Brian Kiser
December 1999

Andon Control Software

Visual FoxPro 6.0

Trim-Masters, Inc.
Bardstown, Kentucky

Table of Contents

1.0 Work Sample Description

The giant LED display below (**Figure A**) is called an Andon. I wrote the application suite that powers these nine andon displays on the production floor, which are fed data from bar code scanning and keyboard entry. Production numbers are automatically adjusted either positively or negatively whenever a bar code is scanned at a production floor workstation. A manager can also manually adjust production numbers, if necessary. In this way, production and inventory are monitored automatically, without human intervention. The application is fully object-oriented and is composed of three separate modules:

- **Andon Communicator** - communicates with the andon display devices (**Figure B**).
- **Andon Watch -** allows office personnel to view the production totals from their workstations (**Figure C**).
- **Andon Control Center -** allows management to set user-defined values (**Figure D**).

Assembly line personnel do not always have access to a PC. Many employees are commonly "on the production floor" performing other tasks. There is a need to display production totals in a readily visible format and without the availability of a PC. These nine andons were hung above the assembly lines to display this production information. *All totals are updated in real time*. Here is what each number means:

- **Plan** is the number of doors that *should have* been manufactured from the start of the shift until now.
- **Actual** is the number of doors manufactured from the start of the shift until now.
- **Diff** displays a negative number if the line is behind schedule. Optimally, this number should be zero.
- **Scrap** is the number of doors that have been damaged beyond repair.
- **Defect** is the number of defective doors build. They will be scanned at the repair station when repaired.
- **FTC** is First Time Capability, and is basically the percentage of doors that are build correct the first time, with no defects. To meet the standards set forth by this company, FTC should remain 96% or better.

The numbers displayed are pulled from a Visual FoxPro 6.0 database. This database receives multiple updates every second from 17 assembly line computers, and is polled for reports and queries by various departments, including accounting, production control, MIS, manufacturing, and quality control. The totals generated by the andon applications are the definitive production totals for the manufacturing facility, so they must be reliable at all times. In all, a minimum of 25 users and often up to 90 users access the database at any one time. The database and application is fully relational, properly indexed, and highly optimized. This is a true multi-user application.

Figure A – The Andon

This LED display, called an Andon, is one of nine that display real-time manufacturing production numbers. These nine andons are powered by an object-oriented Visual FoxPro 6.0 application.

The Andon Communicator

The Andon Communicator runs 24 hours a day and supplies live production numbers to the andon. The manufacture of items is carefully timed and monitored, and items are typically produced according to an exact takt-time. In my manufacturing environment, a typical takt-time is between 35 and 45 seconds.

A 500 MHz PC with 512 MB of RAM powers the nine LED displays. Basically, this PC polls the data written from the assembly line workstations and sends that data to the andon that is located over that particular assembly line. When an item's bar code is scanned the display must update immediately. The andon display is updated every second in order for the display to appear "live". Within this 1 second, the data is read, all necessary computations are performed, and the results are output across the network to the andons. Each andon functions independently of each other, so as to isolate any problems that a particular andon might encounter. One hardware or software problem does not stop every andon in the manufacturing facility. Each hardware andon has a software counterpart that provides the communication layer to supply the

totals to the device. This software representation is called an Andon Communicator (**Figure B**), and provides an attractive visual representation of the andons. The green light is functional. The andon device driver is constantly polled, and if communication is lost, then the light turns red. This module is the heart of the andon application. DDE (Dynamic Data Exchange) is used to communicate with the andon device driver. (See section 4.1.1 for sample code.)

All nine andons are connected to the PC via twisted pair cable with RJ-45 connectors on the andon ends. On the PC side, the twisted pair joins with a box that connects to the serial port of the computer. This box, supplied with the andons, allows communication. All output goes through the serial port to the andons.

