrHandler:   ' Capture errors here
   MsgBox Err.Description   ' Alert user using a messagebox
End Sub

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

   ' Close the ScreenKey console
   ScreenKey.CloseKeyboard

End Sub

' Example on use of EVENT
' When a key press is received through the KeyPressEvent,
' display the KeyNo in hexadecimal system
Private Sub ScreenKey_KeyPressEvent(ByVal KeyNo As Long)

   ' Use message box to tell the user
   MsgBox Hex(KeyNo) + "h"

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 ErrorNumber or ErrorDescription properties.

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

```
Void CSKCTLTestDlg::OnInit()
{
   Cstring csComPort;
   TCHAR szErrMsg[255];

   // Set the error mode to ERRMODE_NONCRITICAL;
   // always generate any errors
   ScreenKey.SetErrorMode( 3 );
   try {
      // Get the COMM-port string from the drop-down box
      GetDlgItemText(IDC_COMPORT, csComPort);
      // Open the 5400 console in EXCLUSIVE_USE mode
      ScreenKey.OpenKeyboard( csComPort, 5, 1 );
      // Update the status field
      SetDlgItemText( IDC_STATUS, “Console Initialised!” );
   }

   // 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 ScreenKey properties.

```
Void CSKCTLTestDlg::OnInit( void )
{
   Cstring csComPort;

   // Set the error mode to ERRMODE_NONE;
   // never generate any errors
   ScreenKey.SetErrorMode( 0 );
   // Get the COMM-port string from the drop-down box
   GetDlgItemText(IDC_COMPORT, csComPort);
   // Open the 5400 console in EXCLUSIVE_USE mode
   ScreenKey.OpenKeyboard( csComPort, 5, 1 );

   // If error occurred display the error message, else an “all well” msg.
   If ( ScreenKey.GetErrorNumber() != 0 )
      SetDlgItemText( IDC_STATUS, ScreenKey.GetErrorDescription() );
   else
      SetDlgItemText( IDC_STATUS, “Console Initialised!” );

}
```

Using MFC

Some special considerations have to be taken into account when using the SKI ActiveX Control from an MFC application, in particular when using the methods dealing with graphical images (DisplayGraphics and DisplayTextOnGraphics). When adding the control to an MFC application the wizard generates a wrapper class, normally called *screenkeyctl.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 methods. 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:

BSTR array
0x12
0x00
0xAB

Conversion for ANSI and Unicode →

Character array
0x22
0x21
0x20
0x20
0x2B
0x2A

Conversion for Unicode only →

Character array
0x22
0x00
0x21
0x00
0x20
0x00
0x20
0x00
0x2B
0x00
0x2A
0x00

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 are 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 );
   ```

   At the moment ScreenKeyType can only be SK_LC16.

3. When using the methods DisplayGraphics or DisplayTextOnGraphics, insert the Convert method from this class, e.g. as follows:
   ```
   m_ScreenKey.DisplayGraphics( SkImgCnv.Convert( Image ),
                                Row, Col, 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)

Delphi 5

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

To install the SKI ActiveX Control into Delphi, do the following:

- From the *Component* menu select *Import ActiveX Control…*
- Select *ScreenKey console ActiveX Control* from the list.
- Click on Install to install the control. The class will by default be *TscreenKey*, and the palette will be *ActiveX*.
- When the *Install dialog* appears, click on *OK*.
- When the *Confirm dialog* appears click on *Yes*.

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

Example of opening the ScreenKey console when a button (Open) is clicked.

```
Procedure Tform1.OpenClick(Sender: Tobject);
begin

   { Set error mode to ERRMODE_NONE;  never give exceptions }
   ScreenKey.ErrorMode := ERRMODE_NONE;

   { Open the console in Exclusive mode }
   ScreenKey.OpenKeyboard('COM1', KBD_TYPE_6000, EXCLUSIVE_USE);

   { Check for any errors, display error message in error  message field }
   if ScreenKey.ErrorNumber <> E_NO_ERROR then
      ErrMsg.Caption := ScreenKey.ErrorDescription;
end;
```

D O C U M E N T A T I O N C O N T R O L

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

ScreenKey	Registered Trademarked key-switch with a backlit LCD screen incorporated.
ScreenKey console	RTI's range of consoles containing ScreenKeys.
PASKeyboard	Former name of ScreenKey console
ActiveX	Name of technology used for communication between applications/modules.
COM	As ActiveX.
BSTR	Visual Basic type string.
LED	Light Emitting Diode.
*MSR	Magnetic Stripe Reader
POS	Point of Sale—the cash register in a shop.

A.3 Historical Change Reference

Issue	Date	Author	Changes Made
1.0	04/04/00	Bjorn Liane	First release
1.1	28/06/00	Bjorn Liane	Updated for Version 1.1
1.2	16/11/00	Bjorn Liane	Corrected errors, changed layout.
1.3	05/10/01	Bjorn Liane	Updated for version 1.3
1.4	16/03/04	M McDonnell	Updated for release 2.0
1.5	07/05/04	M McDonnell	Updated for release 2.1
1.6	22/06/05	M McDonnell	Updated for OEM5400 v5.0 (LC24 and RGB)
1.7	15/12/05	M McDonnell	Slight modifications to text for LC24/RGB
1.8	30/11/06	M McDonnell	Update new registry path

A.4 Change Summary

Issue	Change description
1.1	Updated for SkAxCtl 1.1 Added description of special functions for Wrapper DLL (callbacks) Added troubleshooting section. Added error messages. Added Key Press, KeyLock Turn and MSR Swipe chapters. Minor updates.
1.2	Corrected errors in description of DisplaySimpleText Rearranged chapters, plus other aesthetic changes.
1.3	Updated for version SKI ActiveX Control, Version 1.3. Added description of how to handle different codepages. Added information for MFC users. Corrected errors.
1.4	Removed non-COM wrapper information Removed multiple application interface Added SK-7000 methods Reformatted to use letter page size
1.5	Included information on interface handshaking setting in registry Included reference to COM wrapper document
1.6	Added information on LC24 and RGB support Modified Config Byte info for new OEM5400 and SK-7000 models
1.7	Rephrasing of text relating to LC24 and RGB support
1.8	Update with new registry path for download and handshake info