KeyChk - Checking Keys From Your Program

KeyChk Overview

KeyChk is a function which allows your program to check the presence and status of a Key. Your program has complete control over when **KeyChk** is called and what action is taken if the Key check succeeds or fails.

The **KeyChk** function is supplied in an **EverKey** DLL. Win32 programs should use the **KeyChk.dll** file and programs built for the x64 platform should use the **KeyChk64.dll** file.

KeyChk.dll supports both Rockey4 and Rockey4ND Keys. **KeyChk64.dll** only supports the newer Rockey4ND Keys.

To simplify the text, this manual refers to both DLLs as the **KeyChk** DLL.

Information is exchanged between your program and **KeyChk** via a Secure Communication Block (**SCB**) data area that resides within your program. Before your program calls the **KeyChk** function, it must set up an **SCB** area which tells **KeyChk** what to do. Once **KeyChk** is called, it performs a check for your Key, reports the results in your **SCB** area, sets an Error Code and returns control to your program.

The **KeyChk** function only requires one parameter, a pointer to the **SCB**. The returned value of the **KeyChk** function itself is always zero (**0**) and does not need to be checked. The results of the Key check are returned in the Error Code fields of the **SCB**.

When To Call the KeyChk Function

The best time to check for the presence of your Key is when your program is NOT in the middle of doing something else. The reason for this is that you have to allow for the possibility of the Key not being present. If the Key is not present, then you should inform the Customer and provide the option of Retry or Cancel without having to deal with anything else.

Usually, a good time to check for the Key is when your program first starts executing before it has opened any files or databases. Another good time to call **KeyChk** is when the User starts a major program function, such as printing a report.

How often should you call **KeyChk**? Try to make this a realistic time limit to protect your Product without being overly paranoid. Calling **KeyChk** when the program starts up is fine for most programs. If your program runs all day, you might want to verify the presence of the Key every 15 minutes or between major operations. If your program performs a lot of different functions, then you might want to check for the presence of the Key before starting any of those functions.

Adding a KeyChk Call To Your Program

To show how to use **KeyChk**, we have provided the complete Source Code for a number of Example programs in various languages. The easiest way to add one or more **KeyChk** calls to your program is to copy and paste from our Example Source Code and then re-build your program.

All our Example Source Code uses a **Product-ID** of "**Examples**" and a **Product PIN** of '**375391873**". Of course, your **Product-ID** and **Product PIN** will be different and you will need to update your Source Code with your unique values. Both of these fields are shown on the **Build Keys** screen of the **KeyBuild** program. Be sure to load the **Options File** for your Product because each **Options File** can have a different **Product-ID** and **Product PIN**.

Once a **KeyChk** call has been added to your program, it can be distributed in a compressed format.

KeyChk File Location And Name

If your application uses 32-bit programs, your install process should copy the **KeyChk.dll** file to your Product's program folder. Do NOT copy it to the Windows "System" or "System32" folder.

If your application uses 64-bit programs, your install process should copy the **KeyChk64.dll** file to your Product's program folder.

For more security, you may want to rename the **KeyChk** DLL file to something that will blend in with the names of the files that are included with your Product. This is allowed, but be sure to change the filename that is referenced by your Source Code.

What KeyChk Cannot Do

By design, **KeyChk** is not allowed to do the following:

- Increase the Uses Limit. (**KeyChk** can only Decrease this Limit).
- Turn the Feature-Flags On or Off. (**KeyChk** can only Read these Flags).

Both of the above operations can only be done with the **Renew** program.