The Andon Communicator facilitates communication among various assembly line stations, PC workstations, and the andons themselves. Since each andon behaves similarly, each is represented by a separate form which

Figure B – An Andon Communicator

This is the display that is shown on the Andon Control Center computer. A green light indicates the andon is receiving data normally. A red light indicates a hardware problem has occurred and the andon is offline.

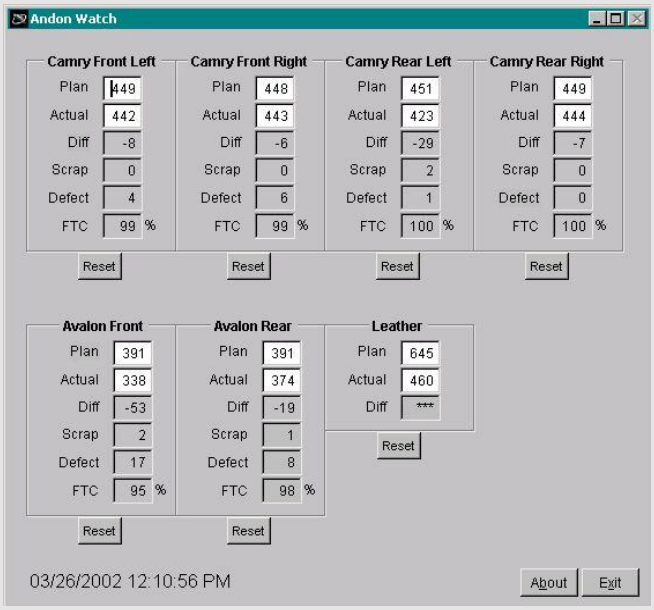

Figure C – The Andon Watch Module

The Andon Watch module was created to allow management to remotely view all assembly line production, in real time, from a PC.

instantiates a reusable class. This calling class contains very little code, and basically serves to identify the particular assembly line. Since this development language was object-oriented, I had the benefit of being able to use properties and methods, as well as inheritance. The form has a property called LineName, which contains an identifier indicating the assembly line for this andon (i.e.: “af” would indicate the Avalon Front assembly line).

The Andon Watch

I created the Andon Watch module as a visual tool for managers to use. This application resides on various personnel workstations and allows them to remotely monitor production totals (**Figure C** above).

Management also asked for the ability to change these totals manually. Since break times can change depending on various unforeseen conditions such as equipment breakdowns or parts not arriving on time, managers also needed the ability to change the break times, as well as the shift start and end times. The takt-time can change at

any moment as well, depending on various factors, like production falling behind or moving too quickly, so the takt-time must be adjustable as well. Any adjustment must take place “real time”, so any changes that are made take effect immediately (the Communications module must check for changes). Both the Andon Watch and the Andon Control Center modules allow production totals to be changed, while only the Andon Control Center allows more advanced control, such as the ability to change takt-time and shift start time. The Andon Watch module is reasonably self-explanatory. All production totals are displayed. The totals displayed in white textboxes are user-editable, while the disabled textboxes are computed fields and not editable. If the user edits a total, the application goes into edit mode and Save and Cancel buttons appear. All changes are logged to an audit trail.

The Andon Control Center

The Andon Control Center interface (**Figure D**) is a bit more complex. All entered values can be saved as defaults, or recalled if necessary. A shift can be cancelled. This is significant because all andon values are stored for reporting purposes, and since the andons are set to run automatically 5-7 days a week, there must be a mechanism to cancel a shift (which sometimes happens because of inclement weather, holidays, or other reasons). Microsoft OLE Automation is used to save production totals to an Excel spreadsheet. This spreadsheet creates a page for each work date. At the end of the month, the application creates a new spreadsheet so that there is one spreadsheet per month. No human intervention is required to perform these tasks. Various management personnel use the data stored in the Excel spreadsheet to perform analysis on production totals. Defects can be changed via the Change Defects button. Access to this screen requires a password, as modifying the number of defects is a potential security violation. User-selectable passwords are stored in an encrypted table.

