```
CREATE proc [dbo].[LabDataImportReactor] @Operation varchar(25), @TestSheet varchar(255)='', @TemplateID varchar(50)='', @TestID
varchar(50)='',
      @SheetSubID varchar(50)='', @SampleID varchar(50)='', @TestCfgID int=0,
      @ExistAction varchar(20)='error', @MeasurementGroup smallint=0, @DataIntegrityRules varchar(50)='UnknownFactor|Boundaries|Format',
      @RetStyle varchar(25)='RowSet', @RetStatus varchar(25)='' output as

-- identify completed reactor test sheets from data contained within sheet combined with official template identification rules
-- read completed sheets to extract identification data
-- read completed test results from sheet using cell-to-test-parameter map corresponding to identified template and test type

-- parameters:

-- @Operation          - 'IdentifySheet'  to return list of all templates and tests identified in specified sheet
--                     - 'SampleTests'    to return list of TestIDs from LabTest records containing specified @SampleID and @TestCfgID
--                     - 'ImportTestData' to create a test and populate results from specified sheet using cell-parameter mapping corresponding
to
--                     -                  @TemplateID and @TestID (from a previous template/test identification)
-- @TestSheet          - complete Windows path and file name of test sheet to be analyzed
-- @TemplateID         - template ID from previous sheet identification (used in data import)
-- @TestID             - test ID from previous sheet identification (used in data import)
-- @SampleID           - sample ID used in SampleTests and ImportTestData options
-- @TestCfgID          - sample ID used in SampleTests and ImportTestData options
-- @ExistAction        - 'NewMG' to import duplicate tests (data for test cfg and sample already exist) to new measurement groups
--                     - used only when @MeasurementGroup=0
-- @MeasurementGroup   - measurement group to assign to generated LabTest record
--                     - if a non-zero mg is specified, prior existence is tested and @ExistAction='NewMG' is ignored
--                     - if 0 is specified and @ExistAction<>'NewMG', then the created LabTest record receives a MeasurementGroup of 1
-- @DataIntegrityRules - list of data integrity rules to enforce (separated by |)
--                     - 'UnknownFactor' - return error when any map definitions reference factors that do not appear in resolved set of test
results
--                     - 'Boundaries'    - return error when any retrieved data value is outside of those configured for the associated factor
--                     - 'Format'        - return error when any retrieved data value has format different from that specified in corresponding
--                     -                   factor cfg - 'Format' not specified -> bypass creation of lab result for violators
-- @RetStyle           - 'RowSet' to return table of imported values and messages
--                     - any other value to suppress output rowset generation
-- @RetStatus          - 'ok' when no error conditions detected
--                     - 'Error' plus possible error explanation

set nocount on

create table #SheetData(RecID smallint identity)
create table #TestData(TemplateID varchar(50), TestID varchar(50), SheetSubID varchar(50), Tab varchar(50),
       CellRange varchar(10), Operator varchar(10), Operand varchar(50), SheetValue varchar(255), RecID int,
       TestIDRuleSatisfied tinyint default 0, FactorCategory varchar(25), Factor varchar(25), ResultCfgID int,
       DataFormat varchar(10), msg varchar(255), CreateTestResultRec bit)
create table #SheetDescriptorData(TemplateID varchar(50), Descriptor varchar(25), SheetValue varchar(50),
       RecID smallint, msg varchar(255))
declare @Tab varchar(50), @Cell varchar(10), @Operator varchar(20), @Operand varchar(50), @sqltext varchar(4000), @Descriptor varchar(50),
        @row0 varchar(10), @col0 varchar(10), @row1 varchar(10), @col1 varchar(10), @Range varchar(10), @ncol smallint, @rbase varchar(10),
        @i smallint, @k smallint, @FactorCategory varchar(25), @Factor varchar(25), @done as bit, @a varchar(255), @b varchar(255),
        @EnforceRuleUnknownFactor bit, @EnforceRuleBoundaries as bit, @EnforceRuleFormat bit,
