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……

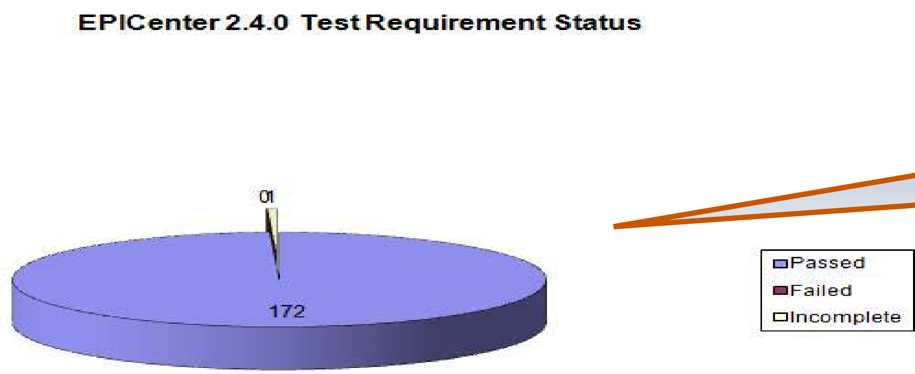
QA at SmartSignal

- 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.

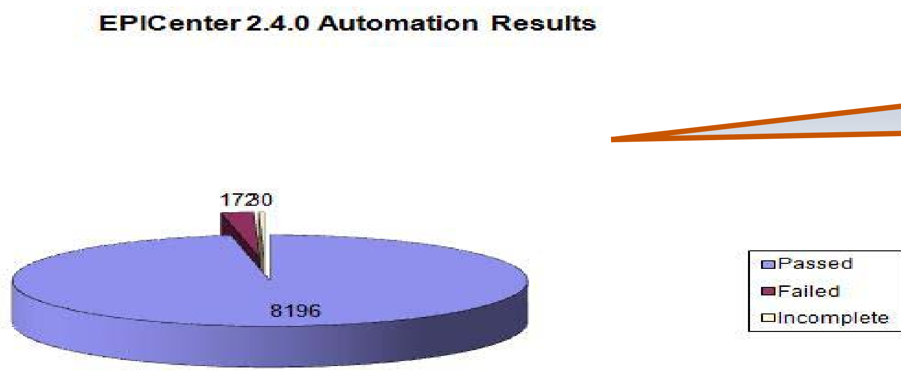
QA Project Tracking Dashboard

- 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



Status of new feature testing



Status of regression testing

Test Requirement Milestones

smartsignal

Test Requirements are high level

Status summarized in snapshot charts

EPICenter 2.4.0 Test Requirement	Res	Est	Status	Automation	12/5 Est	Act	12/12 Est	Act
1. Workbench								
1.1 Regression Testing								
1.1.1 Verify executing and verifying the US Test Bed.	GC	8	Passed	[illegible]	[illegible]	[illegible]	[illegible]	25.00%
1.1.2 Verify executing and verifying the International Test Bed.	GC	8	Passed	Complete	0.00%	0.00%	0.00%	0.00%
1.1.3 Verify executing and verifying the US Test Bed.	EN	4	Passed	Complete	0.00%	0.00%	50.00%	10.00%
1.1.4 Verify executing and verifying the International Test Bed.	EN	4	Passed	Incomplete	0.00%	0.00%	0.00%	0.00%
1.1.5 Verify executing and verifying the US Test Bed.	BB	4	Passed	Complete	0.00%	0.00%	50.00%	50.00%
1.1.6 Verify executing and verifying the International Test Bed.	BB	4	Passed	Complete	0.00%	0.00%	0.00%	0.00%
1.2 G2 Upgrade								
1.2.1 Verify executing and verifying the US Test Bed.	GC	8	Passed	Complete	100.00%	100.00%	100.00%	100.00%
1.2.2 Verify executing and verifying the US Test Bed.	EN	8	Passed	Complete	100.00%	100.00%	100.00%	100.00%
1.2.3 Verify executing and verifying the US Test Bed.	BB	8	Passed	Complete	100.00%	100.00%	100.00%	100.00%
1.3 Import Historical Data Merge								
1.3.2.1 Verify Historical Data merge for ASCII with new External and Calculated tag types.	EN	4	Passed	Incomplete	0.00%	[illegible]	[illegible]	25.00%
1.3.2.2 Verify Historical Data merge for ASCII with FLOAT/INTEGER/STRING/BOOLEAN data type tags.	EN	4	Incomplete	Inc[illegible]	[illegible]	[illegible]	[illegible]	25.00%
1.3.2.3 Verify Historical Data merge for ASCII by adding an external tag and modifying an existing Calculated tag.	EN	4	Passed	Inc[illegible]	[illegible]	[illegible]	[illegible]	0.00%
1.3.2.4 Verify Historical Data merge for ASCII filters correctly	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Break down by week and track to estimates

