Object-Oriented Test Automation

George Cerny (QA Manager)
SmartSignal Corporation
SmartSignal at a Glance

- The leader in predictive analytics for equipment performance worldwide.
- We provide early detection of equipment and process problems—prevent them from growing into large or catastrophic failures.
- Software and full-service software/services packages.
- Across industries: power, oil & gas, aviation, pulp & paper
- Around the world: N America, Europe, Asia, Africa
- Clients include: Ameren, BC Hydro, Caterpillar, Constellation, Delta Airlines, Dynegy, DTE, Entergy, Mitsubishi, Raytheon, Reliant, Southwest Airlines……
QA at SmartSignal

- Quality Assurance (QA) is a critical component of the software delivery lifecycle.
- The QA team has been a catalyst for driving efficiencies and improved customer satisfaction across the organization.
- Test Plans are complex and extensive. One Test Plan has over 8,000 test cases with over 750,000 lines of test code constituting more than 100 man hours of testing.
- QA needs to validate the products for multiple regions including North America, Europe, Asia, Africa.
- Virtual Environments and Automated testing have enabled testing across all supported environments.
The QA Project Tracking Dashboard is a repository for all the materials that the QA Team develops and exposes the data to the entire organization.

- Project status snapshot pie charts represent testing status.
- Test Requirements are developed, estimated, assigned, and targeted for weekly milestones and project tracking.
- Test Executions for all test plans are correlated and summarized.
- Performance Testing of server processing and user interfaces are executed under various system loads to measure response times.
Project Status Snapshot

EPICenter 2.4.0 Test Requirement Status

- Status of new feature testing

EPICenter 2.4.0 Automation Results

- Status of regression testing
# Test Requirement Milestones

## Test Requirements are high level

Status summarized in snapshot charts

Break down by week and track to estimates
Automation Results

Version: 2.3.2

- Plan smartsignal.pln - 663 errors
- Machine: (All Machines)
- Elapsed: 287:25:11
- Passed: 8191 tests (98.877354)
- Failed: 93 tests (1.122646)
- Totals: 8284 tests, 653 errors, 11924 warnings

Version: 2.4.0

- Plan smartsignal.pln - 1113 errors
- Machine: (All Machines)
- Elapsed: 258:17:1
- Passed: 8196 tests (97.944551)
- Failed: 172 tests (2.055449)
- Totals: 8368 tests, 1113 errors, 10856 warnings

- Plan smartsignal.pln - 308 errors
- Locale: US
- Elapsed: 125:3:5
- Passed: 3769 tests (98.949856)
- Failed: 40 tests (1.050144)
- Totals: 3809 tests, 308 errors, 5988 warnings

- Plan smartsignal.pln - 345 errors
- Locale: GR
- Passed: 4422 tests (98.815642)
- Failed: 53 tests (1.184358)
- Totals: 4475 tests, 345 errors, 5936 warnings

- Plan smartsignal.pln - 808 errors
- Locale: GR
- Elapsed: 143:53:21
- Passed: 4362 tests (97.019573)
- Failed: 134 tests (2.980427)
- Totals: 4496 tests, 808 errors, 5367 warnings

- Locale: US
- Plan smartsignal.pln - 4 errors
- Machine: (GCERNY2003)
- Elapsed: 16:36:05
- Passed: 239 tests (99.3%)
- Failed: 2 tests (0.7%)
- Totals: 301 tests, 4 errors, 126 warnings
- Developer: George Cerny

- Plan smartsignal.pln - 2 errors
- Machine: (GCERNY2003)
- Elapsed: 15:31:43
- Passed: 305 tests (99.7%)
- Failed: 1 test (0.3%)
- Totals: 306 tests, 2 errors, 29 warnings
- Developer: George Cerny

© 2009 SmartSignal Corp. Confidential

- Compare results to previous release
- Summarize results at different levels
- Compare Test Plans by locale
- Track bugs that are causing failures
Performance Benchmark Results