        -- note the typing of @RecID to int, to accomodate int identity values returned by scope_identity()
        @RecID int
declare @ListElement table(RecID smallint, Val varchar(255))
```

```
if(@Operation='IdentifySheet')

  begin

    -- read template identification tab(s) from specified sheet
    -- one template cfg is returned, with associated identifiers, for each template identified

    -- add cols to SheetData table as specified in CellRange cfg parameter
    alter table #SheetData add Tab varchar(50)
    select @Range=CellRange from ReactorDatamanagementCodes where ClassA='Map' and ClassB='TemplateIDTab' and ClassC='CellRangeLowerRight'
    if(@Range is null)
      select @Range='R20'
    -- extract column of lower right corner cell of block to read (in format M24, R45, AB132, etc)
    select @ncol=case when(@Range like '[a-z][0-9]%')then
                        -- single letter in cell ID
                        ascii(left(upper(@Range),1))-64
                      when(@Range like '[a-z][a-z][0-9]%')then
                        -- two letters in cell ID
                        26*(ascii(left(upper(@Range),1))-64)+ascii(substring(upper(@Range),2,1))-64
                      else 0
                end
    -- add cols to SheetData table
    select @i=0
    while(@i<@ncol)
      begin
        -- assign col ID (A ... B, AA ... AZ, BA ... BZ, ...)
        select @k=@i/26
        select @col0='c' + case when(@k=0)then '' else char(64+@k) end + char(65+@i%26)
        select @sqltext='alter table #SheetData add ' + @col0 + ' varchar(1000)'
        exec(@sqltext)
         select @i=@i+1
      end

    -- identify test templates
    -- read entire contents of all template ID tabs from which to extract test ID data elements
    -- identify with tab name
    -- note that, in a static query (select * from sheet$a1:z100), fewer than the number of cols requested
    -- are returned when the rightmost occupied (or formatted) sheet col precedes the rightmost requested col
    -- this causes an error when attempting to insert into an a-priori, fixed set of columns
    -- therefore, read "safe" sheet dimensions and tailor temp table column count accordingly
    -- note, also, that openrowset returns columns beginning with the leftmost one that contains data or formatting
    -- (if col A is empty, col B non-empty, then col B is returned as F1)
    -- and ending with the rightmost column with data or formatting, all intermediate columns are returned
    -- and contain nulls when empty or unformatted
    select @done=0
    declare TemplateIDTab cursor for
            select   Tab, Operator
            from     ReactorDatamanagementCodes
            where    ClassA='Map' and ClassB='TemplateIDTab' and ClassC='Tab'
            order by AppearanceOrder
    open TemplateIDTab
    fetch next from TemplateIDTab into @Tab, @Operator
    while(@@fetch_status=0 and @done=0)
      begin
        begin try
```

```
        -- query specified range of test sheet
        -- note that, although explicit columns are not supplied on the insert statement, the first column (RecID) is
        -- an identity and therefore does not need to appear
        -- therefore, providing there are sufficient populated columns in the queried sheet, the number of returned
        -- columns equals the number just added to #SheetData
        -- clear data from past template ID sheet
        select @sqltext='select ''' + @Tab + ''', *
                        from   openrowset(''Microsoft.ACE.OLEDB.12.0'',
                                          ''excel 12.0;database=' + @TestSheet + ';hdr=no;imex=1'',
                                          ''select * from [' + @Tab + '$A1:' + @Range + ']'' )'
        insert into #SheetData exec(@sqltext)
        -- if here, then tab was found
        -- if tab's operation is 'continue' then continue testing for additional id tabs, otherwise stop looking (and move on to something
else)
        if(@Operator<>'continue')
          select @done=1
      end try
      begin catch
        -- ignore 7350 ('Cannot process the object "select * from tab name ...')
        -- and 7357 ('Cannot get the column information from OLE DB provider ...')
