



**Inno-Tech**  
Consulting Group LLC



*Bringing Innovation & Technology Together*

**tcLib**

**A KiXtart UDF Library for  
Controlling the Windows Task Scheduler**

***Programmer's Reference Guide***

Version 2.0

January, 2004

Updated April 2005, December 2007, November 2009

## **tcLib – Version 2.0**

### **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
JT.exe	<a href="http://www.microsoft.com">www.microsoft.com</a>	Task Scheduler command line tool

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 © 2004-2011, Glenn Barnas. All rights reserved. This document may not be altered or reproduced without the express written permission of Innovative Technology Consulting Group.

### **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 Innovative Technology Consulting Services.

### **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.

## **tcLib.kxf**

### **– a KiXtart UDF for controlling Windows Task Scheduler**

tcLib.kxf is a library of user-defined functions for the KiXtart scripting language. It provides an interface to Microsoft's *jt.exe* resource kit tool – an advanced replacement for the *at* command. The functions provide the ability to create, modify, and delete tasks on local or remote systems, all under script control.

The functions work under the following assumptions:

- The term “task” is used to describe what command will be run, and what system parameters will control how it will be run. Command-related parameters would define the command, its parameters, the working directory, and possibly a comment. System parameters would control when a task could (or could not) start, such as when the system is idle, running on battery power, or when the user is logged on.
- The term “trigger” describes when the task will be run. Information such as start time, start and end dates, and how often the task should run is kept in the trigger. A task can have multiple triggers associated with it, allowing it to run using multiple configurations. The TcLib library arbitrarily allows up to 10 triggers to be associated with a task, although this can be easily modified.
- An “event” is a collection of a task and all of its related triggers. Any function that affects both the task and its triggers is referred to as an “event”.

TcLib allows the programmer to work with multiple events simultaneously. Task and trigger settings are maintained in a pair of arrays, and multiple array sets can be easily managed. Event settings can be defined programmatically or extracted from an existing event. This allows events to be quickly modified or even copied between systems.

## ***Changes in Version 1.1***

- Changes to the return values  
There are two formats for return values – Boolean True/False, or ErrorMessage/Nothing. If the function performs parameter validation, it will exit with Error 87, and return a text message indicating which parameter check failed. If the parameters are valid, then no value is returned, and the @ERROR macro should be checked to insure proper completion. For the functions that check for existence of tasks or triggers, or perform deletions, the functions exit with a true/false value. The @ERROR macro should be checked if the returned value is not true. In prior versions the value returned was “1” instead of nothing. This could have led to false positives when a parameter error message was returned.
- Use of WSH to invoke JT  
This release uses WSH to capture the output of JT.exe, instead of using the Shell command. This eliminates the creation of temporary files and improves performance.
- Interpretation of JT error messages  
JT did not exit with a proper error code, but instead wrote the errors to its output stream. These messages are now properly captured and evaluated.
- Addition of tcErrMsg function  
Interprets & displays specific Task Scheduler error messages.

## ***Changes in Version 1.2***

- Use of KixForms.dll to invoke JT.EXE when the DLL is registered. If Kixforms is not available, WSH is used instead. This change enhances the display when invoked with WKix32.exe by preventing the command window running the JT command from appearing. When running a command-line script, the output (if any) is in the primary console window, so no change is apparent. When running a KF GUI script, WSH can cause a brief appearance of a command window, which is suppressed by running the command via KixForms.DLL.
- Code standardization – use of “\$\_” variable names, individual variable declaration and explanations.

## ***Changes in Version 2.0***

- Implementation of “positive logic” return values.
- Elimination of text returned from functions.
- Performance enhancements to parameter parsing.
- Use of Array of Arrays for managing trigger data instead of a 2-dimensional array. While this change is fairly significant, user code rarely, if ever, directly manipulates the trigger arrays, and changes are simple, from a \$a\_tcTRIG[X,Y] format to \$a\_tcTRIG[X][Y].
- Addition of tcLoad and tcSave functions to create INI files for task data.
- Change to tcInit to return a single initialized trigger array.