Automation Results

Version: 2.3.2	Version: 2.4.0	
[] Plan smartsignal.pln - 653 errors	[] Plan smartsignal.pln - 1113 errors	
[] Machine: (All Machines)	[] Machine: (All Machines)	
[] Elapsed: 287:25:11	[] Elapsed: 258:17:1	
[] Passed: 8191 tests (98.877354)	[] Passed: 8196 tests (97.944551)	Passed
[] Failed: 93 tests (1.122646)	[] Failed: 172 tests (2.055449)	Failed
[] Totals: 8284 tests, 653 errors, 11924 warnings	[] Totals: 8368 tests, 1113 errors, 10856 warnings	Incomplete
[] Plan smartsignal.pln - 308 errors	[] Plan smartsignal.pln - 305 errors	
[] Locale: US	[] Locale: US	
[] Elapsed: 125:3:5	[] Elapsed: 114:23:40	
[] Passed: 3769 tests (98.949856)	[] Passed: 3834 tests (99.018595)	
[] Failed: 40 tests (1.050144)	[] Failed: 38 tests (0.981405)	
[] Totals: 3809 tests, 308 errors, 5988 warnings	[] Totals: 3872 tests, 305 errors, 5489 warnings	
[] Plan smartsignal.pln - 345 errors	[] Plan smartsignal.pln - 808 errors	
[] Locale: GR	[] Locale: GR	
[] Elapsed: 162:22:6	[] Elapsed: 143:53:21	
[] Passed: 4422 tests (98.815642)	[] Passed: 4362 tests (97.019573)	
[] Failed: 53 tests (1.184358)	[] Failed: 134 tests (2.980427)	
[] Totals: 4475 tests, 345 errors, 5936 warnings	[] Totals: 4496 tests, 808 errors, 5367 warnings	
[] Locale: US	[] Locale: US	10486 (New)
[] Plan smartsignal.pln - 4 errors	[] Plan smartsignal.pln - 2 errors	10537 (New)
[] Machine: (GCERNY2003)	[] Machine: (GCERNY2003)	
[] Elapsed: 16:38:05	[] Elapsed: 15:31:43	
[] Passed: 299 tests (99.3%)	[] Passed: 305 tests (99.7%)	
[] Failed: 2 tests (0.7%)	[] Failed: 1 test (0.3%)	
[] Totals: 301 tests, 4 errors, 126 warnings	[] Totals: 306 tests, 2 errors, 29 warnings	
[] Developer: George Cerny	[] Developer: George Cerny	

Compare results to previous release

Summarize results at different levels

Compare Test Plans by locale

Track bugs that are causing failures

Performance Benchmark Results

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

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

```
⊞ Section 3 (GCERNYXP, QA2-GR-XP, QA20-GR-VISTA;QA28-GR-2
⊟ Section 4 (MDWA2003, QA6-GR-XP, QA22-GR-VISTA;QA30-GR
  ⊟ Workbench and WatchList Setup
    ⊞ 1: Setup Runtime/ASCII Error Notifications Assets
    ⊞ 7: Sensor Editing, Sensor Tag Mapping and Change D
    ⊞ 8: Create Complex Calculated Formula Test
    ⊞ 9: If_Exists Validation Tests
    ⊞ 10: Import Test Data where Tag doesn't exist
    ⊟ 11: Add Asset from MatLab Validation Tests
      ⊟ 11.1 Validation Testing and Asset Setup
        ⊟ 11.1.1: Create the Asset Instance: LABADIE U1 AH B -
          ◆ script: addassetfrommatlab.t
          ◆ testcase: T11_1_1 ("LABADIE U1 AH B")
        ⊞ 11.1.2: Verify LABADIE U1 AH Ba Test Model and outfile
        ⊟ 11.1.3: Create the Asset Instance: LABADIE U1 AH C - W
          ◆ script: addassetfrommatlab.t
          ◆ testcase: T11_1_1 ("LABADIE U1 AH C")
```

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

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

```
//-----------------------------------------------------------
// THIS TESTCASE IS USED TO VERIFY IMPORTING Asset Instance
//-----------------------------------------------------------

testcase T11_1_1 (STRING sInstanceName) appstate none
{
    //-----------------------------------------------------------
    // Parameter1: Instance Name of for the Asset to be created
    //-----------------------------------------------------------

    STRING sControllerFileDirectory;        // Directory to the MatLab

    if (sLocale == "US")
        sControllerFileDirectory = "C:\AutoSmartSignal\Autotest\Support\Maintenance\AutoFiles\ImportFromMatlab\{sInstanceName}";
    else
        sControllerFileDirectory = "C:\AutoSmartSignal\Autotest\Support\Maintenance\AutoFiles\ImportFromMatlab\{sInstanceName}_{sLocale}";


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

        if (bAAFM_ImportAsset (sControllerFileDirectory, sInstanceName) == TRUE)
        {
            print ("Asset: {sInstanceName} was created successfully.");
        }
        else
            logerror ("Couldn't open the Controller file in the Add Asset from MatLab Tool.");
    }
    except
    {
        ExceptLog ();
        CloseAll ();
    }
```

