



# **KGen – UDF Resolver & Sanity Checker**

## ***User Guide***

Version 3.4h

## **KGen – Version 3.4h**

### **Trademarks**

This product employs components covered under Public Domain or GPU licenses. These components are included freely with the installation suite as a convenience. It is strongly recommended that the current versions of these components be downloaded and their licenses reviewed before use.

KiXtart	<a href="http://www.kixtart.org">www.kixtart.org</a>	Administrative scripting tool
KixForms	<a href="http://www.kixforms.org">www.kixforms.org</a>	GUI interface for KiXtart

This documentation may contain references or links to third-party web sites. These sites are not under the control of Innovative Technology or its principals, and Innovative Technology is not responsible for their content. Access to any third-party website or facility is performed solely at the user's discretion and risk.

### **Copyright**

This application and its documentation is Copyright © 1995-2016, Glenn Barnas. All rights reserved. This document may not be altered or reproduced without the express written permission of Glenn Barnas.

### **Restricted Use**

This product can be used freely for personal or commercial use so long as all copyright notices remain in place. No part of this product or documentation can be sold or used in the sale of any commercial product or service without the express written agreement of Glenn Barnas.

### **Acknowledgements**

Thanks to the following people for their support, feedback, and suggestions for making this product the best it can be:

NTDoc, Shawn, and all the rest of you that took the time to provide feedback and suggestions.

## **KiXtart Development**

The KiXtart Development Suite consists of the KiXtart executable, the KGen linker tool, a library of common User Defined Functions (UDFs), and related documentation. The latest version of the KiXtart executable can be downloaded from the KiXtart web site at <http://www.kixtart.org>. If you need to develop for multiple KiXtart versions, we recommend that you download the needed versions and name them Kix32\_4.xx.exe. This will support version-specific tokenizing through the KGen tool. It is recommended that you maintain copies of Kix32 in this folder with a consistent naming format as new versions are released.

### ***KiXtart Function Library***

The KiXtart function library is a folder on your computer (or a network share) that contains general purpose functions that extend the capabilities of the core programming language. Each UDF is maintained in its own file within the library, although certain collections of related UDFs can be combined into a single file. An example of this is the XLib, which contains several individual UDFs that interface with Microsoft Excel.

The use of a function library serves to standardize the coding of KiXtart-based scripts. It also speeds development of new tools by providing a suite of commonly needed functions in a central repository.

The use of the library is enhanced through the use of the *KGen* tool. This program reads the script source file and generates a script file containing the core code and any additional functions needed to complete the script. It locates the missing UDFs from the KixLib folder as well as the folder where the core code is maintained. This allows you to more easily maintain a large script project by breaking it into smaller, more manageable segments.

### ***KGen tool***

The *KGen* tool is an advanced KiXtart script generator. It automatically scans the source file for all UDF dependencies, locates the UDFs, and then resolves dependencies within the UDFs themselves. The result is a portable script file that contains all needed UDFs. There is no reliance on external UDF files, the script is fully self-contained!

KGen is a command-line tool. You simply open a command prompt, CD to your project folder, and invoke KGen. You can specify the source filename on the command line, or use a BUILD.INI file to specify the project parameters.

### **Installation & Configuration**

The first step is to create a development environment on your computer. While the tools and library are fairly flexible, it is a good idea to create a structure that is easy to manage. Generally, we create a folder – Dev – that is the root of our development system. This can be placed anywhere, but a network share allows many people to access the facilities. In the Dev folder, create a KixLib folder to hold the UDF files, and then create at least one project folder.

By default, the installation package creates a LIB folder in the installation root (C:\Program Files\ITCG) and copies the UDF files to that location. If you create a network UDF library, you can enter that path during the installation process.

Run the installer by extracting the ZIP file to a folder, then, from a command prompt, navigate to that folder and type SETUP KIXDEV.

## Getting Started

The best way to see the power of KGen is to take a test drive. The install media has a sample project included, but you must manually copy it to your dev folder. Navigate to the project folder called “samples”. In that folder you should find the file OSI.KIX.

Run the script – kix32 OSI.kix. What happens?

Since “OSInfo” isn’t a valid KiXtart command, so it will generate an error. You could locate the OSInfo UDF, copy the text into your test script and try again, but it would still fail, because the OSInfo UDF relies on yet other UDFs – TimeDiff and Memory.

Time to try KGen!

Rename “osi.kix” to “osi.src”. Run “KGEN OSI” at the command prompt and watch the output. It should report the number of UDFs located, the number of UDF files examined, and an indication that generation of test.kix is complete.

Examine the contents of your project folder. Where you had a single Test.src file, you now have eight files!

- osi.src - your original script source
- osi.kix - the newly generated script
- osi.log - a summary of what UDFs were used to build the script
- osi.rpt - a report of possible script errors
- osi.gen - the script in text format, with all comments intact
- codemap.txt – a map of the paired functions (such as If/EndIf)
- Warnings.csv - similar to the rpt file, can be opened and formatted with Excel
- VarInfo.ini - an INI file that reports variable usage, by UDF

View the .KIX file – you should see all of the needed UDFs to run the command. At this time, the .KIX and .GEN files are identical. The .GEN files are useful when certain options are invoked, such as tokenizing or stripping comments.

KGen accepts two optional argument types – *ProjectID* and *OutputOptions*.

