

ScanStream

Reference Manual

Table of Contents

Introduction	1
The ScanStream Main Window	2
ScanStream Configuration	5
Errors in ScanStream.....	8
Programming ScanStream	8
Errors in ScanStream Scripts.....	11
Sample Scripts That Work With ScanStream	12
Appendix A – SendKeys Codes	17
Appendix B – Installing ScanStream	18
Appendix C – Using ScanStream with Floormation	19

Introduction

Target Audience

This manual is written for Process Engineers who will be setting up and maintaining site configurations on the ScanStream system, and for site personnel who want a greater understanding of ScanStream. Since this is a reference manual, it contains very little in the way of step-by-step instruction. Instead, this manual attempts to document the features and structures of ScanStream's component parts.

General Overview of ScanStream

ScanStream is a software wedge with custom scripting for serial port data input. A software wedge is an application that takes input from the PC's serial port and wedges the data into the keyboard input of the PC. To the receiving application, the keystrokes appear to have been entered via the keyboard.

ScanStream can be used to fill in fields in any Windows application and provides data filters and sorting. It can also fill in standard date and user fields in addition to the data that comes through the serial port. ScanStream can be used to fill out virtually any form via bar code scans.

ScanStream stores all of its configuration data in two .ini files. One, Scanner.ini provides some of the selections for configuring ports in ScanStream. The other, ScannerPC.ini identifies the COM port configurations as entered in ScanStream. See ScanStream Configuration section for more details regarding these ini files.

An important note: Any COM ports that have been configured on the PC with ScanStream as device type 'scanner' are locked exclusively when the application is running. This means that other applications cannot communicate with those COM ports until the ScanStream program is exited.

Once ScanStream is configured for a PC, it runs by itself in the background. Since ScanStream runs seamlessly in the background, a bar code icon is displayed in the tool tray while the program is running.

The ScanStream Main Window

Purpose

The major purpose of the ScanStream main window is to provide users with a tool for configuring ScanStream. The main window lists the ports currently configured for the PC, provides the ability to configure ports, allows setting/editing the script file used when scanning, and allows viewing/deleting the error logs.

Description

When ScanStream is opened, it automatically hides itself (after displaying a small splash screen). This is because the main window in ScanStream is only used for configuration, and the program runs seamlessly in the background after that.

Once the program is running, there are three ways to display ScanStream's main window:

1. Double clicking on the bar code icon in the tool tray;
2. Right-click on the bar code icon in the tool tray, then click on Show Config option;
3. Invoke the hot key (defaults to ALT-F12).

Changes made in the main window are not applied until the window is closed.

Port configurations should match those of the scanner (not the PC). Most scanners are set to a baud rate of 9600, even parity, 7 data bits and 2 stop bits. Refer to the documentation for your scanner if these settings do not work.

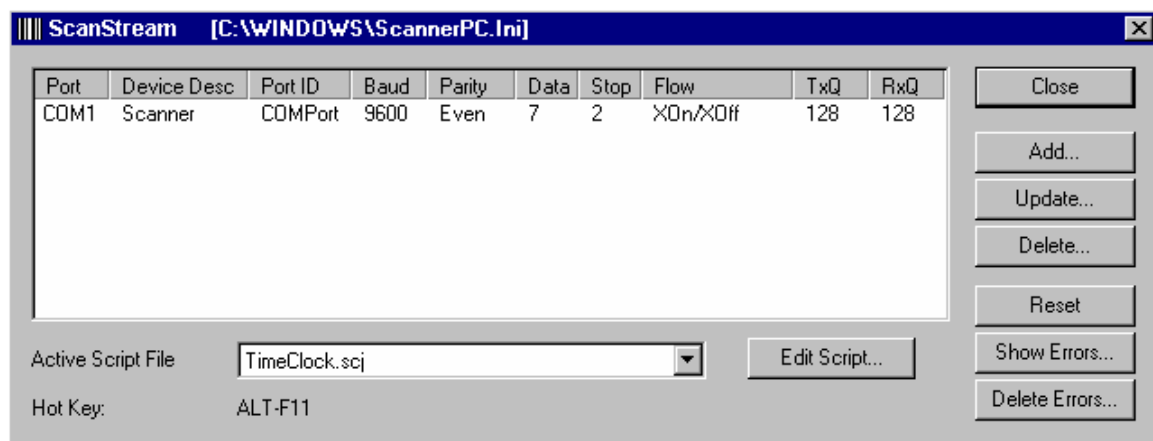


Figure 1 – ScanStream Main Window

ScanStream

Each item on the main window is described below:

Title: The title bar displays the name of the application followed by the location of the ScannerPC.ini file that is currently in use by the application. In Figure 1, the location of the ScannerPC.ini file is C:\WINDOWS.