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

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

WINDOW AddAstMatLabToolOutputFileDir = AddAssetSFromMatlabTool.TextField("Matlab tool output file directory:");
WINDOW AddAstMatLabBrowse = AddAssetSFromMatlabTool.PushButton("Browse");
WINDOW AddAstMatLabCreateNewAssetsOnly = AddAssetSFromMatlabTool.PushButton("Create new assets only");

//-----------------------------
// INTERNATIONAL TAG HANDELING
//-----------------------------

STRING sChartColorWinTag = {"Color", "Farbe", "Color", "Couleurs"}[LANGUAGE];
WINDOW ChartColorWin = eCMSetup.DialogBox (sChartColorWinTag);

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

WINDOW MainWin_ModeldTag_Grid = eCMSetup.HistoricalDataSet.Frame1.OLEGrid1.Box;

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

window DialogBox AddAssetSFromMatlabTool
{
   tag "Add Asset(s) from Matlab tool files";
   parent IncidView_InternalTags;

}
window MainWin eCMSetup
{
    tag "*Workbench*";

    WindowsForms10Window8App6 HistoricalDataSet
    {
        tag "*Data*";

        WindowsForms10Window8App6 Frame1
        {
            tag "#1";

            OLEGrid OLEGrid1
            {
                tag "#1";

                StaticText Box
                {
                    tag "/({iColumn + 2}:{10}, {iRow + 1}:{43})";
```

Group dialogs logically

Uniquely identify objects

Embed localized data

Define frames once and then reference

Handle to custom objects

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

smartsignal

```
//----------------------------------------------------------------
// THIS FUNCTION IS USED TO IMPORT A ASSET FROM THE MATLAB TOOL
//----------------------------------------------------------------
BOOLEAN bAAFM_ImportAsset (STRING sControllerFileDirectory_Loc, STRING sInst
{
    //------------------------------------------------------------
    // Parameter 1: The directory Path to the controller file
    // Parameter 2: The Asset to be imported
    //------------------------------------------------------------

    INTEGER iWaitForImportToComplete;                   // Looping Variable used to ch
    BOOLEAN bSuccessfulImport = FALSE;                          // Boolean to determin

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

        if (bAAFM_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 ();
                        logerror (sGetText);
                        bSuccessfulImport = FALSE;
                        AddAstMatLabControllerErrorOK.Click ();
                        break;
                    }
                    if (!CreateAssetsFromMatlabFile.Exists ())
                    {
                        do
                        {
                            eCMSetup.SetActive ();
                            TreeViewControl.Select ("/#1");
                            bSuccessfulImport = TRUE;
                            break;
```

Function Objective

Comment on input parameters and variables

Use Meaningful Names

Verify each step is a success

Put a time limit on loops

Exit functions with a +/- result

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

smartsignal.

```
//-----------------------------------
// 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];
```

Initialize data specific to the machine

Initialize the localization variable

Any other machine specific information

Based on the locale initialization, set the data types

Data Structures and Data-Driven Testing

- Data Structures are a excellent way to pass data from data files through scripts and into the application.
- Data-Driven testing allows the tester to test a infinite number 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

```
````````````````````MODEL GENERAL INFORMATION`````````````````````````````````````````````````
/SSC_MODEL/AdaptationInhibit_FaultAll_Model1
FALSE
10
Minutes
GCTEST2
{"Take Off Gas Path Model"}
````````````````````MODELED TAG INFORMATION```````````````````````````````````````````````````
/SSC_MODEL/AdaptationInhibit_FaultAll_Model1
{"ALT","Yes","Adaptation","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"BV","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"CAS","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"CSHM","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"CSLM","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"WAI","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","113.073744503937","79.0199506414549","16.5109742463596","10","10"}
````````````````````MODEL CLASS TAG INFORMATION```````````````````````````````````````````````
/SSC_MODEL/AdaptationInhibit_FaultAll_Model1
````````````````````MODEL ALGORITHMS``````````````````````````````````````````````````````````
/SSC_MODEL/AdaptationInhibit_FaultAll_Model1
{"SSCOP3","Moving Average","","15","1.25","65","","TAT","ALT","MACH","EPR","FF"}
````````````````````MODEL ADAPTATION``````````````````````````````````````````````````````````
/SSC_MODEL/AdaptationInhibit_FaultAll_Model1
{"","None","","","","",""}
~
````````````````````MODEL GENERAL INFORMATION`````````````````````````````````````````````````
/SSC_MODEL/AdaptationInhibit_FaultAll_Model2
FALSE
10
Minutes
GCTEST1
{"Take Off Oil and Vibration Model"}
````````````````````MODELED TAG INFORMATION```````````````````````````````````````````````````
/SSC_MODEL/AdaptationInhibit_FaultAll_Model2
{"ALT","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"BV","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"CAS","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
{"CSHM","No","SSC_MODELED_TAG","SPRT","","","20","20","Independent","10.3584694320051","12.2000359710281","1.37180647973851","10","10"}
```