KeyChk Calling Procedure

The following steps are used to perform a **KeyChk** function call from your Program:

Step 1	Define the Secure Communications Block (**SCB**).
Step 2	Initialize the required fields of the **SCB**. The fields that must be initialized are listed with the description of the functions.
Step 3	Call the **KeyChk** function.
Step 4a	If the **Error Code** field in the **SCB** is zero, the Key check was successful. This means that the Key was found and was not expired. All the fields in the **SCB** will be available for display or other use. Your program should continue as normal.
Step 4b	If the **Error Code** field in the **SCB** is non-zero, the Key check failed. This means that your Key is expired, invalid or NOT plugged into the computer. None of the fields in the SCB should be considered valid, except the **Error Code** fields. Your program should take whatever action you feel is necessary. This can range from an Error message to just shutting down. Some Companies allow their Customers to run in "**Demo**" mode if a valid Key is not present.

NOTE: If the Key check fails, it is recommended that you display or log the **Error Code**, **Error Status 1** and **Error Status 2** fields in a Hexadecimal format. If you need to consult our Support staff about the Error, they will need this information.

KeyChk Function Codes Summary

Here is a short description of each **KeyChk** function:	
Function #1	This function allows you to verify the presence of your Key based on the **Product-ID**, **Product PIN** and **Serial Number** (optional). If found, the following fields are returned from the Key: • **Serial Number** (if not specified). • **Uses Limit.** • **Time Limit**. • **Expire-Date**. • **User-Data**. • **Lasered-ID**. • **Feature-Flags**. • **Registration Info** (if supported by Key). No fields in the Key are updated by this call. If you have set a **Time Limit** and the Key's Timer has not yet been started, this function will start the Timer ticking.
Function #2	Same as **Function #1** except the **Uses Limit** is updated.
Function #3	Same as **Function #1** except the **Uses Limit** AND the **User-Data** are updated. Be careful with this function! Whatever **User-Data** you provide in the **SCB** will overwrite the entire 16 bytes of **User-Data** in the Key. If you want to preserve part of the **User-Data** in the Key, use **Function #1** first to read the existing **User-Data**, update the **SCB** with what needs to be changed and then use this function to write it back to your Key.
Function #4	The main purpose of this function is to allow for a very fast check for the presence of your Key. If you want to frequently check for the presence of your Key without slowing down your program, you can first use **Function #1** to get the **Lasered-ID** of your Key. After that, you can use this function to ensure that your Key is still plugged into the computer. Nothing in the Key is updated and the only field returned is the Error Code which tells you if the Key was found.
Function #9	As of **EverKey**, **Version 6**, this function is no longer needed. New programs should not use this function. However, it will continue to be supported for existing programs. It is the same as **Function #1** except NOTHING in the Key is updated. This is a **READ-ONLY** function.

KeyChk Function Codes Details

Here is the detailed description of **KeyChk** Function #1:

Function #1	This function allows you to verify the presence of your Key based on the **Product-ID**, **Product PIN** and **Serial Number** (optional). If the Key is found, the following fields are returned: • **Serial Number** (if not specified). • **Uses Limit.** • **Time Limit**. • **Expire-Date**. • **User-Data**. • **Lasered-ID**. • **Feature-Flags**. • **Registration Info** (if supported by Key). No fields in the Key are updated by this call. If the Key has a **Time Limit** and it's Timer has not yet started, this function will start the Timer ticking.
Before calling **KeyChk** setup the following **SCB** fields:	
Error Code	**6240** (hex) It's a good programming practice to always set the Error Code to this value before calling **KeyChk**. If **KeyChk** doesn't find your **SCB**, it can't set the Error Code, but it will already be set to the correct value and your program will take the proper action.
Function Code	**1**
SCB-ID	"**Kc.6**" (ASCIIZ)
Product-ID	Set this to your **Product-ID** shown on the **Build Keys** screen of the **KeyBuild** program. This field is case sensitive.
PIN	Set this to your **Product PIN** shown on the **Build Keys** screen of the **KeyBuild** program. Please note that you CANNOT use the **User-Data PIN** here.
Product Serial Number	Normally, you will set this field to zero so the Key check will succeed as long as **KeyChk** finds a Key with your specific **Product-ID** and **Product PIN**. If you also want to verify that the Key present has a specific **Product Serial Number**, then set this field to the specific Serial Number that you want to verify.
The remaining **SCB** fields will be ignored by **KeyChk**.	