Close: This button minimizes ScanStream's main window, but does not exit the application. The Close button also applies any changes that have been made in ScanStream's main window.

Add...: This button invokes the Port Config window and allows users to add a port configuration for the PC. Defaults are displayed in the Port Config window for all the settings. The default settings are hard coded into the ScanStream executable as follows:

Device Type	Scanner
Port ID	COMPort
Baud Rate	9600
Parity	Even
Data Bits	7
Stop Bits	2
Flow Control	None
Tx Queue	128
Rx Queue	128

Different defaults can be specified in the [Defaults] section of the Scanner.ini file (refer to the ScanStream Configuration section for more details).

Update...: This button invokes the Port Config window for the currently selected port and allows users to edit the port configuration. Select the port to update by clicking on it in the Port column, and then clicking the update button.

Delete ...: This button deletes the port configuration for the currently selected port. Select the port to delete by clicking on it in the Port column, and then clicking the delete button.

Reset: The reset button causes ScanStream to direct its focus to the last active application. It does not re-read the ScannerPC.ini file. Resetting ScanStream is required any time the active application changes (to something other than ScanStream).

Show Errors...: This button brings up the Scanner.Log file in NotePad. ScanStream appends to this file whenever an error occurs in the application. If the file does not exist, ScanStream creates the log file. The log file resides in the ScanStream 'home' directory (usually C:\Program Files\ScanStream).

ScanStream

Delete Errors ...: This button deletes the Scanner.log file from the PC. A confirmation prompt is displayed prior to deletion.

Active Script File: The name of the currently active script that is used for reading in data scanned is displayed here. The most commonly used script is 'Simple.asf'. Any files found in ScanStream's home directory with extensions of .asf .scj or .svc will be displayed in the drop down box. The .asf and .scv extensions indicate that it is a VB Script ®, while the .scj extension indicates the script is written in JavaScript ® format. Either format will work with ScanStream. There can only be one script active at a time (not one per port configuration). The file extension must be in lower case letters and either scj, scv or asf to be visible in the Active Script drop down list.

Edit Script: This button brings up the active script file in NotePad for editing. It is recommended that a back up copy be made of the script before editing. Changes to the script are not reflected until the script is reloaded into memory. See the Programming ScanStream section for further details on reloading the active script.

Hot Key: The hot key is another way of displaying ScanStream's main window. In Figure 1, the hot key is specified as ALT-F12. This is extremely useful for those PCs that do not have a working mouse.

Definitions:

Port: Specifies which port on the PC the configuration applies to. ScanStream verifies the Port starts with COM and is followed by a number.

Device Type: Describes what kind of gadget the port configuration is for (e.g. Scanner, Zebra Printer, etc.). The options shown here when updating or adding are taken from the [Devices] section of the Scanner.ini file.

Port ID: The PortID is what ScanStream uses to determine the function name to invoke (from the active script) when a bar code is scanned. The default is COMPort. For a PortID of COMPort, the main function in the active script must be named COMPort_OnInput.

Baud Rate: The baud rate setting for the port configuration. Baud is a unit of measure of transmission speed for binary-coded data. Baud rate usually indicates the number of bits per second. ScanStream baud rate options range from 110 to 256000.

Parity: The parity setting for the port configuration. In binary-coded data, parity is used in error-detecting and error-correcting codes. The parity setting indicates

ScanStream

if the total number of 1s or 0s should be always odd or always even. Often 7 bits are used to represent each character and 1 bit is use as a parity check bit. ScanStream parity options are None, Odd, Even, Mark or Space.