Separate Data file into logical sections

Store data by variable type

Use unique characters for easy parsing
```

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 sSMType;
    STRING sSMSplSmoFactor;
    STRING sSMNumObsSmoRes;
    STRING sVarScaleFactor;
    STRING sMaxHisQueSize;
    LIST OF STRING lsVirtSig;
    LIST OF STRING lsCustEstFF;

}
```

Data Structure name

Organize variables according to application order

Define variable types to match application data

Data Structures – Read Function Example

```
//--------------------------------------------------------------------------------------------------
// THIS FUNCTION IS USED TO EXTRACT PARMS FROM FILE AND ENTER INTO THE MODE CLASS - CLASS INFO TAB
//--------------------------------------------------------------------------------------------------

BOOLEAN bGetModelClassModeledTagParms (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 + "    Support\PlantECM\Classes\Model\";
    HFILE hReadParms;                                                                   // File Handeler
    ClassModelTagInfo dParmsModeledTagInfo;                                             // Modeled Tag Data Structure
    do
    {
        hReadParms = FileOpen (sBaseFolder + sAssetNameLoc + ".txt", FM_READ);
        FileReadValue (hReadParms, lsGetText);
        FileClose (hReadParms);
        dParmsModeledTagInfo.iTagID = iHoldRow;
        dParmsModeledTagInfo.sTagName = lsGetText[1];
        dParmsModeledTagInfo.sUsedInModel = lsGetText[2];
        dParmsModeledTagInfo.sClassType = lsGetText[3];
        dParmsModeledTagInfo.sAlarmType = lsGetText[4];
        dParmsModeledTagInfo.sResPlusThres = lsGetText[5];
        dParmsModeledTagInfo.sResNegThres = lsGetText[6];
        dParmsModeledTagInfo.sSPRTPlusSens = lsGetText[7];
        dParmsModeledTagInfo.sSPRTNegSens = lsGetText[8];
        dParmsModeledTagInfo.sInferred = lsGetText[9];
        dParmsModeledTagInfo.sMean = lsGetText[10];
        dParmsModeledTagInfo.sStdDev = lsGetText[11];
        dParmsModeledTagInfo.sResVar = lsGetText[12];
        dParmsModeledTagInfo.sOutlierPlusThres = lsGetText[13];
        dParmsModeledTagInfo.sOutlierNegThres = lsGetText[14];
        if (bEnterSubClassModeledTagInfo (dParmsModeledTagInfo, bAddTagsLoc) == FALSE)
        {
            logerror ("Failed to create Tags.");
            return FALSE;
        }
        else
            return TRUE;
    }
    except
    {
        ExceptLog ();
        return FALSE;
    }
```

Call the data structure handler

Open the data file and read the row

Parse the data read and assign it to the data structure

Pass the data structure into a function

Data Structures – Write Function Example

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

BOOLEAN bEnterSubClassModeledTagInfo (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.TypeKeys (dParmsLoc.sUsedInModel[1]);
        SubClassModelTagGrid.TypeKeys ("<Tab>");

        //-------------------------------
        // Select the Residual + Threshold
        //-------------------------------

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

        return TRUE;
    }
    except
    {
        logerror ("Enter Subclass Class Tag Information Function Failed.");
        ExceptLog ();
        CloseAll ();
        return FALSE;
    }
```

Data Structure is passed into the function

Data is written into the application

Localization example

Data Structures - Localization Example

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

BOOLEAN ConvertData (STRING sInputFile, STRING sOutputFile)
{

    //------------------------
    // Parm1: Input File Path
    // Parm2: Output File Path
    //------------------------

    hRead = FileOpen (sInputFile, FM_READ);
    hWrite = FileOpen (sOutputFile, FM_WRITE);
    while (FileReadLine (hRead, sGetText))
    {

        //------------------------------------------------------------------
        // Transform the demimetor and decimal characters to localized format
        //------------------------------------------------------------------

        sGetText = StrTran (sGetText, sInputDel, sListSep);
        sGetText = StrTran (sGetText, ".", sRSNumbers);

        //----------------------------------------
        // 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);
                    sGetText = Stuff (sGetText, 1, iRow - 1, sDay);
                    iRow = StrPos ("/", sGetText);
                    iColumn = StrPos ("/", sGetText, TRUE);
                    sGetText = Stuff (sGetText, iRow + 1, iColumn - iRow - 1, sMonth);
                }
                case "dd.mm.yyyy":
                {
                    sMonth = GetField (sGetText, "/", 1);
                    sDay = GetField (sGetText, "/", 2);
                    iRow = StrPos ("/", sGetText);
                    sGetText = Stuff (sGetText, 1, iRow - 1, sDay);
                    iRow = StrPos ("/", sGetText);
                    iColumn = StrPos ("/", sGetText, TRUE);
                    sGetText = Stuff (sGetText, iRow + 1, iColumn - iRow - 1, sMonth);
                    sGetText = StrTran (sGetText, "/", ".");
                }
```