## **Summary of Functions**

**tcLibVer()**

**tcInit ( [nullflag] )**

**tcDefineTask(“TaskParam=value ...”)**

APP	ApplicationName	PRM	Parameters
WKD	WorkingDirectory	CMT	Comment
CTR	Creator		
USR	User ID	PWD	Password

**tcDefineTrigger (TriggerID, “TrigParam=value ...”)**

SDT	StartDate	EDT	EndDate
STM	StartTime	DUR	MinutesDuration
INT	MinutesInterval	HED	HasEndDate
KAD	KillAtDuration	DIS	Disabled
TYP	Type	ARG	TypeArguments

**tcSetEventCredentials(Host, TaskName, userid, password )**

**tcSetEvent(Host, TaskName)**

**tcGetEvent(Host, TaskName)**

**tcDelEvent(Host, TaskName)**

**tcDelTrigger (Host, TaskName, Trigger\_ID)**

**tcIsTask(Host, TaskName)**

**tcIsTrigger(Host, TaskName, Trigger\_ID)**

**tcGetTasks(Host)**

**tcExecute(Host, TaskName)**

**tcTerminate(Host, TaskName)**

**tcLoad(IniFile, IniSection)**

**tcSave(IniFile, IniSection)**

**tcErrMsg(ERR#)**

**tcLibVer()**

## Function Descriptions

### ***tcInit ( [flag] )***

tcInit() starts by declaring all global variables the first time it is called. It initializes two Global arrays - \$a\_tcTASK and \$a\_tcTRIG - each time it is called. These hold the task and trigger data that comprise an event. If no argument is provided or its value is zero, the arrays are initialized with default data, including flags for required fields. This is appropriate when defining new tasks. If the *flag* parameter is specified as “1”, the arrays are initialized to nulls, as appropriate for receiving information from an existing event. When the parameter supplied is “2”, the function returns an array of initialized data that can be used to clear or prepare a Trigger array group.

The value “RQ” is used to indicate a required parameter not yet defined by the user.

tcInit() *must* be called prior to invoking any other tcLib function, with the exception of tcLibVer.

#### **Usage:**

tcInit() or tcInit(*n*)

#### **Default \$a\_tcTASK Array Values:**

<b>Element</b>	<b>Parameter Name</b>	<b>Default Value</b>
0	ApplicationName	“RQ”
1	Parameters	<NULL>
2	WorkingDirectory	null
3	Comment	“Task created with tcLib”
4	Creator	@USERID (current user)
5	Priority	NORMAL
6	MaxRunTime	3600000
7	Idle	10 60
8	DontStartIfOnBatteries	0
9	KillIfGoingOnBatteries	0
10	RunOnlyIfLoggedOn	0
11	SystemRequired	0
12	DeleteWhenDone	0
13	Suspend	0
14	StartOnlyIfIdle	0
15	KillOnIdleEnd	0
16	RestartOnIdleResume	0
17	Hidden	0
18 *	Interactive	0
19 *	HaltOnError	0
20	User ID	Default
21	Password	Default

\* Although these values are defined in *jt.exe* documentation, they do not function and are not used at this time.

### Default \$a\_tcTRIG Array Column Values:

Element	Parameter Name	Default Value
0	StartDate	<current date>
1	EndDate	<NULL>
2	StartTime	<next hour>
3	MinutesDuration	0
4	MinutesInterval	0
5	HasEndDate	0
6	KillAtDuration	0
7	Disabled	0
8	Type	“RQ”
9	TypeArguments	“RQ”
10	Defined	“F”

There are (by default) ten sets of trigger arrays, resulting in a 10x11 element matrix. The number of triggers is limited by the \$tcMAXTRIGGERS value that is set in the tcInit function.

Element 10 – Defined – is used internally to indicate that the trigger has been defined. This speeds processing when creating task events or enumerating events read from the scheduler.