***ProjectID*** can be the name of the file that contains your script source. It *must* have a .TXT extension. It can also be the name of a project section if you are using a *build.ini* file – more on this later. For large projects, you can create as many .UDF files in the project folder as you need in order to simplify code management. KGen will automatically include all of the .UDF files in the project folder into the final script. (If you require them in a specific order, use a 2 digit prefix on each file.) The .GEN file is built starting with the .TXT file, followed by the project .UDF files, and finally any external library .UDF files.

***OutputOptions*** control what KGen does to your script. You can specify any of the following:

- S** - Strips comments from the resulting file. Strip is ignored when tokenizing is active. You may specify “S” or “STRIP” to invoke this action.
- T** - Toggles the current tokenization state. If the default is “N” and the build file is “Y”, this parameter will turn tokenization off. If the default is “Y” and the build file is “N”, this will turn tokenization on. When tokenization is enabled, it tokenizes the resulting script using the default version of Kix32.exe. See the parameter below to use a specific Kix version to tokenize with.  
  
The complete script is stored in a .GEN file so an untokenized/unstrapped copy is always available.
- T:V.vv** - Tokenizes the script by invoking Kix32\_V.vv.exe. This allows you to specify which version of Kix32 to use to create the tokenized file. This might be necessary to allow a script to run with mixed versions of Kix. Currently, scripts tokenized with 4.53 or 4.60 will not always function with later versions. The properly named Kix32.exe file must be available for this to work properly.
- D** - Enables debug mode. Reports on many internal variables and their values after parsing input and preparing to begin the generation process. Using this parameter twice will cause KGen to terminate after displaying the values.
- S** - Suppress the SANITY process. You may specify “-S” or “-Sanity”.
- Flush** - Immediately flushes the KGen UDF cache, causing a new cache to be generated the next time KGen is run. You may specify “F” or “Flush” to invoke this option. When this command is used, KGen exits immediately after clearing the cache file.
- Help** - Displays basic usage information.

The OuputOptions will always override the default settings as well as settings in the Build.ini file (if used).

As of version 3.4, the command line arguments can be used to override the Build.ini settings. Prior to this, command line arguments were valid only when specifying a source file (*project.txt*).

KGen has internal defaults for the above options. Since KGen is delivered in source-code form, you can change these defaults if you desire. The standard defaults are to not strip comments, not tokenize the output, and to run Sanity after generating the script.

The settings in the Build.ini file will override the program defaults, and creating a standard Build.ini that is copied for each new project is the recommended way to override the defaults. For example, our standard build file forces tokenization on.

The arguments specified on the command line take precedence over both the internal default values as well as those specified in the build.ini file. For example, tokenization is off by default, turned on in the Build.ini file, but running “KGen T” will turn it back off.

## Advanced KGen Configuration

KGen will run without any arguments if a *build.ini* file is placed in your project folder. This file defines the project settings, and KGen will take its three optional arguments (Strip, Tokenize, & Sanity) from this file as well as several advanced options. This file can be created with any text editor, such as Notepad, and has the following format:

```
[Project]
Name=My Test Project
Base=test
Extn=.kix
Dirs=
Strip=n
Tokenize=y
TokenizeWith=4.61
Lib=.\special
NoPublicLibs=y
```

The *build.ini* file contains at least one section that is called “PROJECT”, with any of the keys below.

- *Name* defines the project descriptive name – not used by *KGen* at this time, but may be used by future GUI management tools
- *Base* will define the project base file. This allows calling *KGen* without specifying the name of the project. This must be a .TXT file
- *Extn* will override the default *.kix* file extension. This is useful if you use the *.kxw* extension for KiXforms scripts. The extension “.BAT” is also supported. When this is defined, *KGen* automatically disables tokenization and adds special commands to the beginning and end of the script. This allows you to distribute the BAT file with a *Kix32.exe* file. Running the BAT file invokes *Kix32.exe scriptname.bat*. This eliminates the need to have a batch file to invoke *Kix32* or the need to have *Kix32* in your path. Of course, to be readable as a batch file, the script cannot be tokenized. *KGen* itself uses this Kix-BAT format!
- *Dirs* defines a list of folders that the completed script will be copied to. The individual destinations are separated by semicolons (“;”).
- *Strip* will remove all comments if defined. Leaves a commented version of the script with a .GEN file extension. Used to somewhat obfuscate the source.
- *Tokenize* – supported with KiXtart 4.5 and later – will create a tokenized version of the script. The resulting file is no longer text readable, and must be used with the same version of KiXtart that was used to tokenize it. The *Tokenize* parameter is mutually exclusive with the *Strip* parameter, and takes precedence if both are specified in the *build.ini* file. You can specify the use of a specific Kix version to perform the tokenization with *TokenizeWith-#.#.#*, where *#.#.#* represents the specific version *kix32.exe* file that is named “*Kix32\_#.#.#.exe*”
- *Lib* – specifies additional UDF library paths, delimited with “;”, that are used with this project in addition to any libraries specified by the *KixLibPath* environment variable. Often used to specify an additional private library folder.
- *IncFiles* – A list of UDF files to include. Note that the entire file specification must be used. Compare this to the *KGEN:INCLUDE:udflist* directive, which specifies particular functions to include.

- *NoPublicLibs* – disables the use of external UDF libraries. Only UDFs in the project folder or specifically included will be used.
- *Sanity* – A Y/N option controlling whether the Sanity process will be performed.