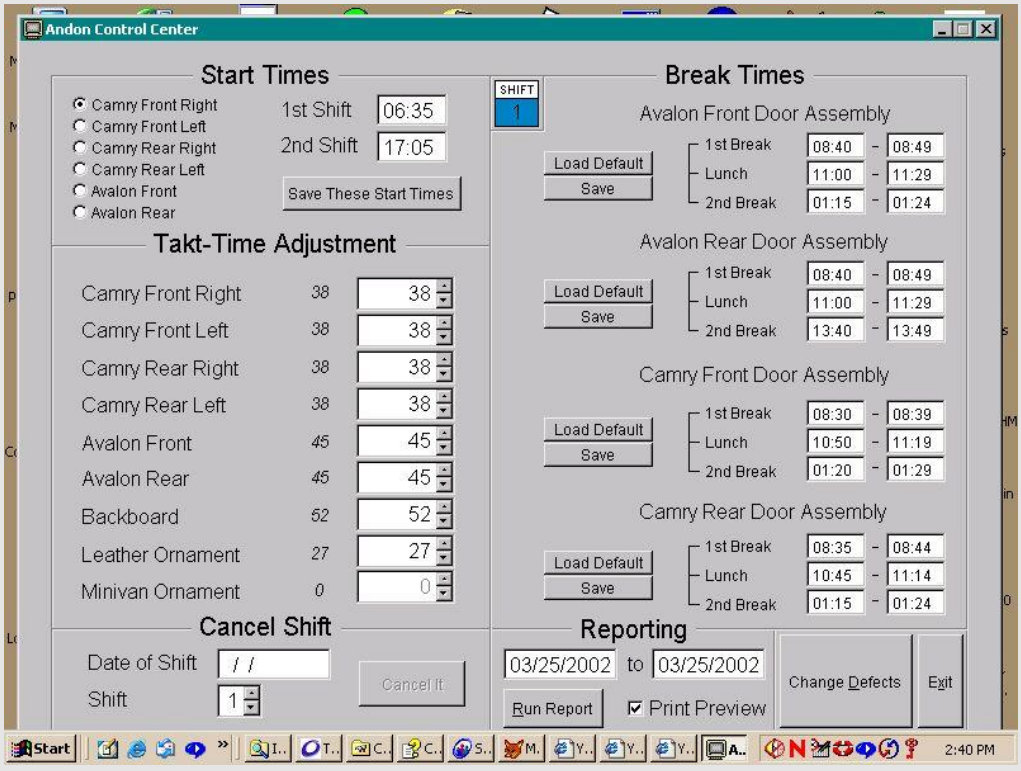

Figure D – The Andon Control Center

The Andon Control Center allows for the takt-time to be adjusted, as well as changing start of shift, end of shift, and break times. The andon does not run during breaks or between shifts. All these values are set by the user and can be saved as a default.

2.0 Skills Demonstrated

2.1 Experienced integrating hardware devices with programming

The Andon applications, although small individually, perform some powerful tasks. 3 of 9 bar code input is retrieved and stored on a high-volume, second by second basis using both corded and cordless hand scanners. That input is then manipulated, per business rules, and eventually stored to database tables and displayed on the Andon Control Center screen and giant LED displays hanging above the production floor. Establishing communication with these displays involved DDE and extensive hardware and software troubleshooting. This project involved installing specialized device drivers, running network cabling, crimping cables, PC hardware skills, and some inventive programming. Troubleshooting involved using third party hardware and software, and establishing where the problem originated in a complicated environment.

2.2 Ability to create a complicated application unassisted

This suite involved writing from scratch multiple Visual FoxPro applications that share database access. File and record locking was used appropriately to ensure correct handling of multi-user situations. Cooperation between different modules accessing the same data at the same time was essential. Performance was a factor due to the takt-time requirements and the DDE communication with the andon device drivers, and code improvements were made to achieve positive results.

The application is fully object-oriented. The nine Andon Communicators were objects that inherited from a base communicator class. Because their functions were so similar, only a small amount of method overriding was required after the initial base class was written.