<table>
<thead>
<tr>
<th>Machine</th>
<th>Login</th>
<th>2.3.2</th>
<th>2.4.0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>3.906</td>
<td>3.907</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>4.828</td>
<td>5.875</td>
<td>-3.56%</td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>4.225</td>
<td>4.375</td>
<td></td>
</tr>
<tr>
<td>Machine 2</td>
<td>Login</td>
<td>4.016</td>
<td>3.937</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>4.937</td>
<td>4.469</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>4.340</td>
<td>4.208</td>
<td>3.04%</td>
</tr>
<tr>
<td>Machine 3</td>
<td>Login</td>
<td>3.906</td>
<td>4.015</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>6.187</td>
<td>5.250</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>4.439</td>
<td>4.423</td>
<td>0.37%</td>
</tr>
<tr>
<td>Machine 4</td>
<td>Login</td>
<td>4.000</td>
<td>3.969</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>4.782</td>
<td>4.859</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>4.276</td>
<td>4.333</td>
<td>-1.35%</td>
</tr>
<tr>
<td>Machine 1</td>
<td>Graph</td>
<td>8.093</td>
<td>7.516</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>11.234</td>
<td>11.282</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>9.842</td>
<td>9.292</td>
<td>5.58%</td>
</tr>
<tr>
<td>Machine 2</td>
<td>Graph</td>
<td>8.500</td>
<td>7.641</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>11.250</td>
<td>12.156</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>10.223</td>
<td>9.669</td>
<td>5.42%</td>
</tr>
<tr>
<td>Machine 3</td>
<td>Graph</td>
<td>8.437</td>
<td>8.234</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>11.610</td>
<td>14.829</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>10.415</td>
<td>10.745</td>
<td>-3.17%</td>
</tr>
<tr>
<td>Machine 4</td>
<td>Graph</td>
<td>8.579</td>
<td>7.953</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>11.578</td>
<td>11.313</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avg</td>
<td>10.275</td>
<td>9.976</td>
<td>2.91%</td>
</tr>
</tbody>
</table>

Summarize results and compare to a benchmark

Highlight positive and negative results
Object-Oriented Automated Testing

- Focus your efforts on creating one script, which utilizes global variables enabling execution in any environment.
- Remove object recognition syntax from your scripts by referencing global window variables for each unique object in the application.
- Create application states and functions which can be referenced in test scripts.
- Develop robust error checking, logging, and recovery procedures for bug diagnosis and reproducibility.
- Utilize data structures in functions for data entry and validation.
- Develop coding standards and provide good comments.
Automated Test Plans

- Use Automated Test Plans that call Test Cases with local variables.
  - Eliminates duplication of code allowing the same test case to be executed with multiple data sets.
  - Allows the tester to organize and rearrange the flow of the test cases.
  - Exposes high level test objectives to people outside the QA group.
Automated Test Plan Example

- Sections allows plan to be broken up across machines
- Plan hierarchy gives visibility into test sequences
- Give high level test objectives
- Reuse test cases with different data

Section 3 (GCERNYXP, QA2-GR-XP, QA20-GR-VISTA; QA28-GR-28)
Section 4 (MDWA2003, QA6-GR-XP, QA22-GR-VISTA; QA30-GR-30)

Workbench and WatchList Setup
1. Setup Runtime/ASCII Error Notifications Assets
2. Sensor Editing, Sensor Tag Mapping and Change Data
3. Create Complex Calculated Formula Test
4. If_Exists Validation Tests
5. Import Test Data where Tag doesn’t exist in the Model
6. Add Asset from MatLab Validation Tests
7. 11.1 Validation Testing and Asset Setup
   a. 11.1.1: Create the Asset Instance: LABADIE U1 AH B
      • script: addassetfrommatlab.t
      • testcase: T11_1_1 ("LABADIE U1 AH B")
   b. 11.1.2: Verify LABADIE U1 AH Ba Test Model and outfile
   c. 11.1.3: Create the Asset Instance: LABADIE U1 AH C
      • script: addassetfrommatlab.t
      • testcase: T11_1_1 ("LABADIE U1 AH C")
Test Cases

- Test Cases should consist of function calls and error handling.
- Naming standards provide traceability back to the Test Plan.
- Comment on the input parameters and local variables.
- Verify each step in the test case procedure and only continue if a success condition has been met.
- Write success and error conditions to the results log and highlight the errors for analysis.
Test Case Example