The default section must be called “PROJECT”. You can create multiple sections within the Build.ini file by creating alternate sections. Specify the section name when you invoke KGen. One use of this is to have a section that does not tokenize the script, while another section tokenizes the script and copies it to a deployment folder, or several sections that each specify a different Kix32 version to use for tokenizing the script.

## KGen Directives

You can embed directives in the script source file to control how KGen operates. Each KGen directive begins with a semicolon, so that the KiXtart script treats it like a comment. The directive must be the first text on the line, and there cannot be any space between the comment character and the directive. Each directive starts with “;KGEN:” and is followed immediately by the directive.

NOPARSE	Do not parse the remainder of the source file. This is often placed at the beginning of UDF library files or files containing only KiXforms form layouts that have no external UDF references. This speeds processing.
INCLUDE:fn[,fn...]	Forces the named function <i>fn</i> to be included even if it is not found in the current project. This is useful in modular code, where you might include utility functions in the main module and exclude them from the secondary modules. Compare this with the IncFiles parameter in build.ini, which can be used to include specific files.
INCLUDHERE:file	Replaces the line with the contents of the file specified by “file”. This is used to include code at a specific location - common code that is not a function. This file is NOT parsed for dependencies, but is processed via Sanity.
NOVERSIONHEADER	Prevents the Version Validation logic from being inserted into the tokenized application. This is usually done when writing code modules that load live into a running application.
INSANE	Suppress variable checking in Sanity - used when building modules that reference variables defined in the main application.
NORECURSION	Turns off recursive parsing. When this is used, recursive parsing of external UDF files is disabled for the entire project, from the point where the directive was encountered. This is often used in large, modular scripts, and in conjunction with the INCLUDE directive.

NOPUBLICLIBS	Does not include any functions from the public library folders (%KIXLIBPATH%). This is a global directive.
IGNORENEXT	Tells the Sanity process to ignore the next line, which might be valid code but report a warning. This is commonly done when breaking strings over multiple lines or when an EXEC command has a variable inside of a string that it needs to evaluate.
NOPARSE	Prevents parsing of the code that follows. Used at the top of UDF "libraries".

To best understand how these directives work, apply them to your test.txt file and run KGen, observing the output of the .KIX and .LOG files.

## Sanity

When KGen completes assembly of your script, it passes the entire script through the Sanity logic. This reports several potential problems, including Variables declared but not used, Variables used but not declared, Variables declared as both local and global, Mismatched quotes, Mismatched parenthesis, and Variables use in strings. In addition to this basic syntax and variable checking, it also reports when paired functions are unbalanced, including If/EndIf, While/Loop, Do/Until, Select/EndSelect, For/Next, and Function/EndFunction.

The items reported are not necessarily errors – it is up to you to determine that you have properly written your code! For example, if you have a quoted string that spans 2 lines, you will receive two “Mismatched Quotes” warning messages.

## UDF Library Caching

Introduced in version 3.4, UDF Library Caching improves performance by building and maintaining a cache of the UDF to File associations. Since build and tokenize operations happen more often than adding UDFs to the common library (or libraries), this saves 3-5 seconds per build operation. This may not sound like a lot, but when the build/tokenize process takes less than 2 seconds and Sanity takes 8-10 seconds for a larger script, it becomes a significant part of the operation.

The cache is refreshed automatically every 7 days and updates you with a “creating cache” message. A “Flush” option is available in KGen to clear the cache. This should be invoked when you add, rename, or remove UDFs from the library. Clearing the cache is not necessary when you update a UDF, as long as the UDF name and the file it resides in doesn't change.

When KGen proceeds, it will see the cache is empty and rebuild it. The cache file is stored in %TEMP%\KGenCache.ini, and thus will usually be unique to each user. The cache file is fairly small, 12-15K for a library with 250 UDFs. Deleting this cache file will have the same effect as clearing the cache from within KGen.

## Tokenizing Notes

Tokenization is *supposed to be* backward compatible. That is, a script tokenized with Kix version 4.61 should be able to run properly using Kix version 4.64. This seems to work for simple scripts, but experience has shown that this is not always the case for larger or



more complex scripts. When a script tokenized with one version of Kix is run with a different (later) version, it may fail with a rather strange error. It is “strange” because it references a command and line number, but that command may not even be present in your script, and certainly isn’t at the indicated location if it does exist in your script.

Our experience has been that it is best to tokenize and run scripts using the same Kix version. To that end, KGen automatically inserts a header with code that compares the Kix version running against the one it was tokenized with and exits with an error if the versions don’t match.

The use of this header code can be controlled by setting the \$UseVersionHeader variable in the KGen script. There are three options available for \$UseVersionHeader:

- 0 – Disabled    The tokenized script will not be restricted from running with any Kix32 version, including prior versions that do not support tokenizing. Running a tokenized script on downlevel Kix32 versions can have unexpected results or total script failures. This setting is not recommended.
- 1 – Loose        The script will be permitted to run if the Kix32 version used to run the tokenized script is greater than or equal to the version used to tokenize it.
- 2 – Strict        The script will be permitted to run only if the Kix32 version used to run the tokenized script exactly matches the version used to tokenize it. This is the default setting.