        -- since it indicates that the template ID tab to search for was not found
        if(error_number() not in(7350, 7357))
          insert into #SheetDescriptorData(TemplateID, Descriptor, msg) values('', 'TemplateIDErr', @Tab + ':  ' + error_message())
      end catch
      fetch next from TemplateIDTab into @Tab, @Operator
    end
  deallocate TemplateIDTab

  -- combine each tab's identification rules with associated sheet data
  -- read all documented rules and test each against corresponding cells of each template ID tab previously read
  -- note that all rule definitions are saved first with independent attempt to retrieve associated data from sheet
  -- so that all rules are recorded - those with no associated sheet data have null SheetValue columns
  insert into #TestData(Tab, TemplateID, TestID, SheetSubID, CellRange, Operator, Operand, Factor)
  select distinct SheetTab.Tab, ReactorDataManagementCodes.ID1 as TemplateID, ReactorDataManagementCodes.ID2 as TestID,
         ReactorDataManagementCodes.ID3 as SheetSubID, ReactorDataManagementCodes.CellRange, ReactorDataManagementCodes.Operator,
         ReactorDataManagementCodes.Operand, ReactorDataManagementCodes.Factor
  from   (select distinct Tab from #SheetData) SheetTab cross join ReactorDataManagementCodes
  where  ReactorDataManagementCodes.ClassA='Map' and ReactorDataManagementCodes.ClassB='TestID'

  -- post sheet data to corresponding rule records
  -- note that all cells referenced in rules are queried in sheet data
  -- values are posted to rules in #TestData by tab, so that template IDs are identified by
  -- having all rules satisified within a corresponding tab
  declare  TestSheetIDRule cursor for
    select distinct #TestData.Tab, #TestData.CellRange, convert(varchar(10),Radj.base)
    from   #TestData
           left join ( -- find last sheet data record ID prior to the first one for each tab
                       -- recall that rec ID is an identity col, making it sequential
                       -- subtracting the greatest ID of all other tabs below aligns rec ID with cell rows
                       select   s.Tab, isnull(max(sbelow.RecID),0) as base
                       from     #SheetData s left join #SheetData sbelow on s.Tab<>sbelow.Tab and sbelow.RecID<s.RecID
                       group by s.Tab
                     ) Radj on #TestData.Tab=Radj.Tab
  open TestSheetIDRule
  fetch next from TestSheetIDRule into @Tab, @Cell, @rbase
```

```
while(@@fetch_status=0)
  begin
    -- query specified sheet cell and save contents with rule identifiers
    if(substring(@Cell,2,1) between '0' and '9')
      select @row0=right(@Cell,len(@Cell)-1), @col0=left(@Cell,1)
    else
      select @row0=right(@Cell,len(@Cell)-2), @col0=left(@Cell,2)
    select @sqltext='declare @a varchar(255)
                     select @a=c' + @col0 + '
                     from   #SheetData
                     where  Tab=''' + @Tab + ''' and RecID=' + @rbase + '+' + @row0 + '
                     update #TestData set SheetValue=@a
                     where  Tab=''' + @Tab + ''' and CellRange=''' + @Cell + ''''
    exec(@sqltext)
    fetch next from TestSheetIDRule into @Tab, @Cell, @rbase
  end
deallocate TestSheetIDRule

-- test sheet identification rules
-- not empty
update #TestData set TestIDRuleSatisfied=1
where  replace(Operator,' ','')='NotEmpty' and isnull(SheetValue,'')<>''
-- is empty
update #TestData set TestIDRuleSatisfied=1
where  replace(Operator,' ','')='IsEmpty' and isnull(SheetValue,'')=''
-- equality
update #TestData set TestIDRuleSatisfied=1
where  replace(Operator,' ','')='=' and ltrim(rtrim(isnull(SheetValue,'')))=ltrim(rtrim(Operand))
-- like
update #TestData set TestIDRuleSatisfied=1
where  replace(Operator,' ','')='like' and ltrim(rtrim(isnull(SheetValue,''))) like '%'+ltrim(rtrim(Operand))+'%'
-- in list
-- surround each list element (comma separated) with spaces