Data Bits: The data bits setting for the port configuration. Indicates the length of bits that contain information, as opposed to bits used for starting, stopping, or error checking. ScanStream data bit options are 4,5,6,7 or 8.

Stop Bits: The stop bits setting for the port configuration. Indicates the length of the bit that signals the end of a unit of transmission on a serial line. ScanStream stop bit options are 1, 1.5, or 2.

Flow Control: The flow control setting for the port configuration. Flow Control is the control of transmission between communications devices, to make sure the sender does not send data until the receiver is ready to receive it. Flow control may be achieved by means of hardware or software. ScanStream flow control options are None, Hardware or XOn/XOff (software).

Tx Queue: The number of bytes to set the internal Windows buffer that is allocated for queueing output that cannot be sent immediately. ScanStream performs secondary buffering until space is available, so there is no problem sending a 10 MB file through a COM port with a 256 byte Tx Queue.

Rx Queue: The number of bytes to set the internal Windows buffer that is allocated for queueing input that arrives faster than ScanStream can process it.

ScanStream Configuration

ScanStream stores all of its configuration data in two initialization files. One, Scanner.ini is stored in 'home' directory for ScanStream. The home directory is the location where the ScanStream executable resides (usually C:\Program Files\ScanStream). Scanner.ini provides some of the selections for configuring ports in ScanStream. The other configuration file, ScannerPC.ini, is stored in the C:\Windows directory on the PC, and identifies the COM port configurations as entered in ScanStream.

Both initialization files follow the standard Windows ini file format: information in the files is grouped into sections, identified by square brackets ([]). Each section contains a set of keys, each of which is assigned a value. A comment line in a ScanStream initialization file is indicated by two consecutive semi-colons (;).

Scanner.ini

The Scanner.ini file is stored in the ScanStream 'home' directory. It contains two sections that define variables for use by ScanStream: [Devices] and [Defaults].

ScanStream

[Devices]

The device section specifies the gadgets that appear in the Device Type drop down list for port configurations. Following is the format of a device entry:

```
Device Name = Device Type:Device Options
```

Device Name:

The Device Name is a description of the device. For the device types ZPL/ZPLII, the description of the device must include the corresponding file extension, enclosed in parentheses and prefixed with a star wildcard (*). No spaces are allowed inside the parentheses. The extension can occur anywhere in the description, although the Windows convention is to place it at the end:

```
Device Name (*.zpl)
```

Device Type:

The Device Type specifies whether the device is a scanner or a printer. Allowable device types are:

SCANNER	Standard input/output device.
ZPL	Support for the Herma ZPL emulation.
ZPLII	Any modern zebra.

Any other device type causes the port to not be usable by ScanStream.

Device Options:

For printer device types, you can specify the following additional options to control the printer interface:

:NOSTATUS	Causes the printer interface to not request status (but it will still send clear graphics directives).
:NOCLEAR	Causes the printer interface to never interact with the printer; it will blindly send label formats with no error checking, etc.

Example:

Following is an example of the devices section of the Scanner.ini file:

```
[Devices]
Unassigned/Reserved = NONE
Scanner = SCANNER
ZEBRA, 200dpi (*.z2n) = ZPLII
HERMA, 200dpi (*.z2n) = ZPL:NOCLEAR
PAGO, 200dpi (*.z2n) = ZPLII:NOCLEAR
```

ScanStream

TEST ONLY!!! (*.z2n) = ZPL:NOSTATUS:NOCLEAR

[Defaults]

The defaults section specifies the gadgets that appear in the Device Type drop down list when adding port configurations. This section is optional in the Scanner.ini file. The Device Name as specified in the [Defaults] section must be listed in the [Devices] section for it to show up when adding a port. Following is the format of a default entry:

```
Defaults = Device Type,PortId,Baud Rate,Parity,Data  
Bits,Stop Bits,Flow Control,Tx Queue,Rx Queue,Device  
Name
```

Example:

Following is an example of the defaults section of the Scanner.ini file:

```
[Defaults]  
Defaults=SCANNER,COMPort,9600,Even,7,2,None,128,128,  
Scanner
```

ScannerPC.ini

The ScannerPC.ini file is stored on the PC in the C:\Windows directory. It identifies the COM port configurations as entered in ScanStream. There are two sections in the ScannerPC.ini file: [General] and [PortParams].