Here is the detailed description of **KeyChk** Function #2:

Function #2	This function allows you to verify the presence of your Key based on the **Product-ID**, **Product PIN** and **Serial Number** (optional). Additionally, this function will update the **Uses Limit** based on the setting of this field when **KeyChk** is called (see field details below). If the Key is found, the following fields are returned: • **Serial Number** (if not specified). • **Uses Limit** (updated value). • **Time Limit**. • **Expire-Date**. • **User-Data**. • **Lasered-ID**. • **Feature-Flags**. • **Registration Info** (if supported by Key). Except for the **Uses Limit**, no other fields in the Key are updated by this call. If the Key has a **Time Limit** and it's Timer has not yet started, this function will start the Timer ticking.
Before calling **KeyChk** setup the following **SCB** fields:	
Error Code	**6240** (hex) It's a good programming practice to always set the Error Code to this value before calling **KeyChk**. If **KeyChk** doesn't find your **SCB**, it can't set the Error Code, but it will already be set to the correct value and your program will take the proper action.
Function Code	**2**
SCB-ID	"**Kc.6**" (ASCIIZ)
Product-ID	Set this to your **Product-ID** shown on the **Build Keys** screen of the **KeyBuild** program. This field is case sensitive.
PIN	Set this to your **Product PIN** shown on the **Build Keys** screen of the **KeyBuild** program. Please note that you CANNOT use the **User-Data PIN** here.
Product Serial Number	Normally, you will set this field to zero so the Key check will succeed as long as **KeyChk** finds a Key with your specific **Product-ID** and **Product PIN**. If you also want to verify that the Key present has a specific **Product Serial Number**, then set this field to the specific Serial Number that you want to verify.
Uses Limit or **Uses Update**	Set this field to zero if you do not wish to update the **Uses Limit** of the Key. Otherwise, set this field to a negative number equal to the number of "uses" that you wish to subtract from the Key's current **Uses Limit**. For example, to take one "use" away, set this field to "-1" (negative one). Please Note: **KeyChk** is not allowed to ADD to the current value of the **Uses Limit**. This can only be done with the **Renew** program.
The remaining **SCB** fields will be ignored by **KeyChk**.	

Here is the detailed description of **KeyChk** Function #3:

Function #3	This function allows you to verify the presence of your Key based on the **Product-ID**, **Product PIN** and **Serial Number** (optional). Additionally, this function will update the **Uses Limit** and **User-Data** based on the setting of these fields when **KeyChk** is called (see field details below). If the Key is found, the following fields are returned: • **Serial Number** (if not specified). • **Uses Limit** (updated value). • **Time Limit**. • **Expire-Date**. • **User-Data** (updated value). • **Lasered-ID**. • **Feature-Flags**. • **Registration Info** (if supported by Key). Except for the **Uses Limit** and **User-Data**, no other fields in the Key are updated by this call. If the Key has a **Time Limit** and it's Timer has not yet started, this function will start the Timer ticking. Be careful with this function! Whatever **User-Data** you provide in this call will overwrite the entire 16 bytes of **User-Data** in the Key. If you want to preserve part of the **User-Data** in the Key, use **Function #1** first to read the existing **User-Data**, update the **SCB** with what needs to be changed and then use this function to write it back to your Key.
Before calling **KeyChk** setup the following **SCB** fields:	
Error Code	**6240** (hex) It's a good programming practice to always set the Error Code to this value before calling **KeyChk**. If **KeyChk** doesn't find your **SCB**, it can't set the Error Code, but it will already be set to the correct value and your program will take the proper action.
Function Code	**3**
SCB-ID	"**Kc.6**" (ASCII)
Product-ID	Set this to your **Product-ID** shown on the **Build Keys** screen of the **KeyBuild** program. This field is case sensitive.
PIN	Set this to your **Product PIN** shown on the **Build Keys** screen of the **KeyBuild** program. Please note that you CANNOT use the **User-Data PIN** here.
Product Serial Number	Normally, you will set this field to zero so the Key check will succeed as long as **KeyChk** finds a Key with your specific **Product-ID** and **Product PIN**. If you also want to verify that the Key present has a specific **Product Serial Number**, then set this field to the specific Serial Number that you want to verify.
Uses Limit or **Uses Update**	Set this field to zero if you do not wish to update the **Uses Limit** of the Key. Otherwise, set this field to a negative number equal to the number of "uses" that you wish to subtract from the Key's current **Uses Limit**. For example, to take one "use" away, set this field to "-1" (negative one). Please Note: **KeyChk** is not allowed to ADD to the current value of the **Uses Limit**. This can only be done with the **Renew** program.
User-Data	Set this field to the 16 bytes of data that you want to store in the secure memory of your Key. Be careful with this function! Whatever **User-Data** you provide here will overwrite the entire 16 bytes of **User-Data** in the Key. If you want to preserve part of the **User-Data** in the Key, use **Function #1** first to read the existing **User-Data**, update the **SCB** with what needs to be changed and then use this function to write it back to your Key. In **EverKey** Version 2, the last two bytes (16 bits) of this field was called the **User-Data Flags**.
The remaining **SCB** fields will be ignored by **KeyChk**.	