-- then identify cases where padded sheet value pattern found in operand
update #TestData set TestIDRuleSatisfied=1
where  replace(Operator,' ','')='in' and ' ' + replace(Operand,',',' ') + ' ' like '% ' + SheetValue + ' %'

-- collect descriptive data for templates with at least one identified test (a test with all identification rules satisfied)
select @RecID=0
declare TestSheetDescriptor cursor for
  select  RulesUp.Tab, ReactorDataManagementCodes.ID1 as TemplateID, ReactorDataManagementCodes.ID2 as Descriptor,
          ReactorDataManagementCodes.CellRange
  from    ReactorDataManagementCodes
          join ( -- list tabs and template IDs with all identification rules satisfied
                 select   Tab, TemplateID
                 from     #TestData
                 group by Tab, TemplateID, TestID
                 having   sum(TestIDRuleSatisfied)=count(1)
               ) RulesUp on ReactorDataManagementCodes.ID1=RulesUp.TemplateID
  where   ReactorDataManagementCodes.ClassA='Map' and ReactorDataManagementCodes.ClassB='SheetDescriptor'
open    TestSheetDescriptor
fetch   next from TestSheetDescriptor into @Tab, @TemplateID, @Descriptor, @Cell
while(@@fetch_status=0)
  begin
    -- save current descriptor data
```

```
        -- note that the descriptor definition is saved with subsequent, independent attempt to retrieve associated data from sheet
        -- so that all descriptors are recorded - those with no associated sheet data have null SheetValue columns
        select @RecID=@RecID+1
        insert into #SheetDescriptorData(TemplateID, Descriptor, RecID)
        select @TemplateID, @Descriptor, @RecID
        -- query specified sheet cell
        -- note that data are read from tab corresponding to successfully identified template ID, so that
        -- identification data appropriate for that template are returned
        if(substring(@Cell,2,1) between '0' and '9')
          select @row0=right(@Cell,len(@Cell)-1), @col0=left(@Cell,1)
        else
          select @row0=right(@Cell,len(@Cell)-2), @col0=left(@Cell,2)
        select @sqltext='declare @a varchar(255)
                         select @a=c' + @col0 + '
                         from   #SheetData
                         where  Tab=''' + @Tab + ''' and RecID=' + @row0 + '
                         update #SheetDescriptorData set SheetValue=@a
                         where  RecID=' + convert(varchar(10),@RecID)
        exec(@sqltext)
        fetch next from TestSheetDescriptor into @Tab, @TemplateID, @Descriptor, @Cell
      end
    deallocate TestSheetDescriptor

    -- return results
    -- to avoid an independent (duplicate) read of sheet (once for template ID level, once for identifying data for each template),
    -- all idetifying data are returned in a list ordered by template ID and identifying data element
    -- a client can use the list to generate a two level heierachical display of data elements (level one for template, level two for
    -- identifying data)
    select   Template.TemplateID, Data.DataElement, Data.DataValue1, Data.DataValue2, Data.DataValue3, Data.AppearanceOrder,
             Data.msg as msg
    from     ( -- list individual identified templates
               select distinct TemplateID
               from   #SheetDescriptorData
             ) Template join
             ( -- list all data elements in order of appearance
               -- list errors first
               select TemplateID, 'TemplateID' as DataElement, null as DataValue1, null as DataValue2, null as DataValue3, msg, 0 as
AppearanceOrder
               from   #SheetDescriptorData
               where  Descriptor='TemplateIDErr'
               union
               -- template ID element has template version in DataValue2, subtype in DataValue3
               select Template.TemplateID, 'TemplateID' as DataElement, Template.TemplateID as DataValue1, Template.SheetValue as DataValue2,