### ***Source Filenames***

The original version of KGen looked for a text file with a .txt extension. When Notepad++ arrived, we wanted to differentiate between simple text files opened with Notepad and source code files opened with Notepad++. We switched to using a .SRC file extension as the default, but KGen will still recognize the .TXT extension as a valid source code file.

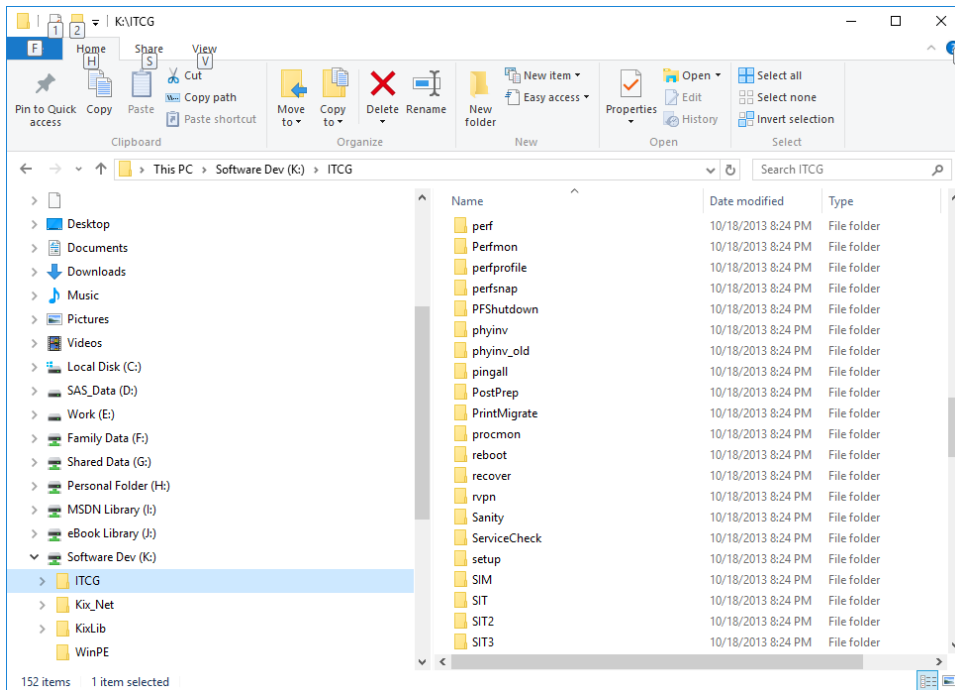
## Managing Large Projects

KGen has been used to manage script projects in excess of 10,000 lines of code. For such projects, the following guidelines might prove useful.

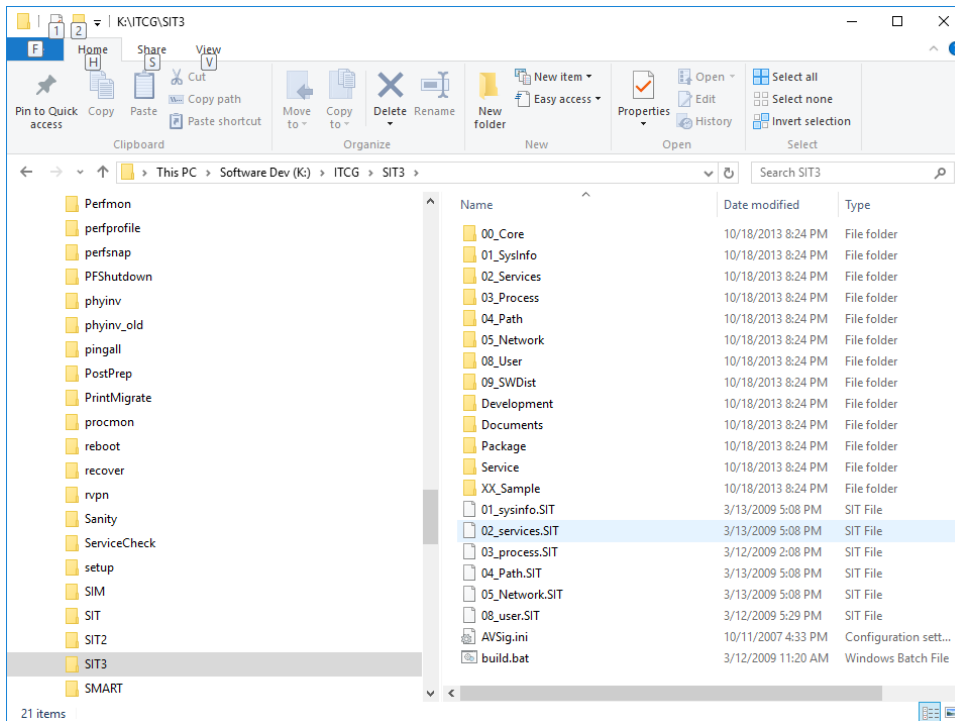
- Minimize the information in your *project.src* file. It will be the first file placed in your output script, so it should contain your header, comments, and common variable declarations and initializations, but little if any actual code.
- Use a Global \$VERSION and define your version string in your *project.txt* file. This will make it easy to maintain update notes and version number changes in one place.
- Break the code into small, manageable parts, stored in .INC or .UDF files in the project folder. Use standard filenames for these project modules to assemble them into a logical order. For example, these names identify the project, sequence, and basic content of each file:
  - BigProject.src           Core code with no functions
  - BigProject.inc           Core functions
  - BP00\_init.udf
  - BP04\_parse.udf
  - BP08\_interface.udf
  - BP99\_misc.udf
- When using KiXForms, separate large form constructs into separate files. Use a .KXF extension to identify them as forms – KGen will recognize that extension as well. If you have a form file and related code file, give them the same file name, with .KXF for the forms and .UDF for the code. They will be included together by KGen.
- Leverage the *build.ini* file! The DIRS= parameter can copy your new script to many different locations at once. For example, if you have a script that runs on your Exchange servers to generate statistics, the DIRS parameter can list the UNC path of every Exchange server. You can even specify a full filespec, which means you can copy the new script to a different location AND name!
- Use multiple sections for different project configurations. For example, the default project might not tokenize the script but will run Sanity. This will be used for development testing. Additional sections might tokenize with different versions of Kix without the need to run Sanity, creating Kix Version-specific production scripts. One of our projects uses this to create two completely different packages from the same core code base.