[General]

The general section specifies some useful information regarding the last (or current) session of ScanStream running on the PC. There are three (3) key/value entries that reside in the general section: Location, ActiveScript, and Hotkey. The Location entry specifies the storage location of the ScanStream executable. The ActiveScript entry specifies the VBScript® or JavaScript® file that is currently configured for ScanStream on the PC. The Hotkey entry specifies the hotkey that is currently configured for ScanStream on the PC. A hotkey is a combination of keys that activate the ScanStream main window (i.e. shortcut). If no hotkey is specified, the default is ALT-F12.

Example:

```
Location=C:\ASPEN\SCANNER  
ActiveScript=Simple.asf  
Hotkey=ALT-F11
```

ScanStream

[PortParams]

The PortParams section specifies the port configurations that are currently configured for ScanStream on the PC. Following is the format of the PortParams section:

```
Port Name = Device Type,Port ID,Baud Rate,Parity, Data  
Bits,Stop Bits,Flow Control,Tx Queue,Rx Queue,Device  
Name
```

Example:

```
[PortParams]  
COM1=SCANNER,COMPort,9600,Even,7,2,None,128,128,Scanner  
COM2=ZPLII,CUSHIONLABEL,9600,Even,7,2,XOn/XOff,128,128,ZEBR  
A,200dpi (*.z2n)
```

Errors in ScanStream

If there is a configuration error, the ScanStream icon in the tool tray will flash red. Bring up the ScanStream main window and click on the Show Errors button.

The error log is called Scanner.log and resides in the ScanStream 'home' directory. Each error in the log file is separated by a row of asterisks (*) followed by a date/time stamp.

Some errors display message prompts with an indication of the problem (instead of flashing the tool tray icon red). Error registering Hotkey is one of these types (this error will occur if ScanStream is already running and it is started again).

If there is no response in the active application that the scanner is working, check the port configuration settings. Most scanners are configured with a baud rate of 9600, even parity, 7 data bits and 2 stop bits.

Programming ScanStream

ScanStream can be easily programmed to perform a variety of tasks. The most common task is to send all input to the active application as keystrokes. More complex tasks such as editing scanned input into multiple inputs or adding prefixes and suffixes to scanned data are easily programmed using JavaScript® or VB Script®.

ScanStream

The script file extension must be in lower case letters and either scj for JavaScript® or either asf or scv for VB Script®.

If there is an Active Script specified in the ScanStream main window, it is loaded into memory when starting ScanStream or when changing the active script (and clicking the Close button). It is important to notice that editing and saving the script does **not** reload it into memory.

The easiest way to reload the script while developing/testing is as follows:

1. Bring up the ScanStream main window.
2. Edit script and save changes via the Edit Script button.
3. Change the Active Script to a known working script and click the Close button.
4. Bring up the ScanStream main window and change the Active Script back to the one being developed/tested and click the Close button.
5. Scan something to test the script.

The main function in the script **must** be named according to the PortId followed by '_OnInput'. For example, if the PortId is COMPort, the main function of the active script must be COMPort_OnInput.

There are three (3) built-in methods available for use in ScanStream scripts: WritePort, SendKeys, and Sleep. When using one of these built-in methods, the format must be the device name followed by a period followed by the method name.

Example:

The following example sends 'ABC' and the enter/return key to the active application.

```
Scanner.SendKeys('ABC' & "{Enter}")
```

WritePort Method

The WritePort method allows data to be sent to the port specified.

Parameters: PortName, data

Return Value: true on success, false on failure

SendKeys Method

The SendKeys method sends data to the active application on the PC as keystrokes. Each key is represented by one or more characters. To specify a single keyboard character, use the character itself. For example, to represent the

ScanStream

letter A, use "A". To represent more than one character, append each additional character to the one preceding it. To represent the letters A, B, and C, use "ABC".