3.0 Difficulties Encountered

In testing, I discovered that the application was not correctly keeping track of the 40-second takt-time. The Plan figure was incrementing at an incorrect rate. Even though the takt-time was set at 40 seconds, the Plan appeared to be incremented every 35 seconds, or 37 seconds, or 42 seconds, etc. The machine that originally hosted this software was a 300 MHz machine with 128MB of RAM. Because of these sluggish results, I changed the machine to a 500 MHz machine with 512MB of RAM. Although takt-time accuracy improved, it still was not 100% accurate—it tended to lag up to 3 seconds each cycle, which is a significant error range. Double-checking the application revealed no problems, but checking the error log of the andon device driver revealed many transmission errors. The network wiring for the displays was twisted pair, which was apparently having a high rate of errors due to interference from the many mechanical devices. There were so many errors detected that network transmission speed was affected, thus causing lag in the real-time takt-time calculations. To correct this problem required first checking the connections, recrimping when necessary, then shielding the twisted pair from interference.

This was a high profile, very visual project, and one of the most enjoyable that I have ever worked on.

4.0 Code Listing

Per company standards, variables and screen objects were named using Hungarian Notation. The standards used in this application are:

Screen objects:

- lbl = label
- tmr = timer
- img = image
- txt = textbox

iables:

- lc = local character
- ln = local numeric
- ll = local logical
- o = object

4.1 Andon Communicator

The Andon Communicator is the heart of the application, as it powers the LED displays. I have included selected methods from that form class. All forms, classes, and tables are available upon request.

4.1.1 InitBoard

```
* InitBoard

lcTakt  = this.Alias + '.Takt'
lnStart = this.Start

lci = alltrim (str (lnStart))

* The alias for the table that we are using is always preceded by "and_" (to
* distinguish it from other dbf filenames) and then is the line name (i.e.
* "and_cfr" for Camry Front Right).

* Initialize the 6 topics on the board.  A topic is one “field” or piece of data,
* like “Actual” or “Plan.”

for i = lnStart to lnStart + 5
   lci = alltrim (str (i))
   public Chan&lci
   Addr = '"disp7seg' + lci + '"'
   Chan&lci = ddeinitiate ('onesix', &Addr)
   =ddesetoption ('safety', .F.)
endfor

* Check the last topic to see if it's online.  If so, assume the board is online
* (which is a reasonable assumption because we have communication at this point).

lcOnline = dderequest (Chan&lci, 'online')

this.Active = (lcOnline <> '0')

if not this.Active
   * If communication could not be established, disable the timers and make the
   * red warning light visible.

   thisform.imgWarning.visible      = .T.
   thisform.tmrQuickUpdate.interval = 0
   thisform.tmrCheckTime.interval   = 0
   thisform.tmrTakt.interval        = 0
else
   thisform.imgWarning.visible      = .F.
   thisform.tmrQuickUpdate.interval = 1000
   thisform.tmrCheckTime.interval   = 10300
   thisform.tmrTakt.interval        = &lcTakt * 1000
endif
```

4.1.2 CheckForBreaks

```
* CheckForBreaks

go top in Shutdown

if not Shutdown.Shutdown
    lcTmr = 'thisform.tmrTakt.enabled'

    lcStart1 = this.Alias + '.Shift1Start'
    lcStart2 = this.Alias + '.Shift2Start'
```

```
    lcTimeFile = left (this.LineName,2) + iif (between (left (time(), 5), &lcStart1,
         &lcStart2), '_time1', '_time2')

    select (lcTimeFile)

    if between (left (time(),5), Break1Start, Break1End) or between (left
         (time(),5), LunchStart, LunchEnd) or between (left (time(),5), Break2Start,
         Break2End)

         if &lcTmr  && We're on break, and the takt-timer is enabled, so disable it.
              &lcTmr = .F.
              this.OnBreak = .T.
              thisform.lblStatus.caption = ' On Break '
              thisform.lblStatus.visible = .T.
         endif
    else             && We're not on break and the timer is disabled, so enable it.
         if not &lcTmr
              &lcTmr = .T.
              this.OnBreak = .F.
              thisform.lblStatus.visible = .F.
         endif
    endif
else
    thisform.oDisable.ForceShutDown()
endif
```