### Configuration Parameters

All parameters that are user configurable are defined within the tcInit function. The values are only defined the first time that tcInit is called, allowing the user application to easily override them. All parameters are declared as Global. The default parameters rarely need to be changed. The parameters include:

- \$tcOUTPUT “NUL:” to suppress output of the JT command. Can be set to “CON:” or a filespec to view or save the JT command line. This is useful for debugging, or to capture the JT command for embedding in another application.
- \$tcJTEXE “JT.EXE” which assumes that the command can be found via the system PATH variable. It can be coded to a specific location.
- \$tcMAXTRIGGERS 10 – the number of triggers that can be processed.
- \$tcWORKDIR “C:\” – the default “Working Directory” for task events. This is usually changed per task, but cannot be null.
- \$tcCOMMENT “Created with tcLib version #.#” is the default comment when a task event is created. This is usually changed when defining the task.

In all cases, it is recommended that these values be defined in your script AFTER the first call to tcInit, rather than changing the values in the library.

## ***tcDefineTask("Mnemonic=value ...")***

Loads the Task array with the values specified by the TaskParm=value parameter pairs. The parameter pairs are implemented as a set of three-character mnemonics to represent all 20 parameters supported by jt.exe. This array can be manipulated directly if desired. This function provides a method to set and validate the parameters in one step.

### **Usage:**

```
tcDefineTask('APP=kix32 PRM=$arg==42 CMT=test script')
```

Note that "=" characters must be escaped with "==". The function will exit with error 87 if an invalid mnemonic is used. If tcDefineTask is called before tcInit, an error 2 will be thrown, indicating that the task array is not initialized.

### **Parameters:**

<b>Element</b>	<b>Mnemonic</b>	<b>Parameter Name</b>	<b>Description</b>
0	APP	ApplicationName	Drive:path\file of executable
1	PRM	Parameters	Executable parameters
2	WKD	WorkingDirectory	Drive:path – where to run
3	CMT	Comment	Free-form comment
4	CTR	Creator	userID that created the event
5	PRI	Priority	IDLE,NORMAL,HIGH,REALTIME
6	MRT	MaxRunTime	Time (in ms) task is allowed to run
7	IDL	Idle	# # format of Idle Minutes & Idle Retry Minutes
8	DSB	DontStartIfOnBatteries	Boolean (0/1)
9	KGB	KillIfGoingOnBatteries	Boolean (0/1)
10	RLO	RunOnlyIfLoggedOn	Boolean (0/1)
11	SRQ	SystemRequired	Boolean (0/1)
12	DWD	DeleteWhenDone	Boolean (0/1)
13	SUS	Suspend	Boolean (0/1)
14	SII	StartOnlyIfIdle	Boolean (0/1)
15	KIE	KillOnIdleEnd	Boolean (0/1)
16	RIR	RestartOnIdleResume	Boolean (0/1)
17	HID	Hidden	Boolean (0/1)
20	USR	UserID	User account to run task under
21	PWD	Password	Password for above account

*Elements 18 & 19 are not supported by JT.EXE.*

### ***tcDefineTrigger(TriggerID, "Mnemonic=value ...")***

Loads the Trigger array with the values specified by the Mnemonic=value parameter pairs. The parameter pairs are implemented as a set of three-character mnemonics to represent all 9 parameters supported by jt.exe. TriggerID must be in the range 0-9 (or the value defined by \$tcMAXTRIGGERS).

#### **Usage:**

```
tcDefineTrigger(0, `TYP=WEEKLY ARG=1,U..... STM=23:59)
If @ERROR
    SERROR ?
EndIf
```

This trigger will run every Sunday at 23:59.

#### **Parameters:**

<b>Element</b>	<b>Mnemonic</b>	<b>Parameter Name</b>	<b>Description</b>
0	SDT	StartDate	<date> when trigger starts
1	EDT	EndDate	<date> when trigger ends
2	STM	StartTime	<hh:mm> when trigger runs
3	DUR	MinutesDuration	<mmm> how long trigger is active when repeating a task
4	INT	MinutesInterval	<mm> how often a task should be repeated
5	HED	HasEndDate	Boolean – 1 to activate EndDate
6	KAD	KillAtDuration	Boolean – 1 if task should be killed it is still running at the end of the duration.
7	DIS	Disabled	Boolean – 1 to disable trigger
8	TYP	Type	Defines the trigger type – see below
9	ARG	TypeArguments	Depends on the type parameter

Both tcDefineTask and tcDefineTrigger can be called multiple times to set all of the required parameters. The programmer can make one call with all necessary parameters, or one call per parameter as they see fit.