Here is the detailed description of **KeyChk** Function #4:

Function #4	This function allows you to verify the presence of your Key **VERY QUICKLY** based only on the **Lasered-ID**, which is unique for every **EverKey** Key. Since the purpose of this function is to perform the Key check as quickly as possible, it does not return any **SCB** fields except **Error Code**, **Error Status 1** and **Error Status 2**. No fields in the Key are updated by this call. If the Key has a **Time Limit** and it's Timer has not yet started, this function **WILL NOT** start the Timer ticking.
Before calling **KeyChk** setup the following **SCB** fields:	
Error Code	**6240** (hex) It's a good programming practice to always set the Error Code to this value before calling **KeyChk**. If **KeyChk** doesn't find your **SCB**, it can't set the Error Code, but it will already be set to the correct value and your program will take the proper action.
Function Code	**4**
SCB-ID	"**Kc.6**" (ASCIIZ)
Lasered-ID	Set this field to the **Lasered-ID** of the Key that you want to be sure is still plugged into the computer. The easiest way to get the **Lasered-ID** for your Key is to use Function #1 specifying your **Product-ID** and **Product PIN**. As a successful result of that function, the **Lasered-ID** of your specific Key will be returned in the **SCB**. Save that returned value and from then on, you can this function for a very fast Key check.
The remaining **SCB** fields will be ignored by **KeyChk**.	

KeyChk SCB Layout

This section describes the Secure Communication Block (**SCB**) data area that is used to exchange information between your program and the **KeyChk** DLL.

Since the **SCB** resides within your program, the layout must be defined (declared) in your program so information can be passed to and from the **KeyChk** DLL. The size of your **SCB** definition MUST be 512 bytes.

Here is the field-by-field layout of the **SCB**:

Field	Type	Offset (decimal)	Size (bytes)	Description
Error Code	Unsigned Word	0	2	This is the Main **KeyChk** Error Code field. It should be set to 6240 (hex) before calling KeyChk. It will be returned as Zero if your Key was found and not Expired.
Error Status 1	Unsigned Word	2	2	This field is only used if the Error Code field is returned non-Zero. It contains additional information about the Error.
Error Status 2	Unsigned Word	4	2	This field is only used if the Error Code field is returned non-Zero. It contains additional information about the Error.
Function Code	Unsigned Byte	6	1	This field tells **KeyChk** what you want it to do. It should be set prior to calling **KeyChk**. Each function is described in detail later in this chapter.
SCB-ID	String (ASCIIZ)	7	5	This is a "Signature" field used by **KeyChk** to verify that this block of memory is the **SCB**. Set it to "**Kc.6**" before calling **KeyChk**. For backward compatibility, if this field is set to "**..?Z**", **KeyChk** will assume that your program is still using an 80-byte **SCB** as defined in **EverKey** v2. This allows old programs to use the newest version of the **KeyChk** DLL with NO changes.
Product-ID	String (UTF-8 Unicode)	12	9	Before calling **KeyChk**, set this field to the **Product-ID** of the Key that you want to find and check. This is shown on the **Build Keys** panel of the **KeyBuild** program. This field is case sensitive.
Reserved	Unsigned Byte	21	1	Reserved for future use.
Feature-Flags	Unsigned Word	22	2	These are 16 single-bit flags that are stored securely in the memory of the Key. If the bit is set to 1, the flag is On. The least significant bit represents Feature-Flag #1 and the most significant bit represents Feature-Flag #16. These flags can be set by **KeyBuild** when the Key is built. Later, once the Key is in the Customer's possession, the **Renew** program may be used to turn On additional Feature-Flags. Also, **Renew** can turn Off any of the Feature-Flags.
Product Serial Number	Unsigned DWord	24	4	Before calling **KeyChk**, set this field to the **Product Serial Number** of the Key that you want to find and check. Normally, you will set this field to Zero so the Key check will succeed as long as **KeyChk** finds a Key with your specific **Product-ID** and **Product PIN**. If you also want to verify that the Key present has a specific **Product Serial Number**, then set this field to the specific Serial Number that you want to verify.
Product PIN	Unsigned DWord	28	4	Before calling **KeyChk**, set this field to the **Product PIN** of the Key that you want to find and check. This is shown on the **Build Keys** panel of the **KeyBuild** program. Please note that you CANNOT use the **User-Data PIN** here.

SCB field Layout (cont.)

Field	Type	Offset (decimal)	Size (bytes)	Description
Uses Limit or **Uses Update**	Signed Word	32	2	The current **Uses Limit** of the Key will always be returned in this field. This is the number of uses left until the Key expires. This value is originally set from the **Uses Limit** option of the **KeyBuild** program. If the Key does not have an **Uses Limit**, this field will be returned as -1 (negative one) which means “Unlimited”. When using **KeyChk** Function 2 or 3, this field can be set to a negative number equal to the number of “uses” that you wish to subtract from the Key's current **Uses Limit**. For example, to take one “use” away, set this field to "-1" (negative one) and then call **KeyChk** Function 2.
Time Limit	Unsigned Word	34	2	Number of days left until the Key expires. This value is originally set from the **Time Limit** option of the **KeyBuild** program. If the Key does not have a **Time Limit**, this field will be returned from **KeyChk** as FFFF (hex) which means “Unlimited”. Please note that if the Key has an Expire-Date, this value will be Zero (0) if there are less than 24 hours until the Key expires. Once the Key does expire, the Error Code will be set to 0x6143.
ED_ASCII	String (ASCIIZ)	36	9	**Note**: This field is kept here for backward compatibility with programs that still use the Version 2 SCB layout. It's recommended that all new programs use the **Expire-Date** field, located at offset 272. That field uses Unicode characters and formats the date using the Local Region Settings. This field is the Expiration date of the Key in ASCIIZ format. This date string will always be 8 ASCII characters with a format of MM/DD/YY. This date is originally set from the **Time Limit** option of the **KeyBuild** program. If the Key does not have a **Time Limit**, this field will be set to "No Limit".
User-Data	Unsigned Byte	45	16	You can use this area to securely store anything you like in the Key. For example, some Customers store a Password in the Key so it is not embedded in their program. This data is originally set from the **User-Data** option of the **KeyBuild** program. In **EverKey** Version 2, the last two bytes (16 bits) were called the **User-Data Flags**.
Reserved	Unsigned Byte	61	6	Reserved for future use.
Lasered-ID	Unsigned Byte	67	8	Permanent Serial Number of the Key. This field cannot be reprogrammed and is guaranteed unique for each **EverKey** Key. The 1st Byte of this field indicates the **Key Type**: **E1** (hex): **Rockey4** Key. **E2** (hex): **Rockey4ND** Key.
Renew Used-Count	Unsigned Byte	75	1	This is how many times the Key has been successfully updated by the **Renew** program.
Reserved	Unsigned Byte	76	4	Reserved for future use.