4.1.3 CheckForShiftChange

```
* CheckForShiftChange

lcStartShift1 = this.Alias + '.Shift1Start'
lcStartShift2 = this.Alias + '.Shift2Start'

lcToggle = 'And_' + this.LineName + '.Toggle'

do case
    case between (left (time(),5), &lcStartShift1, &lcStartShift2) and &lcToggle
         * This.Shift is not the current shift, but rather the shift that just ended
         * and is about to be saved to the history table.

         this.Shift = 2
         this.SaveAndReset()

    case not between (left (time(),5), &lcStartShift1, &lcStartShift2) and not
         &lcToggle

         this.Shift = 1
         this.SaveAndReset()

    case (between (left (time(),5), &lcStartShift1, &lcStartShift2) and not
         &lcToggle) or (not between (left (time(),5), &lcStartShift1,
         &lcStartShift2) and &lcToggle)

         * No reset is required, so check for breaks.
         this.CheckForBreaks()
endcase
```

4.1.4 SaveAndon

```
* SaveAndon

* These fields are updated here:
*    LASTPLAN         what we should have produced at the current takt-time
*    LASTDIFF         Actual - Plan
*    FTC              100 - ((Defects / Actual) * 100)
*                     We multiply (Defects / Actual) by 100 to make it a
*                     percentage.  This result is then rounded.  Then we subtract
*                     this value from 100 to get the FTC percentage.

* These fields are not updated here.  They are updated in systems on the line:
*    LASTACTUAL       the actual number produced (end-of-line PC)
*    LASTSCRAP        the actual number of doors that are unusable (scrap PC)
*    LASTDEFECT       the actual number of defects found (end of line PC)

select (this.Alias)

* IIF prevents numeric overflow.
replace LastPlan with iif (LastPlan = 999, 1, LastPlan + 1) replace LastDiff with
    Lastactual - LastPlan

* FTC is First Time Capability.  It's (Defects / Actual) * 100, but only values
* that are inserted from the end-of-line PC are used to compute FTC.

if this.LineName <> 'lo'
    replace LastFTC with iif (LastDefect = 0, 100, 100 - (round ((eolDefect /
        eolActual) * 100, 0)))
endif

* The QuickUpdate timer will fire and update the screen and andon with these new
* values.

flush
go top
unlock
```

4.1.5 SaveAndReset

```
* SaveAndReset

lcActual = this.Alias + '.LASTACTUAL'

if &lcActual > 0
   * If there was production, save the current andon values to the history table.

   lcStart1 = this.Alias + '.Shift1Start'
   lcStart2 = this.Alias + '.Shift2Start'
   lcScrap  = this.Alias + '.LastScrap'
   lcDefect = this.Alias + '.LastDefect'
   lcFTC    = this.Alias + '.LastFTC'

   ldProdDate = iif (between (left (time(), 5), '00:00', lcStart1), date() - 1,
        date())

   insert into AndHist (Date, Line, Shift, Actual, Scrap, Defect, FTC) ;
      values (ldProdDate, this.LineName, this.Shift, &lcActual, &lcScrap,
```

```
        &lcDefect, &lcFTC)
endif

* Reset the values on the andon regardless of whether there was production.

select (this.Alias)
replace LastPlan with 0, LastActual with 0, eolActual with 0, LastDiff with 0, ;
   LastScrap with 0, LastDefect with 0, eolDefect with 0, LastFTC with 100, Toggle
    with not Toggle
```