The keys {ALT}, {CTRL} and {SHIFT} act as modifier keys and their effect lasts only until the first non-modifier key is sent. For example, when accessing the File | Save menu of a typical Windows application, the menu accelerator is Alt-f followed by an unmodified (no Alt key) s. This key sequence can be sent using `sysSendKeys()` as:

```
Scanner.SendKeys("{ALT}fs");
```

To send Alt-Shift-F9, you would use:

```
Scanner.SendKeys("{ALT}{SHIFT}{F9}");
```

When using upper-case and lower-case letters in the send-keys string, the function automatically inserts {SHIFT} modifiers for each upper-case letter that it encounters. For example, to send the word *ScanStream*, both of the following examples send the same identical keystrokes:

```
Scanner.SendKeys("ScanStream");  
Scanner.SendKeys("{SHIFT}Scan{SHIFT}Stream");
```

Additionally, when using the modifier keys, you will typically specify a lower case letter to be modified, since that is what most applications expects. For example, "{ALT}fx" is typically typed by a user to access the menu shortcut Alt-f-x whereas, "{ALT}FX" would generate the key sequence Alt-Shift-f-Shift-x.

To cause a key to be repeated n times, use the syntax {key n}. There must be a single space between the key name and the repeat count n. For example, to send seven tab keys to an application, you would use:

```
Scanner.SendKeys("{TAB 7}");
```

To send the digit '1' to an application ten times, you would use:

```
Scanner.SendKeys("{1 10}");
```

To specify characters that aren't displayed when you press a key, such as ENTER or TAB, and keys that represent actions rather than characters, use the codes shown in Appendix A.

Parameters: data

Return Value: 0

ScanStream

ScanStream

Sleep Method

The Sleep method causes execution of the current script to be suspended for the specified number of milliseconds.

Parameters: delay (in milliseconds)

Return Value: none

Errors in ScanStream Scripts

If there is an error in the active script, depending upon the error it may occur when loading the script and/or when running the script. If the error occurs when loading the script, the ScanStream icon in the tool tray will flash red. To determine the problem, bring up the ScanStream main window and click on the See Errors button. The last error in the file is probably the one that caused the problem. Scripting errors will display the line number and script error.

Example:

```
*****
03/07/01 11:46:03

Error starting script:
  C:\ASPEN\SCANNER\TimeClock.scj

SCRIPT ERROR [Microsoft JScript compilation error, Line 82]
Expected ')'
          + dOutDate);
```

Sample Scripts That Work With ScanStream

Script to send all scanned data followed by an Enter key directly to application:

```
Function COMPort_OnInput(ByVal sInput)
    Scanner.SendKeys(sInput & "{Enter}")
End Function
```

Script to remove preceding 'US' on all scanned data, then send data directly to the application followed by an Enter key:

```
////////////////////////////////////
// This script checks the first 2 characters of the scanned barcode
// for a match of 'US'.  If found, US is removed from the
// input before sending it to the active application.  An enter
// key is sent to the active application following the scanned input.
////////////////////////////////////
function COMPort_OnInput(sInput)
{
    if ( sInput.substr(0,2) == 'US' )
    {
        sInput = sInput.substr(1, sInput.length);
    }

    Scanner.SendKeys(sInput + "{Enter}");
}
}
```

Script to streamline receiving and send data to application:

```
////////////////////////////////////
// This script allows users to login by scanning their badge bar
// code, keeps track of the number of packages received by this
// user during the login session, and sends the following info to
// the active application anytime a package is scanned:
// User    Date    Time    Bar Code Scanned    Total Pkgs.
////////////////////////////////////

// global variables
var nTotalPkg = 0;        // total packages for this user this shift
var sUser = '';          // user's badge number
var sDate;
var sTime;

////////////////////////////////////
// function COMPort_OnInput
```