**Type** and **TypeArgument** must be one of the following:

DAILY	n	Runs the task every <i>n</i> days. "n" is an integer specifying the days interval between executions, a value of "9" would cause this trigger to execute every 9 days.
WEEKLY	n, WD	Runs every <i>n</i> weeks on the specified days. "n" is an integer specifying the weeks interval between executions, and WD specifies the days on which it should be run.
MonthlyDate	n, months	Runs on the specified day of the month in the specified months. "15,JANAPRJULOCT" will run on the 15 <sup>th</sup> , every 3 months.
MonthlyDOW	n, WD, months	Runs on the day of the specified week of the month, in the specified months. "n" is an integer in the range 1-5, represent the first, second, third, fourth, or last week of the month. "5, .....F., MARJUNSEPDEC" runs on the last Friday of every quarter.
ONCE	<i>none</i>	Runs once at the specified day/time.
ONIDLE	<i>none</i>	Runs when the system is idle
ATSTARTUP	<i>none</i>	Runs when the system boots
AT LOGON	<i>none</i>	Runs when the user logs on.

**WD** has the format "UMTWRFA" to represent the days of the week. Days where the trigger should not run should use a period as a placeholder. ".M.W.F." would define Monday, Wednesday, and Friday.

**Months** is a string composed of the first three letters of each month name. No spaces are permitted. To define every month with 30 days, use "SEPAPRJUNNOV". Sequence and capitalization are not important.

**Notes:**

Any call to this function sets the Defined flag (element 10) to "T".

The function will exit with error 87 if an invalid mnemonic is used. It will exit with error 2 if the tcInit function was not called to initialize the arrays.

***tcSetEventCredentials("Host", "TaskName", "userid", "password")***

Sets the security credentials for the event on the target host. This is useful for updating existing tasks with new account credentials. No information is stored locally, and the local Task array is not used or modified.

Returns a Boolean indicating the status of the function.

**Usage:**

```
If tcSetCredentials("host", MyTask", "scheduser", "sc3dU1#")
    `Credentials were successfully updated.' ?
Else
    `Credential update failed!' ?
EndIf
```

## ***tcSetEvent(Host, TaskName)***

Defines a scheduled event on the selected host, using the parameters in the global Task and Trigger arrays.

The function verifies that all required parameters are defined – the *Command* parameter in the task, and the *Type* and *TypeArgument* parameters in each configured trigger. (A “configured” trigger is one that has element 10 set to “T”.)

It then checks the scheduler on the target host to determine if the task exists. If it does, it prepares to edit the task, otherwise it will create the task. Next, it checks for each configured trigger in the array and determines if it should create or edit the corresponding event trigger on the target host.

A command is then issued to create and/or modify the event on the target system. The programmer does not need to distinguish between create and edit operations.

### **Usage:**

```
If tcSetEvent($Host, $TName)
  'Successfully created event' ?
Else
  If @ERROR = 87
    @SERROR ?
  Else
    'Creation of event failed:' ?
    tcErrMsg(@ERROR) ?
    $RV = tcDelevent($Host, $TName)
  EndIf
EndIf
```

## ***tcGetEvent(Host, TaskName)***

Fills the global event arrays with the task and trigger information obtained from the specified task on the target host. At most, 10 triggers (or the number defined by \$tcMAXTRIGGERS) are returned from the event.

Event status values are returned in Task array elements 22-26. These include (in order) MostRecentRun, NextRun, StartError, ExitCode, and Status. It is up to the user to query these elements directly as they are not used internally by the tcLib library.

An error is returned if the named task does not exist.

### **Usage:**

```
; Tasks is an array of task names returned from tcGetTasks()
For Each $Task in $Task
  If tcGetEvent($Host, $Task)
    If $a_tcTASK[20] = $UserID
      'Found task with specific user ID: ' $Task ?
    EndIf
  Else
    tcErrMsg(@ERROR) ?
  EndIf
Next
```

### ***tcDelEvent(Host, TaskName)***

Deletes the entire event – task and all triggers – from the specified host. No error checking is performed to insure that the task exists.