## A Picture is Worth 1000 Words...

Here's a screenshot of the development share, showing the root folder (with the KixLib folder and our private development folder and several of our project folders.

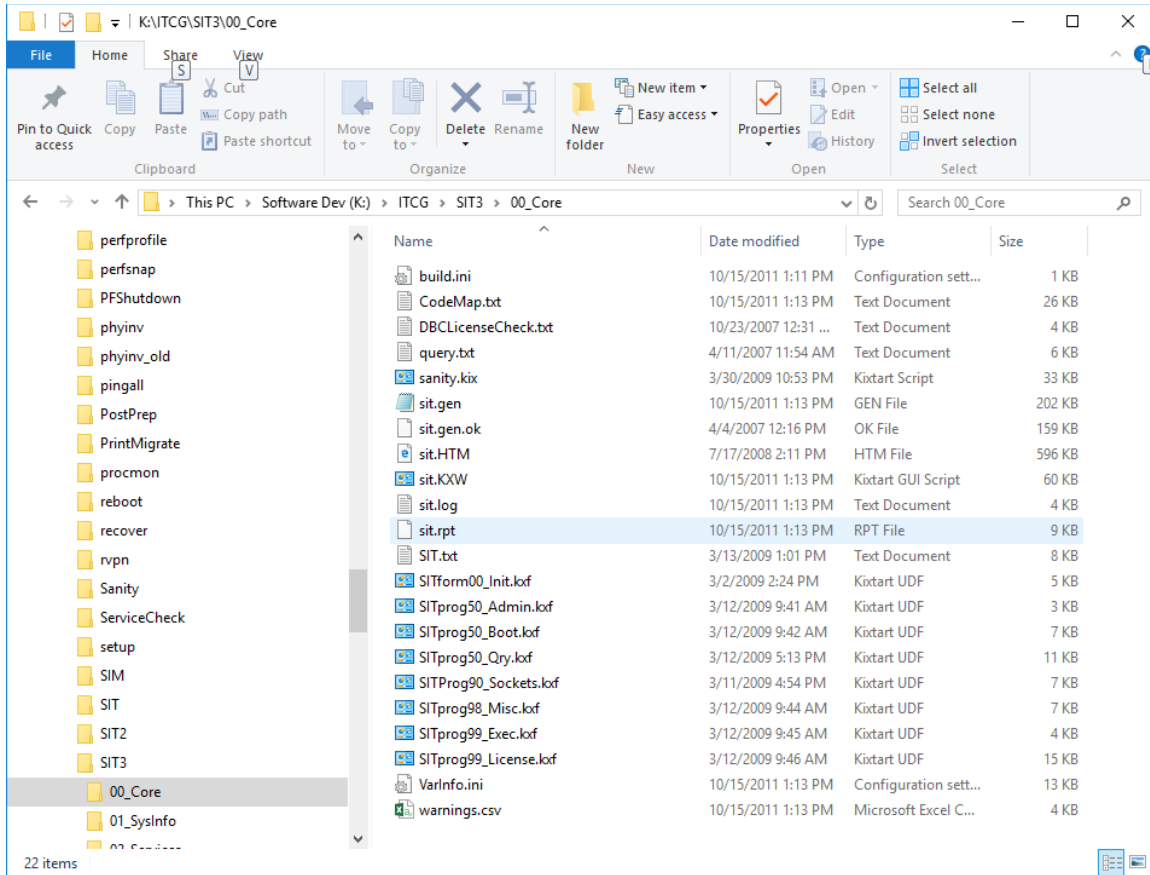


Here's a project folder for our System Interrogation Tool – a help-desk app:



This is a complex package with many components in separate subfolders. A build.bat is used to run KGen in each folder, and Build.ini copies the resulting script back to this folder. Development of individual modules can be carried out by multiple team members.

One last shot of one of the component subfolders – this shows how SIT.txt is used as the primary file, and several .KXF files make up the rest of the core script code. The largest file is 15K and contains all of the licensing logic – most of the remaining components are under 10K, making each file easier to manage and edit. If the file were not broken into separate sections, the core script (without library UDFs) would exceed 75-Kbytes.



Note the SIT.GEN file – which represents the script and all library UDFs in editable source form. This file is helpful if you want to dynamically test/edit the code during debugging sessions. The SIT.KXW file is the tokenized script – you can see that this reduces the size of the file significantly, from over 200K to just 60K.