Read data from file and create a new file

Convert delimiters and decimals to localized format

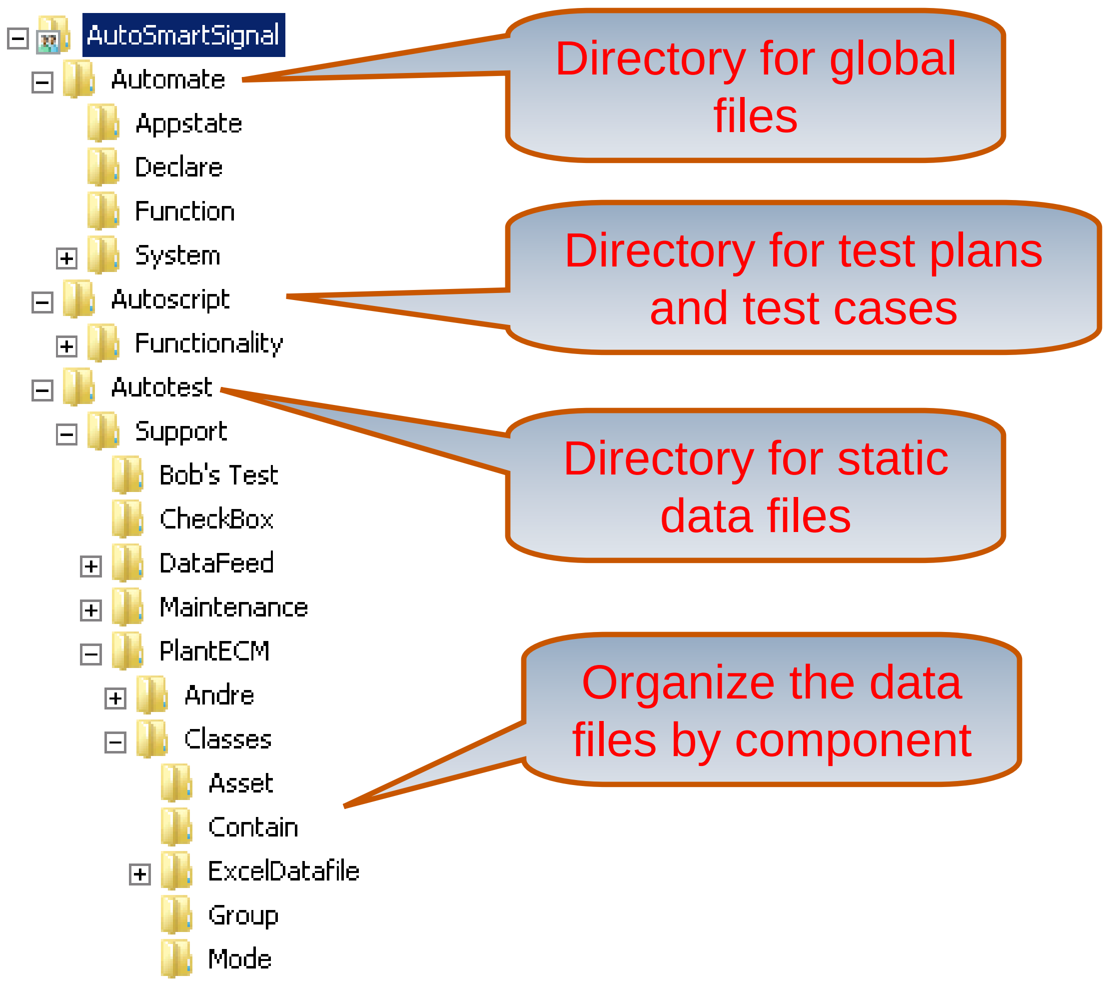
Convert dates

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



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

```
testcase T11_1 (STRING sFunctName, STRING sWhatToExecute) appstate none
{
    do
    {
        //---------------------------------------------------------------------
        // Open the Application State by passing the string using the @ symbol
        //---------------------------------------------------------------------
        if (@(sFunctName) () == TRUE)
        {
            switch (sFunctName)
            {
                //----------------------
                // Open Home Page Case
                //----------------------
                case "Open Home Page":
                {
                    switch (sWhatToExecute)
                    {
                        //----------------------
                        // Verify the SSL Lock
                        //----------------------
                        case "Verify SSL":
                        {
                            if (bVerifySSLIcon () == TRUE)
                                print ("The page: {sFunctName} was not loaded successfully, test {sWhatToExecute} succeeded.");
                            else
                                logerror ("The page: {sFunctName} was not loaded successfully, test {sWhatToExecute} failted.");
                        }
                        default:
                            logerror ("Invalid test key word: {sWhatToExecute}, test aborted."
                    }
                }
                default:
                    logerror ("Invalid page key word: {sFunctName}, test aborted.");
            }
        }
        else
        {
            logerror ("The page: {sFunctName} was not loaded successfully, test {sWhatToExe
        }
    }
    except
    {
        logerror ("Exception occurred running page: {sFunctName} and testcase: {sWhatToExecute}");
        ExceptLog ();
    }
}
```