Any corresponding task or trigger arrays are unaffected.

#### **Usage:**

```
If tcDelEvent("host", "MyTask")
  'Event was deleted!' ?
Else
  'Event delete failed!' ?
  tcErrMsg(@ERROR) ?
EndIf
```

Deletes MyTask and all triggers from the host if it exists.

### ***tcIsTask(Host, TaskName)***

Determines if the named task exists on the specified host. Returns a “1” if it exists, “0” if it does not.

#### **Usage:**

```
If tcIsTask("host", "MyTask")
  'Task exists!' ?
Else
  If @ERROR 2
    'Task does not exist!' ?
  Else
    tcErrMsg(@ERROR) ?
  EndIf
EndIf
```

The sample determines whether a task exists or not. If the task is not present, it verifies that the returned error is 2 (not found). If it is not “2”, a different error occurred and the message is displayed using the tcErrMsg function.

### ***tcIsTrigger(Host, TaskName, Trigger\_ID)***

Determines if the specified trigger is configured in the task on the specified host. Returns a “1” if it exists and a “0” if it doesn't.

#### **Usage:**

See the tcIstask example shown above.

### ***tcGetTasks(Host)***

Fills an array with the names of tasks from the specified host. No confirmation is made of the status of the tasks.

#### **Usage:**

```
$aryTaskList = tcGetTasks("host")
```

A one-dimension array is loaded with the names of the tasks defined on the host.

### ***tcDelTrigger(Host, TaskName, Trigger\_ID)***

Deletes the trigger *trigger\_id* from the specified task *taskname* on the host. No error checking is performed to insure that the specified task / trigger exists.

The corresponding trigger array is not modified. If the corresponding trigger in the array should be deleted, element 10 of that array should be set to "F".

#### **Usage:**

```
If tcDeleteTrigger("host", "MyTask", 3)
  `Trigger was deleted!' ?
Else
  `Trigger delete failed!' ?
  tcErrMsg(@ERROR) ?
EndIf
```

The third trigger in "MyTask" will be deleted if it exists.

### ***tcExecute(Host, TaskName)***

Immediately executes the specified task *taskname* on the host. No error checking is performed to insure that the specified task exists. The programmer should use existing UDFs to determine the existence of the task as well as its completion status. The exit code only identifies the success or failure of the UDF's ability to initiate the task.

#### **Usage:**

```
If tcExecute("host", "MyTask")
  $RV = tcGetEvent("host", "MyTask")
  If $a_tcTASK[25] <> 0
    `Task failed to run!' ?
    tcErrMsg($a_tcTASK[25]) ?
  EndIf
EndIf
```

The task "MyTask" will be executed on "host" if it exists.

### ***tcTerminate(Host, TaskName)***

Immediately terminates the specified task *taskname* on the host. No error checking is performed to insure that the specified task exists or is running. The programmer should use existing UDFs to determine the existence of the task as well as its current status. The exit code identifies the success or failure of the UDF's ability to terminate the task.

#### **Usage:**

```
If tcTerminate("host", "MyTask")
  `Task execution was terminated successfully' ?
Else
  tcErrMsg(@ERROR) ?
EndIf
```

The task "MyTask" will be terminated on "host" if it is running.

### ***tcErrMsg(Err#)***

JT.exe will return special error codes specific to the task scheduler process. These special codes can be intermixed with standard error codes such as Not Found or Access Denied. The tcErrMsg function is a substitute for the @SERROR macro for displaying error messages from JT.EXE.

When called with @ERROR as an argument immediately following a tcLib function call, it will determine if the error code is a standard or Task Scheduler error. If it is a standard code, it will use @SERROR to return the error message, otherwise, it will perform a lookup and return the specific Task Scheduler error message.

#### **Usage:**

```
$RV = tcSetEvent(host, taskname)
If @ERROR
    tcErrMsg(@ERROR) ?
EndIf
```

### ***tcLoad(file, ID) & tcSave(file, ID)***

Loads the event arrays from an INI file, or saves the event array data to an INI file. This provides a fundamental export/import capability. User credentials are *never* exported or imported. The file can contain several task events, with each section ID potentially representing a task name.