4.1.6 StopComm

```
* StopComm

lnResult = messagebox ('This will stop communication to the ' + thisform.caption ;
     + ' andon board.  Are you sure?', 292, 'Verify')

if lnResult = 6  && Blank out all the lights to show that communication is over.

     * To terminate communication to an andon, I terminate all 6 channels for
     * the specified board.  To indicate communication to the board has been
     * terminated, I blank all topics.

   for i = this.Start to this.Start + 5
      lci = alltrim (str (i))
      =ddepoke (Chan&lci, 'displaytext', '    '))
      =ddeterminate (Chan&lci)
   endfor

   thisform.release()
   close all
   cancel
endif
```

4.1.7 UpdateAndon

```
* UpdateAndon

if this.Active
     * If we get inside this IF, then this andon is up and running.  I update
     * the andon whether we're on break or not.  This.Active just indicates if
     * the andon is getting communication.

     lni = this.Start

     lcPlan   = this.Alias + '.LastPlan'
     lcActual = this.Alias + '.Lastactual'
     lcDiff   = this.Alias + '.LastDiff'
     lcScrap  = this.Alias + '.LastScrap'
     lcDefect = this.Alias + '.LastDefect'
     lcFTC    = this.Alias + '.LastFTC'

     * Update the andon board only if the value has changed.  The pokes don't
     * seem to take up much time when testing with a single andon, but over a
     * decent-sized network with lots of topics, they can make a big difference.

     if thisform.lblPlan.caption <> alltrim (str (&lcPlan))
          lcj = alltrim (str (lni))
```

```
            =ddepoke (Chan&lcj, 'displaytext', padl (alltrim (str (&lcPlan)), 4, '
            '))
            thisform.lblPlan.caption = alltrim (str (&lcPlan))

            * Update the Diff topic.
            lcl = alltrim (str (lni + 2))
            =ddepoke (Chan&lcl, 'displaytext', padl (alltrim (str (&lcDiff)), 4, '
            '))
            thisform.lblDiff.caption = alltrim (str (&lcDiff))
        endif

        if thisform.lblActual.caption <> alltrim (str (&lcActual))
            lck = alltrim (str (lni + 1))
            =ddepoke (Chan&lck, 'displaytext', padl (alltrim (str (&lcActual)), 4,
                 ' '))
            thisform.lblActual.caption = alltrim (str (&lcActual))

            * Update the Diff topic.
            lcl = alltrim (str (lni + 2))
            =ddepoke (Chan&lcl, 'displaytext', padl (alltrim (str (&lcDiff)), 4, '
            '))
            thisform.lblDiff.caption = alltrim (str (&lcDiff))
        endif

        * When the form is minimized, display the Actual number.
        thisform.caption = iif (thisform.windowstate = 1, this.Title + ' ' +
            thisform.lblPlan.caption + '/' + thisform.lblActual.caption,
            this.Title)

        if thisform.lblScrap.caption <> alltrim (str (&lcScrap)) and this.LineName
            <> 'lo'
            lcm = alltrim (str (lni + 3))
            =ddepoke (Chan&lcm, 'displaytext', padl (alltrim (str (&lcScrap)), 4,
            ' '))
            thisform.lblScrap.caption = alltrim (str (&lcScrap))
        endif

        if thisform.lblDefect.caption <> alltrim (str (&lcDefect)) and
            this.LineName <> 'lo'
            lcn = alltrim (str (lni + 4))
            =ddepoke (Chan&lcn, 'displaytext', padl (alltrim (str (&lcDefect)), 4,
            ' '))
            thisform.lblDefect.caption = alltrim (str (&lcDefect))
        endif

        if thisform.lblFTC.caption <> alltrim (str (&lcFTC)) and this.LineName <>
            'lo'
            lco = alltrim (str (lni + 5))
            =ddepoke (Chan&lco, 'displaytext', padl (alltrim (str (&lcFTC)), 4, '
            '))
            thisform.lblFTC.caption = alltrim (str (&lcFTC))
        endif

        * This GO TOP is necessary to release the lock on the record, and thus
        * allow the andon to be updated every second.  Without the GO TOP, the
        * andon will not update in real-time.

        select (this.alias)
        go top
    endif
```