ScanStream

```
// Parameters: sInput is the value of the bar code
// that was scanned.
// Notes:
// 1) This function gets called every time something is
// scanned on the port assigned to COMPort.
// 2) If no user has signed in, no response from
// function.
// 3) Total package counter gets reset to 0 when user
// signs out.
////////////////////////////////////
function COMPort_OnInput(sInput)
{
    // see if user is signing in
    if ( sUser == '' )
    {
        if ( sInput.length == 8 & sInput.substr(0,1) == 'X' )
        {
            sUser = sInput;
        }
    }
    // see if user is signing out
    else if ( sUser == sInput )
    {
        sUser = '';
        nTotalPkg = 0;
    }
    // must be scanning a receipt package
    else
    {
        nTotalPkg = nTotalPkg + 1;
        GetAndFormatDate();
        Scanner.SendKeys(sUser + "{Tab}");
        Scanner.SendKeys(sDate + "{Tab}");
        Scanner.SendKeys(sTime + "{Tab}");
        Scanner.SendKeys(sInput + "{Tab}");
        Scanner.SendKeys(nTotalPkg + "{Enter}");
    }
}

} // end of function COMPort_OnInput

////////////////////////////////////
// function GetAndFormatDate()
// Notes:
// 1) Formats date field to MM/DD/YYYY.
// 2) Formats time field to HH:MI.
// 3) For documentation purposes, several temp
// variables have been declared. If RAM memory is
// an issue, could use just one temp variable.
////////////////////////////////////
function GetAndFormatDate()
{
    var dDate = new Date();
    var sTempDay      = new String(dDate.getDate());
    var sTempHours    = new String(dDate.getHours());
    var sTempMinutes  = new String(dDate.getMinutes());
```

ScanStream

```
var sTempMonth      = new String(dDate.getMonth() + 1);

////////////////////////
// add preceding zeros where needed
////////////////////////

if ( sTempMonth.length == 1 )
{
    sTempMonth = '0' + sTempMonth;
}
if ( sTempDay.length == 1 )
{
    sTempDay = '0' + sTempDay;
}
if ( sTempHours.length == 1 )
{
    sTempHours = '0' + sTempHours;
}
if ( sTempMinutes.length == 1 )
{
    sTempMinutes = '0' + sTempMinutes;
}

////////////////////////
// now put it all together
////////////////////////

sDate = sTempMonth + '/' + sTempDay + '/' + dDate.getFullYear();
sTime = sTempHours + ':' + sTempMinutes;

} // end of function GetAndFormatDate
```

Script to simulate a time clock and send data to application:

```
////////////////////////
// This script works as a time clock.  Users 'punch in/out'
// by scanning their bar coded employee number.  When they punch
// in or out, the timestamp is written to the the active application.
// When an employee punches out, the hours/minutes worked is
// calculated and sent to the active application.
////////////////////////

// global variables
var aEmployees = new Array();
var sEmployeeIn;
var sHours;
var sMinutes;

////////////////////////
// function COMPort_OnInput
// Parameters:  sInput is the value of the bar code
// that was scanned.
```

ScanStream

```
// Notes:
// 1) This function gets called every time something is
//    scanned on the port assigned to COMPort.
// 2) If the employee has not punched in, they will be punched
//    in and a record/timestamp is sent to the active application.
// 3) If the employee has already punched in, they will
//    be punched out and their hours worked is calculated
//    and sent to the active application.
////////////////////////////////////
function COMPort_OnInput(sInput)
{
    var bPunchIn = true;
    var dDate = new Date();

    // first lets make sure it was a employee badge that was
    // scanned
    if ( sInput.length != 8 & sInput.substr(0,1) != 'X' )
        return;

    // see if an employee is punching in or out
    for (i = 0; i < aEmployees.length; i++)
    {
        if ( sInput == aEmployees[i].substr(0,8) )
        {
            // employee is punching out
            bPunchIn = false;
            break;
        }
    }

    if ( bPunchIn == true )
    {
        // log employee in
        aEmployees.push(sInput + ',' + dDate);
        Scanner.SendKeys(sInput + " arrived on ");
        Scanner.SendKeys(dDate + "{Enter}");
    }
    else
    {
        // log employee out & calculate hours worked
        sEmployeeIn = new String(aEmployees[i]);
        var sEmpNum = sEmployeeIn.substr(0, sEmployeeIn.indexOf(','));
        Scanner.SendKeys(sEmpNum + " departed on ");
        Scanner.SendKeys(dDate + "{Enter}");
        GetAndFormatHoursWorked();
        Scanner.SendKeys(sEmpNum + " Hours Worked = " + sHours +
            "; Minutes Worked = " + sMinutes + "{Enter}");

        // remove this employee from the array
        aEmployees.splice(i,1);
    }
} // end of function COMPort_OnInput

////////////////////////////////////
```