                      case when(len(SubType.SheetValue)>0)then
                             upper(left(SubType.SheetValue,1)) + lower(right(SubType.SheetValue,len(SubType.SheetValue)-1))
                           else ''
                      end as DataValue3,
                      isnull(Template.msg, SubType.msg) as msg, 10 as AppearanceOrder
               from   #SheetDescriptorData Template left join #SheetDescriptorData SubType on SubType.Descriptor='SheetSubID'
               where  Template.Descriptor='SheetVersion'
               union
               select TemplateID, 'ProjectID' as DataElement, SheetValue as DataValue1, null as DataValue2, null as DataValue3,
                      msg, 20 as AppearanceOrder
               from   #SheetDescriptorData
               where  Descriptor='ProjectID'
```

```
union
select TemplateID, 'ProjectDescription' as DataElement, left(CommercialData.Description,25) as DataValue1,
       null as DataValue2, null as DataValue3, msg, 30 as AppearanceOrder
from   #SheetDescriptorData left join CommercialData
       on #SheetDescriptorData.SheetValue=CommercialData.ProjectID
where  #SheetDescriptorData.Descriptor='ProjectID'
union
select TemplateID, 'SheetDate' as DataElement,
       case when(isdate(SheetValue)=1)then convert(varchar(10),convert(smalldatetime,SheetValue),101) else SheetValue end as
DataValue1,
       null as DataValue2, null as DataValue3, msg, 40 as AppearanceOrder
from   #SheetDescriptorData
where  Descriptor='SheetDate'
union
select TemplateID, 'Layers' as DataElement, SheetValue as DataValue1, null as DataValue2, null as DataValue3,
       msg, 50 as AppearanceOrder
from   #SheetDescriptorData
where  Descriptor='Layers'
union
select TemplateID, 'Sample1' as DataElement, SheetValue as DataValue1, null as DataValue2, null as DataValue3,
       msg, 60 as AppearanceOrder
from   #SheetDescriptorData
where  Descriptor='SampleID1'
union
select TemplateID, 'Sample2' as DataElement, SheetValue as DataValue1, null as DataValue2, null as DataValue3,
       msg, 61 as AppearanceOrder
from   #SheetDescriptorData
where  Descriptor='SampleID2'
union
select TemplateID, 'Sample3' as DataElement, SheetValue as DataValue1, null as DataValue2,null as DataValue3,
       msg, 62 as AppearanceOrder
from   #SheetDescriptorData
where  Descriptor='SampleID3'
union
select TemplateID, 'Sample4' as DataElement, SheetValue as DataValue1, null as DataValue2,null as DataValue3,
       msg, 63 as AppearanceOrder
from   #SheetDescriptorData
where  Descriptor='SampleID4'
union
-- include list of tests for which all test ID rules were satisfied
-- DataValue2 is the LabTestCfgID, DataValue3 is the sample ID corresponding to the test
select distinct TestData.TemplateID, 'TestID' as DataElement,
       TestData.TestID as DataValue1,
       convert(varchar(10),isnull(LabTestCfg.TestCfgID,0)) as DataValue2,
       TestData.SampleID as DataValue3,
       TestData.msg,
       70+isnull(IdentifiedTestCfgID.AppearanceOrder,isnull(IdentifiedTestCfgIDDefault.AppearanceOrder,0)) as AppearanceOrder
from   ( -- generate list of templates and tests with all test ID rules satisified
         select   TemplateID, TestID,
                  -- factor='TestSample' instructs to associate sample ID of rule record with retrieved test data
                  max(case when(Factor='TestSampleID')then SheetValue else null end) as SampleID,
                  max(msg) as msg
         from     #TestData
         group by Tab, TemplateID, TestID
         having   sum(TestIDRuleSatisfied)=count(1)
```

```
                        ) TestData
                        -- get test cfg IDs corresponding to identified tests
                        -- note the use of left joins, so that a test cfg record is returned when the sample does not exist
                        -- or when a sample's item/sample cfg does not have a test corresponding to the indicated test type (system, dP, etc.)