Pass in keyword from plan or data file

Call functions by keyword name

Apply additional logic to script

Handle unsupported keywords

Key-Word Driven Interface Example

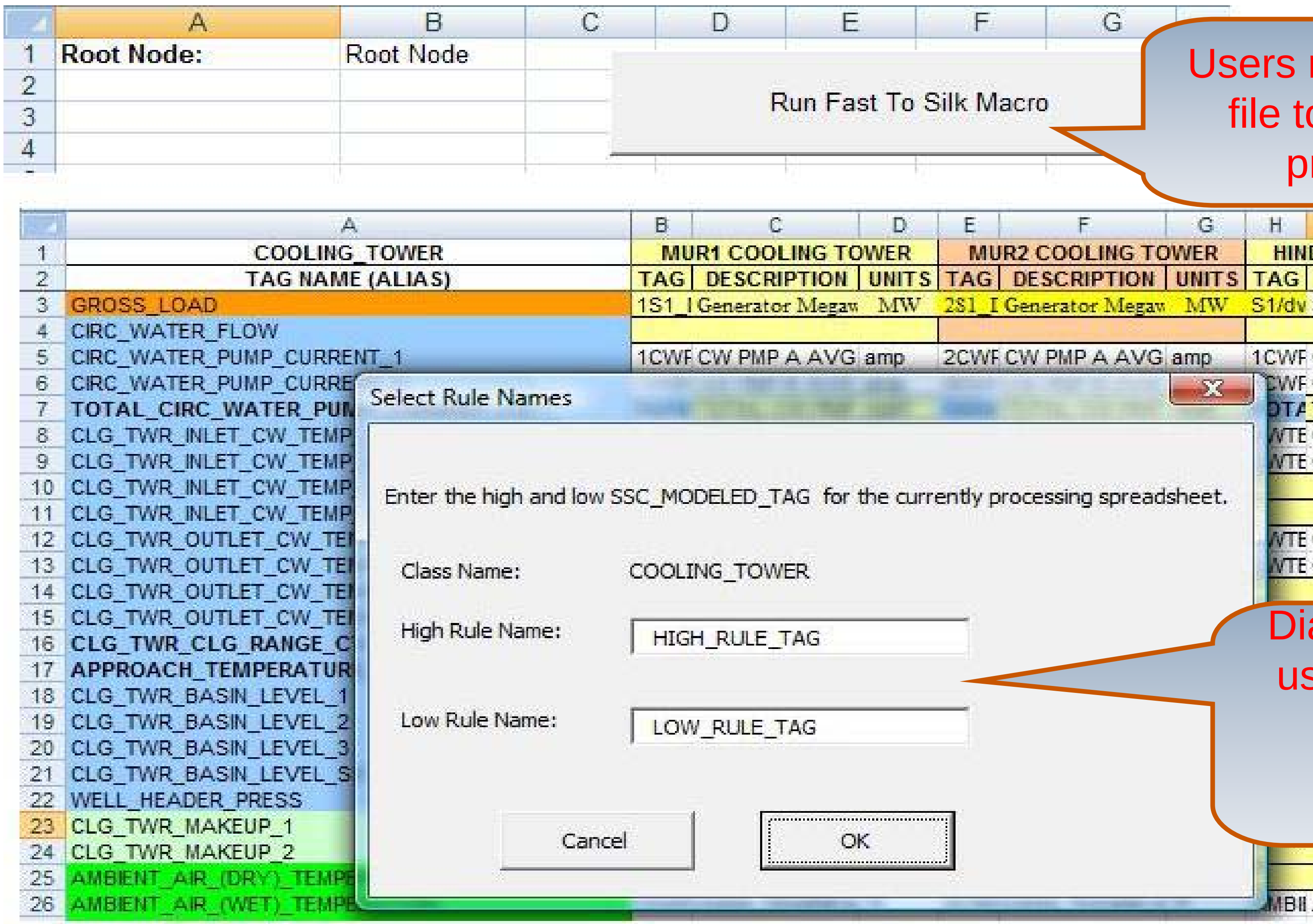
smartsignal

	A	B	C	D
1	COOLING_TOWER	MUR1 COOLING TOWER		
2	TAG NAME (ALIAS)	TAG	DESCRIPTION	UNITS
3	GROSS_LOAD	1S1_DWATT	Generator Megawatts	MW
4	CIRC_WATER_FLOW			
5	CIRC_WATER_PUMP_CURRENT_1	1CWP0100AAC_AI1_OUT		
6	CIRC_WATER_PUMP_CURRENT_2	1CWP		
7	**TOTAL_CIRC_WATER_PUMP_CURRENT_CV**	**TOTAL_CIRC_WATER_PUMP_CURRENT_CV**	T	
8	CLG_TWR_INLET_CW_TEMP_1	1CWTI0255_AI1_OUT	C	
9	CLG_TWR_INLET_CW_TEMP_2	1CWTI0265_AI1_OUT	C	
10	CLG_TWR_INLET_CW_TEMP_3			
11	CLG_TWR_INLET_CW_TEMP_4			
12	CLG_TWR_OUTLET_CW_TEMP_1	1CWTI0250_AI1_OUT	CONDENSER INLET A TEMP	°F
13	CLG_TWR_OUTLET_CW_TEMP_2			
14	CLG_TWR_OUTLET_CW_TEMP_3			
15	CLG_TWR_OUTLET_CW_TEMP_4	1CWTI0134A_AI1_OUT	CIRC WATER BASIN TEMP	°F
16	**CLG_TWR_CLG_RANGE_CV**	**TWR_CIRC_WTR_CLG_RANGE_CV**	COOLING TOWER CLG RANGE	DEG F