Document the objective and trace back to Plan

Comment on input parameters, local variables, and test steps

Test function calls for errors and log error/success conditions

Use naming standards

Trap unknown error conditions and recover the system
Global Window Declarations

- Removing object recognition syntax from test cases and functions and placing them into one globally referenced file is key to maintainability.
- When the application changes, controls are moved, or even running on a different locale; the object recognition logic is controlled in one spot, not the thousands of references within the scripts.
- Organize the declaration files by major component, then each file alphabetically by dialog.
- Develop standards for naming conventions.
Global Window Declaration Example

```plaintext
// ADD ASSET FROM MATLAB
//-------------------------------

WINDOW AddAssetFromMatlabToolOutputFileDir = AddAssetSFromMatlabToolTextField("Matlab tool output file directory:");
WINDOW AddAssetFromMatlabToolBrowse = AddAssetSFromMatlabToolPushButton("Browse");
WINDOW AddAssetFromMatlabCreateNewAssetsOnly = AddAssetSFromMatlabToolPushButton("Create new assets only");

//-------------------------------
// INTERNATIONAL TAG HANDLING
//-------------------------------

STRING eChartColorWinTag = ("Color", "Farbe", "Color", "Couleur")[LANGUAGE];
WINDOW ChartColorWin = eCMSsetup.DialogBox(eChartColorWinTag);

//-------------------------------
// MODELED TAG GRID FIELD
//-------------------------------

WINDOW MainWin_ModelTag_Grid = eCMSsetup.HistoricalDataSet.Frame1.OLEGrid1.Box;

//-------------------------------
// WINDOW DECLARATIONS
//-------------------------------

window DialogBox AddAssetSFromMatlabTool
{
    tag "Add asset(s) from Matlab tool files";
    parent ModelView_InternalTags;
}

window MainWin eCMSsetup
{
    tag "*Workbench*"
    WindowsForms10Window9App6 HistoricalDataSet
    {
        tag "*Data*"
        WindowsForms10Window9App6 Frame1
            {
                tag ";1"
                OLEGrid OLEGrid1
                    {
                        tag ";1"
                        StaticText Box
                            {
                                tag "/{data[@Column + 2]:[10], [iRow + 1]:[43]}"
                            };
            
```
Global Functions

- Comment on the input parameters and local variables.
- Verify each step in the test case procedure and only continue if a success condition has been met.
- Write success and error conditions to the results log and highlight the errors for analysis.
- Comment on complex coding logic and branching points.
- Beware of infinite loops, always put a time limit on synchronization points.
- Use meaningful variable names and create coding and naming standards.
Global Functions Example

```c
// THIS FUNCTION IS USED TO IMPORT A ASSET FROM THE MATLAB TOOL

BOOLEAN bAAFN_ImportAsset (STRING sControllerFileDirectory_Loc, STRING sInstanceName_Loc)
{
    // Parameter 1: The directory Path to the controller file
    // Parameter 2: The Asset to be imported

    INTEGER iWaitForImportToComplete;  // Looping Variable used to control
    BOOLEAN bSuccessfulImport = FALSE; // Boolean to determine

    do
    {
        // Open the Controller file in the Add Asset From Matlab Tool

        if (bAAFN_ImportControllerFile [sControllerFileDirectory_Loc])
        {
            // Select the Asset Instance to be imported and start the process

            AddAstMatlabAssetGrid.Select (sInstanceName_Loc + "*");  
            AddAstMatlabCheckInAsset.Click ();  
            AddAstMatlabOK.Click () ;

            // Wait for the Import Process to complete. 1 hour is max time

            for (iWaitForImportToComplete = 1; iWaitForImportToComplete <= 10; iWaitForImportToComplete++)
            {
                do
                {
                    if (AddAstMatlabControllerErrorText.Exists () )
                    {
                        sGetText = AddAstMatlabControllerErrorText.GetText () ;
                        bError (sGetText ) ;
                        bSuccessfulImport = FALSE;
                        AddAstMatlabControllerErrorOK.Click () ;
                        break;
                    }
                    if (!CreateAssetsFromMatlabFile.Exists () )
                    {
                        bSuccessfulImport = TRUE;
                        break;
                    }
                }
                while (bSuccessfulImport == FALSE);
            }
        }
    }
    while (bSuccessfulImport == FALSE);
}
```
Global Variables

Create a Global Variable file with initialization data for each Test Plan/Machine.