                        -- it is expected that if a particular test is identified for a sample then the test applies and a test cfg ID
                        -- should also be identifiable
                        -- therefore, identified tests are always returned, although missing test cfg IDs generate error messages
                        -- if upload is attempted
                        -- failing to present identified tests, due to absent LabTestCfg records, gives the user has no indication that
                        -- test data exist and that LabTestCfg records must be configured
                        -- first, get test cfg IDs for sheet sub type (if they exist)
                        left join #SheetDescriptorData SheetSubID on SheetSubID.Descriptor='SheetSubID'
                        left join ReactorDataManagementCodes IdentifiedTestCfgID on IdentifiedTestCfgID.ClassA='Map'
                        and IdentifiedTestCfgID.ClassB='LabTestCfgID' and TestData.TemplateID=IdentifiedTestCfgID.ID1
                        and TestData.TestID=IdentifiedTestCfgID.ID2 and IdentifiedTestCfgID.ID3=SheetSubID.SheetValue
                        -- now default test cfg IDs for when none associated with sheet sub ID or when sheet sub ID not specified
                        left join ReactorDataManagementCodes IdentifiedTestCfgIDDefault on IdentifiedTestCfgIDDefault.ClassA='Map'
                        and IdentifiedTestCfgIDDefault.ClassB='LabTestCfgID' and TestData.TemplateID=IdentifiedTestCfgIDDefault.ID1
                        and TestData.TestID=IdentifiedTestCfgIDDefault.ID2 and isnull(IdentifiedTestCfgIDDefault.ID3,'')=''
                        -- get item sample cfg IDs
                        left join SampleMaster on TestData.SampleID=SampleMaster.SampleID
                        left join ItemMaster on SampleMaster.ItemID=ItemMaster.ItemID
                        left join ItemSampleCfg on ItemMaster.ItemCfgID=ItemSampleCfg.ItemCfgID
                        and SampleMaster.SampleCfgID=ItemSampleCfg.SampleCfgID
                        -- check for configured tests
                        -- note that applicable test cfg IDs are in a comma delimited list (Operand col of IdentifiedTestCfgID records)
                        -- padding spaces on elements of the list prevents confounding of, say, 50 with 500 in the a like comparison
                        -- note the use of default test cfg ID records if none associated with current sheet sub ID
                        left join LabTestCfg on ItemSampleCfg.ItemSampleCfgID=LabTestCfg.ItemSampleCfgID
                        and ' ' + replace( isnull(IdentifiedTestCfgID.Operand,IdentifiedTestCfgIDDefault.Operand),',',' , ')
                                        + ' ' like '% ' + isnull(convert(varchar(10),LabTestCfg.TestCfgID),'') + ' %'
                ) Data on Template.TemplateID=Data.TemplateID
        order by Template.TemplateID, Data.AppearanceOrder, Data.DataValue1, Data.DataValue2, Data.DataValue3

      end

    else if(@Operation='SampleTests')

      -- return list of TestIDs from LabTest records that contain specified SampleID and TestCfgID
      select   TestID, MeasurementGroup, ChangeDate as TestDate
      from     LabTest
      where    SampleID=@SampleID and TestCfgID=@TestCfgID
      order by TestDate

    else if(@Operation='ImportTestData')

      -- import test data from specified sheet into new test records for specified sample and test (@TestCfgID)

      begin

        -- verify existence of sample
        if(exists(select * from SampleMaster where SampleID=@SampleID))

          -- verify compatibility with specified test cfg ID
          if(exists(select *
```

```
                from   SampleMaster join ItemMaster on SampleMaster.ItemID=ItemMaster.ItemID
                       join ItemSampleCfg on ItemMaster.ItemCfgID=ItemSampleCfg.ItemCfgID and
SampleMaster.SampleCfgID=ItemSampleCfg.SampleCfgID
                       join LabtestCfg on ItemSampleCfg.ItemSampleCfgID=LabTestCfg.ItemSampleCfgID
                where  SampleMaster.SampleID=@SampleID and LabTestCfg.TestCfgID=@TestCfgID))

        begin

          -- test for prior import for specified test, sample, and measurement group
          if(@MeasurementGroup<>0)