This folder has a few extra files from debugging tasks and these can be ignored (DBCLicenseCheck.txt, query.txt, sanity.kix, and sit.gen.ok). Of note is the SIT.HTM file, with is a colored HTML representation of the complete source file and was generated by our PostPrep script.

This folder also contains several log files generated by KGen:

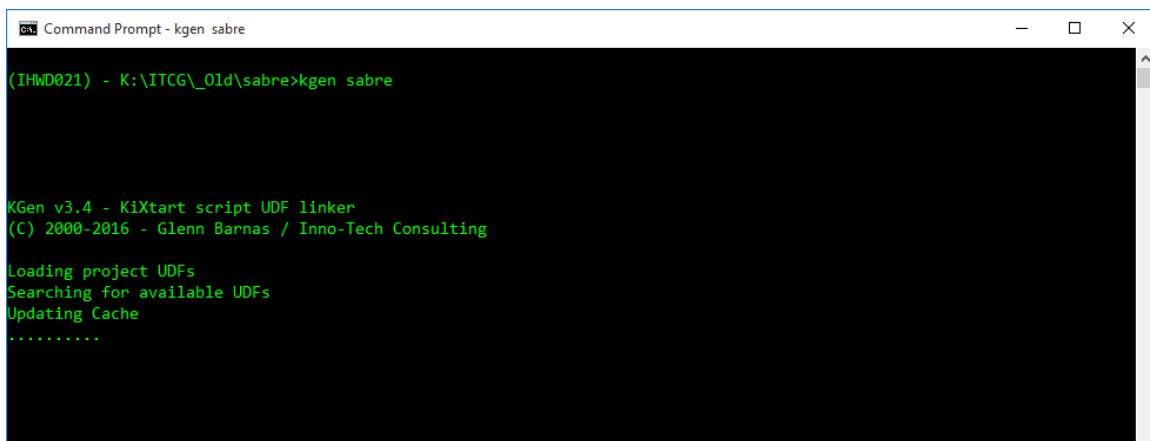
- VarInfo.ini – a table of every variable name, by function, the line on which it was declared, and the line on which it was first referenced.
- Warnings.csv – a summary report of all warnings generated by the KGen process.
- CodeMap.txt – a listing of all paired functions (such as If/EndIf) reporting the line where they begin and end. Each is indented by two spaces, allowing a quick visual check of pairings. Unmatched pairs reported by Sanity are so noted in this file. Each function has a separate section in this file.

- Sit.Log – a report of the recursive evaluation of all functions and the files that were loaded as part of the script generation. The file’s name is based on the project base name.
- A detailed report of all warnings, by line number. This duplicates the messages displayed on the screen while KGen is running.

Not shown here is a new log – KGen.log – which duplicates the screen output of KGen, with the exception of the Sanity report and on-screen progress indicators.

## Command Progress

The following 3 screenshots show running KGen against a source file with a blank cache. Note the project UDFs are loaded, then the UDFs from the library are indexed and the cache updated.



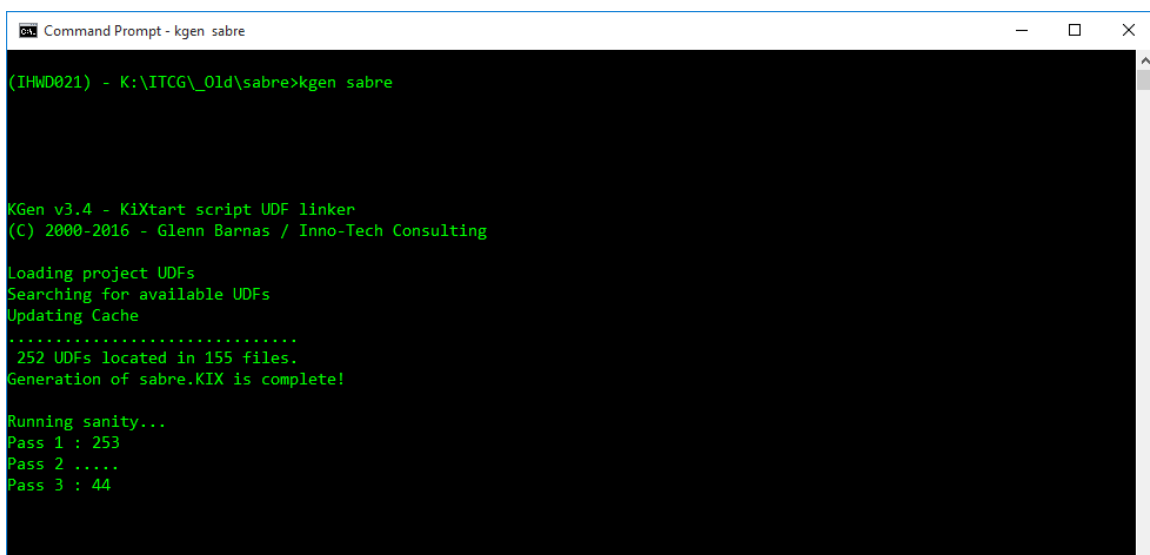
```
Command Prompt - kgen sabre

(IHWD021) - K:\ITCG\_01d\sabre>kgen sabre

KGen v3.4 - KiXtart script UDF linker
(C) 2000-2016 - Glenn Barnas / Inno-Tech Consulting

Loading project UDFs
Searching for available UDFs
Updating Cache
.....
```