- Enables the scripts to be executed in any environment.
- Allows the scripts to be quickly setup and personalized to the environment and build to be tested.
- Removes redundant data from the scripts.
- By maintaining a separate file variable initialization file, automation files can be quickly refreshed from source control.
Global Variables Example

//--------------
// MACHINE/DATA BASE CONNECTION DATA
//--------------

STRING sServerName = "QA27-GR-2003";
STRING sRemoteClient = "QA-2k3-EN-03";
STRING sUserNameSet = "admin";
STRING sSystemUser = "gcerny";

const LANGUAGETYPE LANGUAGE = LT_GR;
type LANGUAGETYPE is enum
{
    LT_US, // English
    LT_GR, // German
    LT_ES, // Spanish
    LT_FR // French
}

// 1280 & 1024 resolution
//--------------

INTEGER iYResolutionFactor = 259;
INTEGER iXResolutionFactor = 256;

//------------------
// LOCALIZATION
//------------------

STRING sLocale = {"US", "GR", "ES", "FR"}[LANGUAGE];
STRING sListSep = {"", ",", ",", ","}[LANGUAGE];
STRING sRSNumbers = {"", ",", ",", ","}[LANGUAGE];
STRING sRSDates = {"mm/dd/yyyy", "dd.mm.yyyy", "dd/mm/yyyy", "dd/mm/yyyy"}[LANGUAGE];
STRING sProgramFiles = {"Program Files", "Programme", "Archivos de programa", "Program Files"}[LANGUAGE];
Data Structures are an excellent way to pass data from data files through scripts and into the application.

Data-Driven testing allows the tester to test an infinite number of combinations using the same script.

Using localization logic allows the tester to maintain one set of US data files and then transform them into any locale.
### Data Structures – Data File Example

<table>
<thead>
<tr>
<th>MODEL GENERAL INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
</tr>
<tr>
<td>10 Minutes</td>
</tr>
<tr>
<td>GCTEST2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MODELED TAG INFORMATION</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MODEL CLASS TAG INFORMATION</th>
</tr>
</thead>
</table>

### Separate Data file into logical sections

- Use unique characters for easy parsing

### Store data by variable type
Data Structures – Variable Declaration Example

type ClassModelTagInfo is record
{
  STRING sPathToNode;
  INTEGER iTagID;
  STRING sTagName;
  STRING sUsedInModel;
  STRING sClassType;
  STRING sAlarmType;
  STRING sResPlusThres;
  STRING sResNegThres;
  STRING sSPRTPlusSens;
  STRING sSPRTNegSens;
  STRING sInferred;
  STRING sMean;
  STRING sStdDev;
  STRING sResVar;
  STRING sOutlierPlusThres;
  STRING sOutlierNegThres;
}