To maintain compatibility with the tsAdm (Task Scheduler Administrator) tool, only one task should be defined per file, the file should have a .EVT extension, and the ID should be "EVENT".

### ***tcLibVer()***

Returns the version string of the tcLib package. Introduced in version 1.2.

## How to Use tcLib functions

**IMPORTANT NOTE:** If you are using an electronic copy of this document, be aware that you *cannot* simply copy/paste these examples into your script! MS-Word changes dashes and quotes to typographical characters that will not work in scripts. You must replace these with the standard ASCII single or double quotes, or dash characters. Refer to the sample script file for examples.

### Basics:

Start by calling tcInit() to initialize the task and trigger arrays. These global arrays, called \$a\_tcTASK and \$a\_tcTRIG, are created automatically the first time tcInit() is invoked. Other global parameters are also defined during the initial invocation of tcInit.

Plan on using an event name. We'll use TestEvent in this example.

Initialize the arrays with default values prior to any read or create process:

```
tcInit()
```

Next, load the arrays with the parameters for your task. We'll run "kix32 test,kix \$X=4" every Mon, Wed, and Fri at 4pm. While we're at it, let's set the task credentials. Note how tcDefineTask can be called multiple times, each invocation setting different data elements.

```
$RV = tcDefineTask("APP=kix32 PRM=test.kix $X==4 CMT=my test")
$RV = tcDefineTask ("USR=username PWD=password")
$RV = tcDefineTrigger(0, "STM=16:00 TYP=WEEKLY ARG=1,.M.W.F.")
```

Note that tcDefineTrigger can be called multiple times as well, if desired.

Delete the task if it already exists on our test host.

```
If tcIsTask("host", "TestEvent")
  If Not tcDelEvent("host", "TestEvent")
    'Delete failed' ?
  Else
    'task was deleted' ?
  EndIf
Else
  'TestEvent task did not exist.' ?
EndIf
```

Finally, send the information to the host and activate the event.

```
If tcSetEvent("host", "TestEvent")
  "Event was successfully created!" ?
Else
  "Failed to create event!" ?
  tcErrMsg(@ERROR)
EndIf
```

The above example shows how to check the @ERROR macro. All of the functions set this value when they exit. A "zero" value indicates success.

Get a list of events on a target system

```
$TaskNames = tcGetEvents("host")
```

## ***Special Notes***

Certain situations require that a task not have a password defined. Most often, this is when a task is used to interact with a specific user. The Trigger should employ a RLO=1 parameter, to Run Only If Logged On, the USR parameter should specify the user account, and the PWD should use the keyword “NULL”.

## ***More Ideas for tcLib***

### **Change the password for the account used for scheduled tasks:**

- Enumerate a list of computer names.
- For each computer, invoke tcGetTasks(computer)
- Enumerate the array returned by tcGetTasks, calling tcGetEvent for each task.
- Check element 20 (UserID) – if the account name matches, then call tcSetEventCredentials(computer, taskname, user, password) to update the password.

We were able to update the scheduled tasks on over 400 servers across 5 data centers, with an average of 8 scheduled events per server in just under 3 hours. The time estimate to perform this by other methods was 40+ hours and involved several staff members! The key point to recognize is that by enumerating the tasks on each machine, and then checking the user account defined in the task, only tasks using the affected account are updated.

### **Change the parameters for an existing task**

- Use tcGetEvents to obtain a list of events from a computer.
- Enumerate the array of task names. If the event you need to change is present, obtain the current parameter set via tcGetEvent(host, eventname)
- Use tcDefineTask and/or tcDefineTrigger to modify the parameters
- Use tcSetEvent(host, eventname) to write the modified data back to the computer. You will need to include the PWD parameter, since the password is never read from the configured task!

### **Don't forget to look at our tsAdm – Task Scheduler Administrator!**

This is a Kixtart/Kixforms GUI application that employs tcLib and provides a ready to use interface for creating, editing, running, and even mass-deploying tasks to Windows-based computers.