17	**APPROACH_TEMPERATURE_CV**	**APPROACH_TEMPERATURE_CV**	CLG TWR APPROACH	DEG F
18	CLG_TWR_BASIN_LEVEL_1	1CWLI0132_AI1_OUT	COOLING TOWER LEVEL	%
19	CLG_TWR_BASIN_LEVEL_2	1CW-LVL-SEL_AI2_OUT	COOLING TOWER LVL SELECT	%
20	CLG_TWR_BASIN_LEVEL_3	1CW-LVL-SEL_AI1_OUT	COOLING TOWER LVL SELECT	in
21	CLG_TWR_BASIN_LEVEL_SP	1CWLC0132_AO1_OUT	COOL TWR BASIN LEVEL CNTL	%
22	WELL_HEADER_PRESS			
23	CLG_TWR_MAKEUP_1	0MWFI0110_AI1_PV	MAKE-UP WATER FLOW	GPM
24	CLG_TWR_MAKEUP_2			
25	AMBIENT_AIR_(DRY)_TEMPERATURE	weather_device_Fahrenheit	Ambient Air Temp	F
26	AMBIENT_AIR_(WET)_TEMPERATURE	1CWTI0277	COOL TOWER AMBIENT TEMP	°F
27	BAROMETRIC_PRESSURE	1G1_AF	ometric Pressure Transducer 96AP	psia
28	TWR_WIND_SPEED_1			
29	TWR_WIND_SPEED_2			
30	TWR_WIND_DIRECTION_1			
31	TWR_WIND_DIRECTION_2			
32	RELATIVE_HUMIDITY	weather_device_AI1	Relative Humidity	%

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

COOLING_TOWER.txt - Notepad

File Edit Format View Help

```
Root Node
j
COOLING_TOWER    HIGH_RULE_TAG    LOW_RULE_TAG
```