type ClassModelAlgorithms is record
{
  STRING sPathToNode;
  STRING sSSEstGen;
  STRING sSHType;
  STRING sSMSplSmoFactor;
  STRING sSMNumObsSmoRes;
  STRING sVarScaleFactor;
  STRING sMaxHisQueSize;
  LIST OF STRING lsVirtSig;
  LIST OF STRING lsCustEstFF;
}
Data Structures – Read Function Example

```c
// THIS FUNCTION IS USED TO EXTRACT PARMs FROM FILE AND ENTER INTO THE NODE CLASS - CLASS INFO TAB

BOOLEAN bGetNode1ClassModeledTagParms (STRING sAssetNameLoc, INTEGER iWhatMode, BOOLEAN bAddTagsLoc optional)
{
    // Parameter 1: The Test Container Name
    // Parameter 2: What Mode to get data from
    // Parameter 3: Adding Tags to existing classes (optional)
    
    STRING sBaseFolder = sAutoFolder; /* Store Support \Plant\EC\Classes\Model\*/
    FILE hReadParms; /* Read the parameter file */
    ClassModelTagInfo dParmModeledTagInfo;
    do
    {
        hReadParms = FileOpen (sBaseFolder + sAssetNameLoc + ".txt", FM_READ);
        FileClose (hReadParms);
        dParmModeledTagInfo.iTagID = iHoldRow;
        dParmModeledTagInfo.sTagName = lsGetText[1];
        dParmModeledTagInfo.sTagInModel = lsGetText[2];
        dParmModeledTagInfo.sClassType = lsGetText[3];
        dParmModeledTagInfo.sAlertType = lsGetText[4];
        dParmModeledTagInfo.sResPlusThree = lsGetText[5];
        dParmModeledTagInfo.sResNegThree = lsGetText[6];
        dParmModeledTagInfo.sSPTPlusSens = lsGetText[7];
        dParmModeledTagInfo.sSPTNegSens = lsGetText[8];
        dParmModeledTagInfo.iInterred = lsGetText[9];
        dParmModeledTagInfo.iMean = lsGetText[10];
        dParmModeledTagInfo.iStdDev = lsGetText[11];
        dParmModeledTagInfo.iResVar = lsGetText[12];
        dParmModeledTagInfo.iOutlierPlusThree = lsGetText[13];
        dParmModeledTagInfo.iOutlierNegThree = lsGetText[14];
        if (bEnterSubClassModeledTagInfo (dParmModeledTagInfo, bAddTagsLoc) == FALSE)
        {
            logerror ("Failed to create Tags.");
            return FALSE;
        }
    } while (FileRead (hReadParms, sBaseFolder + sAssetNameLoc + ".txt"));
    return TRUE;
}
```
Data Structures – Write Function Example

```csharp
// THIS FUNCTION IS USED TO ENTER DATA IN THE CLASS TAG INFORMATION GRID ROW

BOOLEAN bEnterSubClassModelledTagInfo (ClassModelTagInfo dParmsLoc, BOOLEAN bAddTagsModel)
{
    // Parm 1: Data Structure for Modeled Tag Info Grid
    // Parm 2: Either create new Tags and add to existing class

    do
    {
        // Enter the Tag Name
        SubClassModelTagGridField.Click ()
        SubClassModelTagGridField.SetText (dParmsLoc.sTagName);

        // Select the Used In Model

        SubClassModelTagGrid.Field.Key (dParmsLoc.sUsedInModel[1]);
        SubClassModelTagGrid.Field.Key ("<Tab>");

        // Select the Residual + Threshold

        sGetText = dParmsLoc.sResPlusThres;
        sGetText = StrTran (sGetText, ",", sRSNumbers);
        dParmsLoc.sResPlusThres = sGetText;
        SubClassModelTagGridField.Field.Key (dParmsLoc.sResPlusThres);

        return TRUE;
    } except
    {
        logerror ("Enter Subclass Class Tag Information Function Failed.");
        ExceptionLog ();
        CloseAll ();
        return FALSE;
    }
}
Data Structures - Localization Example

// THIS TESTCASE IS USED TO CONVERT A ASCII DATA FILE TO A LOCALIZED FORMAT

BOOLEAN ConvertData (STRING sInputFile, STRING sOutputFile) {
    // Param1: Input File Path
    // Param2: Output File Path
    
    hRead = FileOpen (sInputFile, FM_READ);
    hWrite = FileOpen (sOutputFile, FM_WRITE);
    while (FileReadLine (hRead, sGetText))
    {
        // Transform the delimiter and decimal characters to localized format
        sGetText = StrTran (sGetText, sInputDel, sListSep);
        sGetText = StrTran (sGetText, ",", sRNumbers);

        // Transform the Dates to localized format
        if (MatchStr ("/\/*", sGetText))
        {
            switch (sRSDates)
            {
            case "dd/mm/yyyy":
                { sMonth = GetField (sGetText, "/", 1);
                sDay = GetField (sGetText, "/", 2);
                iRow = StrPos ("/", sGetText);
                iRow = StrPos ("/", sGetText);
                iColumn = StrPos ("/", sGetText);
                sGetText = Stuff (sGetText, iRow + 1, iColumn - iRow - 1, sMonth);
                }
            case "dd mm yyyy":
                { sMonth = GetField (sGetText, "/", 1);
                sDay = GetField (sGetText, "/", 2);
                iRow = StrPos ("/", sGetText);
                iRow = StrPos ("/", sGetText);
                iColumn = StrPos ("/", sGetText, TRUE);
                sGetText = Stuff (sGetText, iRow + 1, iColumn - iRow - 1, sMonth);
                }
            }
        }

        // Finally write the converted line to the new file
        WriteLine (hWrite, sGetText);
    }
}

Read data from file and create a new file

Convert delimiters and decimals to localized format

Convert dates

Finally write the converted line to the new file
Automation Test File Architecture

- Create a Hierarchy with a logical breakdown.
- Separate the files according to function.
- Utilize a Source Control system.
- Static and dynamic directories should be separated for quick synchronization with source control.
Automation Test File Architecture Example

- Directory for global files
- Directory for test plans and test cases
- Directory for static data files
- Organize the data files by component
Key-Word Driven Automation

- Key-Word Driven automation utilizing data-driven scripts allows other non-QA team members to leverage automation.
- Allows test cases to dynamically reference functions by name.
- Design scripts to handle unknown keywords and error conditions gracefully.
- Report meaningful information back to the user.
- Utilize office tools like Excel with macros as a user Interface to configure the data.
Key-Word Driven Automation Example