A total of 252 UDFs were indexed from 155 files. The script has been generated and the Sanity process has started. There are 5 passes, and passes 1, 3 and 5 report each line number as they are examined. Passes 2 and 4 are overall scans and don’t provide line number data.



```
Command Prompt - kgen sabre

(IHWD021) - K:\ITCG\_01d\sabre>kgen sabre

KGen v3.4 - KiXtart script UDF linker
(C) 2000-2016 - Glenn Barnas / Inno-Tech Consulting

Loading project UDFs
Searching for available UDFs
Updating Cache
.....
252 UDFs located in 155 files.
Generation of sabre.KIX is complete!

Running sanity...
Pass 1 : 253
Pass 2 : .....
Pass 3 : 44
```

```
Command Prompt

KGen v3.4 - KiXtart script UDF linker
(C) 2000-2016 - Glenn Barnas / Inno-Tech Consulting

Loading project UDFs
Searching for available UDFs
Updating Cache
.....
 252 UDFs located in 155 files.
Generation of sabre.KIX is complete!

Running sanity...
Pass 1 : 253
Pass 2 .....
Pass 3 : 253
Pass 4 .
Warning: Variable declared but not referenced.
      Variable Name: $TData
      In function: Main
      Referenced on line: 8
      Declaration line is referenced above.
....
Pass 5 : 252
 1 warnings generated, 253 lines processed.

(IHWD021) - K:\ITCG\Old\sabre>
```

Sanity has completed and has generated one warning – a variable was declared but was not referenced in the completed script. This could be an oversight, a plan to use it in the future (we often KGen as we develop to catch issues early!), or it might be part of a dynamically loadable module. KGen can't read minds, so it's up to you to review these warnings and know which can be ignored.

## Support

KGen is free to download and use, although no official support is provided. Support is offered on an as-time-permits basis through email to [support@barnas.us](mailto:support@barnas.us) or through the Kixtart.org web site.

Please review this document, the readme.txt file, and review the sample files before asking for assistance.

## Error Codes

This is a simple listing of Windows error codes and the corresponding error message that they represent. This is presented for convenience in coding, particularly for implementing useful Exit codes.

0	The operation completed successfully.
1	Incorrect function.
2	The system cannot find the file specified.
3	The system cannot find the path specified.
4	The system cannot open the file.
5	Access is denied.
6	The handle is invalid.
7	The storage control blocks were destroyed.
8	Not enough storage is available to process this command.
9	The storage control block address is invalid.
10	The environment is incorrect.
11	An attempt was made to load a program with an incorrect format.
12	The access code is invalid.
13	The data is invalid.
14	Not enough storage is available to complete this operation.
15	The system cannot find the drive specified.
16	The directory cannot be removed.
17	The system cannot move the file to a different disk drive.
18	There are no more files.
19	The media is write protected.
20	The system cannot find the device specified.
21	The device is not ready.
22	The device does not recognize the command.
23	Data error (cyclic redundancy check).
24	The program issued a command but the command length is incorrect.
25	The drive cannot locate a specific area or track on the disk.
26	The specified disk or diskette cannot be accessed.
27	The drive cannot find the sector requested.
28	The printer is out of paper.
29	The system cannot write to the specified device.
30	The system cannot read from the specified device.
31	A device attached to the system is not functioning.
32	The process cannot access the file because it is being used by another process.
33	The process cannot access the file because another process has locked a portion of the file.
34	The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.
36	Too many files opened for sharing.
38	Reached the end of the file.
39	The disk is full.
50	The network request is not supported.
51	The remote computer is not available.

52 A duplicate name exists on the network.  
53 The network path was not found.  
54 The network is busy.  
55 The specified network resource or device is no longer available.  
56 The network BIOS command limit has been reached.  
57 A network adapter hardware error occurred.  
58 The specified server cannot perform the requested operation.  
59 An unexpected network error occurred.  
60 The remote adapter is not compatible.  
61 The printer queue is full.  
62 Space to store the file waiting to be printed is not available on the server.  
63 Your file waiting to be printed was deleted.  
64 The specified network name is no longer available.  
65 Network access is denied.  
66 The network resource type is not correct.  
67 The network name cannot be found.  
68 The name limit for the local computer network adapter card was exceeded.  
69 The network BIOS session limit was exceeded.  
70 The remote server has been paused or is in the process of being started.  
71 No more connections can be made to this remote computer at this time  
because there are already as many connections as the computer can accept.  
72 The specified printer or disk device has been paused.  
80 The file exists.  
82 The directory or file cannot be created.  
83 Fail on INT 24.  
84 Storage to process this request is not available.  
85 The local device name is already in use.  
86 The specified network password is not correct.  
87 The parameter is incorrect.  
88 A write fault occurred on the network.  
89 The system cannot start another process at this time.  
100 Cannot create another system semaphore.  
101 The exclusive semaphore is owned by another process.  
102 The semaphore is set and cannot be closed.  
103 The semaphore cannot be set again.  
104 Cannot request exclusive semaphores at interrupt time.  
105 The previous ownership of this semaphore has ended.  
106 Insert the diskette for drive %1.  
107 The program stopped because an alternate diskette was not inserted.  
108 The disk is in use or locked by another process.  
109 The pipe has been ended.  
110 The system cannot open the device or file specified.  
111 The file name is too long.  
112 There is not enough space on the disk.  
113 No more internal file identifiers available.  
114 The target internal file identifier is incorrect.  
117 The IOCTL call made by the application program is not correct.  
118 The verify-on-write switch parameter value is not correct.