SCB field Layout (cont.)

Field	Type	Offset (decimal)	Size (bytes)	Description
Registered Name	String (UTF-16 Unicode)	80	64	**Name** of the registered Owner of your Product Key. This field is set from the **Registration Information Name** option of **KeyBuild** and cannot be changed by **KeyChk** or **Renew**.
Registered Company	String (UTF-16 Unicode)	144	64	**Company** Name of the registered Owner of your Product Key. This field is set from the **Registration Information Company** option of **KeyBuild** and cannot be changed by **KeyChk** or **Renew**.
Registered Other Info	String (UTF-16 Unicode)	208	64	Use this field for any **Other** information about the registered Owner of your Product Key. For example, Phone Number or Email Address. This field is set from the **Registration Information Other** option of **KeyBuild** and cannot be changed by **KeyChk** or **Renew**.
Expire-Date	String (UTF-16 Unicode)	272	32	Expiration date of the Key. This date is originally set from the **Time Limit** option of the **KeyBuild** program. The format of the **Expire-Date** field will always use the Local Region Settings. If the Key does not have a **Time Limit**, this field will be set to "No Limit".
Expire-Time	String (UTF-16 Unicode)	304	32	Expiration time of the Key on it's last day of operation. This time is originally set to the time that the Key was built by the **KeyBuild** program. The format of the **Expire-Time** field will always use the Local Region Settings. If the Key does not have a **Time Limit**, this field will be set to "No Limit".
Reserved	Unsigned Byte	336	176	Reserved for future use.

Upgrading From Version 2

The new **KeyChk DLL** has several improvements over the **Kechk32.dll** used in **EverKey** Version 2:

- Supports both Rockey4 and Rockey4ND Keys. Rockey4NDs require no Device Drivers, are about 4 times faster and have more memory so they can store Customer Registration Information.
- Returns the Expire-Date and Time in Unicode characters using the format dictated by the User's Local Region Settings. This eliminates the need for your Program to reformat the data returned in the SCB.
- Returns the Renew-Used Count field.
- Returns the new Key fields for Feature-Flags and Registered Customer Name, Company and Other.
- Does not require **lbfs32.dll**.
- Is backward compatible and can be used as a "drop-in" replacement for **Kechk32.dll**, usually requiring "**No Source Code Changes**" or re-compiling.

In order to take full advantage of all the improvements in the new **KeyChk.dll**, you will need to make a few changes to of your Product. Here's how you upgrade:

- Change your source code to call function "**KeyChk**" instead of "**KECHK32**".
- Change your source code to use "**KeyChk.dll**" instead of "**Kechk32.dll**".
- Update your source code definition of the **SCB** to be 512 bytes in size instead of 80 bytes. For a complete description of the **SCB** and its field layout, refer to the "**KeyChk SCB Layout**" section of this chapter.
- Change your source code to set the "**SCB-ID**" field to "**Kc.6**" instead of "**..?Z**".
- Replace the "**Kechk32.dll**" file with "**KeyChk.dll**".
- Enjoy!

If you are not able to make source code changes to your Product programs, there's still a good chance you can use the new **KeyChk.dll**.

We have designed the new DLL as a "drop-in" replacement for **Kechk32.dll**. It is totally backward compatible, having the same entry point, using the same calling convention and supporting the old SCB layout.

To use the new DLL with your existing Product program(s), simply rename the "**KeyChk.dll**" file to "**Kechk32.dll**" and copy it over the existing "**Kechk32.dll**" file. That's it!

Of course, using the new DLL without making any source code changes means you will not be able to access the new **SCB** fields. Even so, your Product programs WILL be able to use the newer and faster Rockey4ND Keys.