```csharp
#include <iostream>

int main() {
    // Pass in keyword from plan or data file
    String sFunctName, STRING sWhatToExecute;
    // Call functions by keyword name
    switch (sFunctName) {
        // Apply additional logic to script
        case "Open Home Page":
            switch (sWhatToExecute) {
                // Handle unsupported keywords
                default:
                    logerror ("Invalid test key word: \"sWhatToExecute\", test aborted.");
                    break;
            }
    }
    else
    {
        logerror ("The page: \"sFunctName\" was not loaded successfully, test \"sWhatToExecute\" failed.");
    }
    except
    {
        logerror ("Exception occurred running page: \"sFunctName\" and testcase: \"sWhatToExecute\"."");
    }
}
```
**Key-Word Driven Interface Example**

Users utilize Excel to organize their data.

Data structure can be in any format.
Key-Word Driven Macro Example

Users run a macro file to start the process.

Dialogs guide the user through the process and prompt for keywords.
Key-Word Driven Data Files Example

- Controller file created with keywords
- Supporting data files are created from spreadsheet data
- Data structure functions are used to read and write data into application
Key-Word Driven Data Files Example

Users open a automation plan and start execution

Keywords are used to drive the data entry functions and logic

```plaintext
// Maintenance Procedures
// Power/Airline Scripts
// script: maintenancescripts.t
// $StopTime = "" // The time to stop running

// Execute Main Script
// testcase: MainScript {$StopTime}

/*
 j = Create category and asset, open asset edit and import historical data file. import historical data,
 set up mode definition from parameters (only 1 mode), begin model loop: add model 1 and open sensor info,
 begin Sensor Loop: set used in model, incident filter type, episodic decision, window size, alarm type,
 +/- residual threshold, +/- sprt sensitivity, inferred, mean, std dev, res var. repeat Sensor Loop:
 (if filtering, reference, state matrix datafiles exist then start StateMatrix loop: filter the model's
 historical data node, select reference data on the model's filtered data node, create state matrix. )
 repeat model loop: check the asset in

 Controller DataFile: (any name)
 × line2: Mode1 Pachs (ex: h."C:\Public\HistoricalDataFile.txt")
 × perm1: tells the script to process the Asset/Model Creation Script (ex: h)
 × line3 - n: category asset (ex: 7001-1 051194)
 Mode DataFile: (named "{Asset Name} Mode.txt")
 × line1: Name of Mode (ex: Running State)
 × line2: Mode Definition (ex: {"ALT","Greater Than","1555"}. if {{}} then no modes, multiple nodes sep
 perm1: Sensor Name
 perm2: Comparison Operator
 perm3 - n-1: Comparison value
 permn: e0f

 case #T:
 {

 // Set the Asset Class Name
 //---------------------------

 sNewAsset = GetField (lsCategories[iLoop], " ", 1);
 if (MatchStr ("* *", lsCategories[iLoop]))
```
Summary and Q & A

- 100 percent reduction in rework through automation with one set of test cases.
- 15 times faster testing with improved overall efficiency and application quality.
- 10 times faster structure creation with improved consistency and customer care.

SmartSignal Automation Case Study:

SmartSignal Web Site:
www.smartsignal.com