- 119 The system does not support the command requested.
- 120 This function is not supported on this system.
- 121 The semaphore timeout period has expired.
- 122 The data area passed to a system call is too small.
- 123 The filename, directory name, or volume label syntax is incorrect.
- 124 The system call level is not correct.
- 125 The disk has no volume label.
- 126 The specified module could not be found.
- 127 The specified procedure could not be found.
- 128 There are no child processes to wait for.
- 129 The %1 application cannot be run in Win32 mode.
- 130 Attempt to use a file handle to an open disk partition for an operation other than raw disk I/O.
- 131 An attempt was made to move the file pointer before the beginning of the file.
- 132 The file pointer cannot be set on the specified device or file.
- 133 A JOIN or SUBST command cannot be used for a drive that contains previously joined drives.
- 134 An attempt was made to use a JOIN or SUBST command on a drive that has already been joined.
- 135 An attempt was made to use a JOIN or SUBST command on a drive that has already been substituted.
- 136 The system tried to delete the JOIN of a drive that is not joined.
- 137 The system tried to delete the substitution of a drive that is not substituted.
- 138 The system tried to join a drive to a directory on a joined drive.
- 139 The system tried to substitute a drive to a directory on a substituted drive.
- 140 The system tried to join a drive to a directory on a substituted drive.
- 141 The system tried to SUBST a drive to a directory on a joined drive.
- 142 The system cannot perform a JOIN or SUBST at this time.
- 143 The system cannot join or substitute a drive to or for a directory on the same drive.
- 144 The directory is not a subdirectory of the root directory.
- 145 The directory is not empty.
- 146 The path specified is being used in a substitute.
- 147 Not enough resources are available to process this command.
- 148 The path specified cannot be used at this time.
- 149 An attempt was made to join or substitute a drive for which a directory on the drive is the target of a previous substitute.
- 150 System trace information was not specified in your CONFIG.SYS file, or tracing is disallowed.
- 151 The number of specified semaphore events for DosMuxSemWait is not correct.
- 152 DosMuxSemWait did not execute; too many semaphores are already set.
- 153 The DosMuxSemWait list is not correct.
- 154 The volume label you entered exceeds the label character limit of the target file system.
- 155 Cannot create another thread.
- 156 The recipient process has refused the signal.

157 The segment is already discarded and cannot be locked.  
158 The segment is already unlocked.  
159 The address for the thread ID is not correct.  
160 The argument string passed to DosExecPgm is not correct.  
161 The specified path is invalid.  
162 A signal is already pending.  
164 No more threads can be created in the system.  
167 Unable to lock a region of a file.  
170 The requested resource is in use.  
173 A lock request was not outstanding for the supplied cancel region.  
174 The file system does not support atomic changes to the lock type.  
180 The system detected a segment number that was not correct.  
182 The operating system cannot run %1.  
183 Cannot create a file when that file already exists.  
186 The flag passed is not correct.  
187 The specified system semaphore name was not found.  
191 Cannot run %1 in Win32 mode.  
192 The operating system cannot run %1.  
193 %1 is not a valid Win32 application.  
196 The operating system cannot run this application program.  
197 The operating system is not presently configured to run this application.  
199 The operating system cannot run this application program.  
200 The code segment cannot be greater than or equal to 64K.  
203 The system could not find the environment option that was entered.  
205 No process in the command subtree has a signal handler.  
206 The filename or extension is too long.  
207 The ring 2 stack is in use.  
208 The global filename characters, \* or ?, are entered incorrectly or too many  
global filename characters are specified.  
209 The signal being posted is not correct.  
210 The signal handler cannot be set.  
212 The segment is locked and cannot be reallocated.  
214 Too many dynamic-link modules are attached to this program or dynamic-  
link module.  
215 Cannot nest calls to LoadModule.  
216 The image file %1 is valid, but is for a machine type other than the current  
machine.  
230 The pipe state is invalid.  
231 All pipe instances are busy.  
232 The pipe is being closed.  
233 No process is on the other end of the pipe.  
234 More data is available.  
240 The session was canceled.  
254 The specified extended attribute name was invalid.  
255 The extended attributes are inconsistent.  
258 The wait operation timed out.  
259 No more data is available.  
266 The copy functions cannot be used.

267 The directory name is invalid.  
275 The extended attributes did not fit in the buffer.  
276 The extended attribute file on the mounted file system is corrupt.  
277 The extended attribute table file is full.  
278 The specified extended attribute handle is invalid.  
282 The mounted file system does not support extended attributes.  
288 Attempt to release mutex not owned by caller.  
298 Too many posts were made to a semaphore.  
299 Only part of a ReadProcessMemory or WriteProcessMemory request was completed.  
300 The oplock request is denied.  
301 An invalid oplock acknowledgment was received by the system.  
317 The system cannot find message text for message number 0x%1 in the message file for %2.  
487 Attempt to access invalid address.  
534 Arithmetic result exceeded 32 bits.  
535 There is a process on other end of the pipe.  
536 Waiting for a process to open the other end of the pipe.