Controller file created with keywords

COOLING_TOWER_ASSET CLASS SUMMARY.csv - Notepad

File Edit Format View Help

```
ENGINEERING DESIGN TOOL,Asset Level,ASSET LEVEL LIMITS,,MODEL,TAG CLAS
TAG NAME (ALIAS),,HIGH,LOW,PERFORMANCE,RULES,ON/OFF,POSITIVE,NEGATIVE,
GROSS_LOAD,EXT,,,YES,,,,,,,YES,,,,,,,YES,,,,,,,YES,YES,YES
CIRC_WATER_FLOW,EXT,,,NO,,,,,,,NO,,,,,,,,,,,,,,YES,YES,
CIRC_WATER_PUMP_CURRENT_1,EXT,,,NO,,,,,,,,,,,,,,,,,,,,,YES,,
CIRC_WATER_PUMP_CURRENT_2,EXT,,,NO,,,,,,,,,,,,,,,,,,,,,YES,,
TOTAL_CIRC_WATER_PUMP_CURRENT_CV,CALC,,,YES,,,,,,,YES,,,,,,,,,,,,,,YES,YE
CLG_TWR_INLET_CW_TEMP_1,EXT,,,YES,,,,,,,,,,,,,,,,,,,,,YES,,
CLG_TWR_INLET_CW_TEMP_2,EXT,,,YES,,,,,,,,,,,,,,,,,,,,,YES,,
CLG_TWR_INLET_CW_TEMP_3,EXT,,,YES,,,,,,,,,,,,,,,,,,,,,YES,,
CLG_TWR_INLET_CW_TEMP_4,EXT,,,YES,,,,,,,,,,,,,,,,,,,,,YES,,
CLG_TWR_OUTLET_CW_TEMP_1,EXT,,,YES,BOTH,,,,,,YES,,,,,,,,,,,,,,LINKED,YES,
CLG_TWR_OUTLET_CW_TEMP_2,EXT,,,YES,BOTH,,,,,,YES,,,,,,,,,,,,,,LINKED,YES
CLG_TWR_OUTLET_CW_TEMP_3,EXT,,,YES,BOTH,,,,,,YES,,,,,,,,,,,,,,LINKED,YE
CLG_TWR_OUTLET_CW_TEMP_4,EXT,,,YES,BOTH,,,,,,YES,,,,,,,,,,,,,,LINKED,YE
CLG_TWR_CLG_RANGE_CV,CALC,,,YES,BOTH,,,,,,,,,,,,,YES,,,,,,,LINKED,,YES
APPROACH_TEMPERATURE_CV,CALC,,,YES,BOTH,,,,,,,,,,,,,YES,,,,,,,LINKED,,Y
CLG_TWR_BASIN_LEVEL_1,EXT,,,YES,BOTH,,,,,,YES,,,,,,,,,,,,,,LINKED,YES,
CLG_TWR_BASIN_LEVEL_2,EXT,,,YES,BOTH,,,,,,YES,,,,,,,,,,,,,,LINKED,YES,
CLG_TWR_BASIN_LEVEL_3,EXT,,,YES,BOTH,,,,,,YES,,,,,,,,,,,,,,LINKED,YE
CLG_TWR_BASIN_LEVEL_SP,EXT,,,YES,,,,,,,,,,,,,,,,,,,,,YES,,
WELL_HEADER_PRESS,EXT,,,YES,BOTH,,,,,,,,,,,,,,,,,,,,LINKED,,
CLG_TWR_MAKEUP_1,EXT,,,YES,,,,,,,YES,BOTH,,,,,,,,,,,,,YES,LINKED,
CLG_TWR_MAKEUP_2,EXT,,,YES,,,,,,,YES,BOTH,,,,,,,,,,,,,YES,LINKED,
AMBIENT_AIR_(DRY)_TEMPERATURE,EXT,,,YES,,,,,,,YES,,,,,,,YES,,,,,,,YES,YES,YES
AMBIENT_AIR_(WET)_TEMPERATURE,EXT,,,YES,,,,,,,YES,,,,,,,YES,,,,,,,YES,YES,YES
BAROMETRIC_PRESSURE,EXT,,,YES,,,,,,,,,,,,,,YES,,,,,,,YES,,YES
TWR_WIND_SPEED_1,EXT,,,YES,,,,,,,,,,,,,,YES,,,,,,,YES,,YES
TWR_WIND_SPEED_2,EXT,,,YES,,,,,,,,,,,,,,YES,,,,,,,YES,,YES
TWR_WIND_DIRECTION_1,EXT,,,YES,,,,,,,,,,,,,,YES,,,,,,,YES,,YES
```

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

```
⊟ Maintenance Procedures
  ⊟ Power/Airline Scripts
      ◆ script: maintenancescripts.t
      ◆ $StopTime = ""    // The time to stop running
      ◆
    ⊟ Execute Main Script
        ◆ testcase: MainScript ($StopTime)
```

Users open a automation plan and start execution

```
//-------------------------------------------------------------------------------------------------------------
// 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, sscadi decision, window size, alarm type,
// +/- residual threshold, +/- sprt sensitivity, inferred, mean, std dev, res var: repeate 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, )
// repeate model loop:, check the asset in
//
// Controller DataFile: (any name)
//      line1: Path to Root Category (ex: Delta Aircraft/B767-300?300ER?400 : CF6-80C2B6?B8)
//      line2: Model Parms (ex: h,"C:\Public\HistoricalDataFile.txt")
//          parm1: 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 modes sep
//          parm1: Sensor Name
//          parm2: Comparisn Operator
//          parm3 - n-1: Comparisn value
//          parmn: eof
//-------------------------------------------------------------------------------------------------------------

case "j":
{

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

    sNewAsset = GetField (lsCategories[iLoop], "    ", 1);
    if (MatchStr ("*    *", lsCategories[iLoop]))
```

Keywords are used to drive the data entry functions and logic

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:

  http://www.borland.com/resources/en/pdf/case_studies/smart_signal.pdf
- SmartSignal Web Site:

  www.smartsignal.com