ScanStream

```
// function GetAndFormatHoursWorked
// Parameters: none
// Notes:
// 1) This function gets called when an employee badge is
// scanned that is already in the aEmployee array (must
// be punching out).
// 2) Global variables sHours & sMinutes get set based
// on time elapsed since the employee punched in.
////////////////////////////////////
function GetAndFormatHoursWorked()
{
    var dInDate = new Date(sEmployeeIn.substr(sEmployeeIn.indexOf(',')
        + 1, sEmployeeIn.length));
    var dOutDate = new Date();

    // calculate the # of seconds worked
    var sCount = (dOutDate - dInDate)/1000;

    // see if they forgot to signout
    var sDays = sCount / (24 * 3600);

    if ( parseInt(sDays) > 0 )
    {
        Scanner.SendKeys("Clocked in for " + sDays + " Day(s)! {Enter}");
        sCount = parseInt(sCount) - (24 * 3600 * sDays);
    }

    // calculate hours
    sHours = parseInt(sCount) / 3600;

    if ( parseInt(sHours) > 0 )
    {
        sCount = parseInt(sCount) - (3600 * parseInt(sHours));
        sHours = Math.floor(parseInt(sHours));
    }
    else
    {
        sHours = 0;
    }

    // calculate minutes
    sMinutes = Math.round(parseInt(sCount) / 60);
} // end of function GetAndFormatHoursWorked()
```

Appendix A - SendKeys Codes

To specify characters that aren't displayed when you press a key, such as ENTER or TAB, and keys that represent actions rather than characters, use the codes shown below:

KeyName	SendKeys Code	Comments
	{}	Since { is special, it must be enclosed in curly braces
}	{}}	Since } is special, it must be enclosed in curly braces
Alt	{ALT}	Modifier key.
Backspace	{BACKSPACE} or {BKSP}	
Break/Cancel	{BREAK}	
Clear	{CLEAR}	
Control	{CTRL}	Modifier key.
Delete	{DELETE} or {DEL}	
Down Arrow	{DOWN}	
End	{END}	
Escape	{ESCAPE} or {ESC}	
Execute	{EXECUTE} or {EXEC}	
F1 – F12	{F1} – {F12}	Function keys
Help	{HELP}	
Home	{HOME}	
Insert	{INSERT} or {INS}	
Left Arrow	{LEFT}	
Page Down	{PGDN}	
Page Up	{PGUP}	
Pause	{PAUSE}	
Enter/Return	{ENTER} or {RETURN}	
Right Arrow	{RIGHT}	
Select	{SELECT}	
Shift	{SHIFT}	Modifier key.
Space	{SPACE}	A space character also works except when specifying a repeat count inside curly braces.
Print Screen	{PRTSCR} or {PRTSC}	
Tab	{TAB}	
Up Arrow	{UP}	

Appendix B - Installing ScanStream

Following are instructions for installing ScanStream manually:

- 1) Create the directory C:\Program Files\ScanStream.
- 2) Copy ScanStream.exe and sample script files to the C:\Program Files\ScanStream directory.
- 3) Copy MFC42.DLL and MSVCRT.DLL to the Windows system directory. For WindowsNT, the system directory is WINNT\System32. For Windows95/98, the system directory is Windows\System.
- 4) Either create a desktop short cut to ScanStream.exe; or to set up ScanStream to start automatically when the PC turns on, create a short cut in the C:\WINDOWS\Start Menu\Programs\StartUp directory. When adding a short cut in the StartUp directory, the PC must be re-booted before the changes take affect.

Appendix C - Using ScanStream with Floormation

ScanStream is most commonly used to scan data into and print labels from Floormation. ScanStream is designed for use with Floormation, but it is a separate and independent program. It is not a database application.

There are some special ScanStream requirements when printing labels from Floormation:

- 1) ScanStream must be running on the Station PC.
- 2) The ScanStream port configuration for the Zebra printer must have the PortID set to the LabelID that corresponds with the Zebra label format file (from the LabelConfig table).

Most Floormation station PCs set up ScanStream to start up automatically whenever the PC is turned on. To set up a PC to start ScanStream automatically, create a short cut to ScanStream in the C:\WINDOWS\Start Menu\Programs\StartUp directory.