            begin
              -- return invalid group if test exists
              if(exists(select * from LabTest where SampleID=@SampleID and TestCfgID=@TestCfgID and MeasurementGroup=@MeasurementGroup))
                select @MeasurementGroup=0
            end
          else if(@ExistAction='NewMG')
            begin
              -- increment last measurement group on record
              select @MeasurementGroup=max(MeasurementGroup) from Labtest where SampleID=@SampleID and TestCfgID=@TestCfgID
              select @MeasurementGroup=isnull(@MeasurementGroup,0)+1
            end
          else if(not exists(select * from LabTest where SampleID=@SampleID and TestCfgID=@TestCfgID and MeasurementGroup=1))
            -- use default measurement group, if not already assigned
            select @MeasurementGroup=1
          else
            select @MeasurementGroup=0

          if(@MeasurementGroup<>0)

            begin try

              -- parse data rules to enforce
              select @EnforceRuleUnknownFactor=case when('|'+replace(@DataIntegrityRules,' ','')+'|' like '%|UnknownFactor|%')then 1 else 0
end,
                     @EnforceRuleBoundaries=case when('|'+replace(@DataIntegrityRules,' ','')+'|' like '%|Boundaries|%')then 1 else 0 end,
                     @EnforceRuleFormat=case when('|'+replace(@DataIntegrityRules,' ','')+'|' like '%|Format|%')then 1 else 0 end

              -- to avoid sheet cell to result value mapping misalignment due to empty sheet columns (described in comments above)
              -- each configured result mapping is queried individually (non-existent or empty cells/columns return null)

              -- create a col to store retrieved data
              alter table #SheetData add x varchar(255)

              -- get sheet data cell mappings, result cfg IDs, and formats
              -- note the use of a left join to LabResultCfg, to permit direct encoding (constants) within ReactorManagementCodes
              -- (for reactor ID, for instance, which is omitted in test sheets where only one rector conducts such a test)
              insert into #TestData(Tab, CellRange, Operator, Operand, FactorCategory, Factor, ResultCfgID, Dataformat, CreateTestResultRec)
              select ReactorDataManagementCodes.Tab, ReactorDataManagementCodes.CellRange, ReactorDataManagementCodes.Operator,
                     ReactorDataManagementCodes.Operand, ReactorDataManagementCodes.FactorCategory, ReactorDataManagementCodes.Factor,
                     LabResultCfg.ResultCfgID, FactorCfg.DataFormat,
                     case when(ReactorDataManagementCodes.FactorCategory in('Condition','Result'))then 1 else 0 end
              from   ReactorDataManagementCodes left join FactorCfg on ReactorDataManagementCodes.FactorCategory=FactorCfg.Category
                     and ReactorDataManagementCodes.Factor=FactorCfg.Factor
                     left join LabResultCfg on FactorCfg.FactorID=LabResultCfg.FactorID and LabResultCfg.TestCfgID=@TestCfgID
              where  ReactorDataManagementCodes.ClassA='Map' and ReactorDataManagementCodes.ClassB='TestData'
```

```
       and ReactorDataManagementCodes.ID1=@TemplateID and ReactorDataManagementCodes.ID2=@TestID
       and ReactorDataManagementCodes.ID3=@SheetSubID and isnull(LabResultCfg.UploadLab,1)=1

-- test for unknown factors
if(@EnforceRuleUnknownFactor=1)
  update #TestData set msg='Error:  Unknown factor (' + FactorCategory + '/' + Factor + ')'
  where  CreateTestResultRec=1 and isnull(msg,'') not like 'Error:%' and ResultCfgID is null
else
  update #TestData set CreateTestResultRec=0
  where  #TestData.CreateTestResultRec=1 and ResultCfgID is null

if(not exists(select * from #TestData where msg like 'error%'))

begin

  -- identify result cfg records to be mated with sheet results
  -- note that all begin with null sheet values to facilitate identification (later) of missing data

  -- uniquely identify each result record
  select @RecID=0
  while(exists(select * from #TestData where RecID is null))
    begin
      select @RecID=@RecID+1
      update top(1) #TestData set RecID=@RecID
      where  RecID is null
    end

  -- sequentially read result records and mate with corresponding, mapped sheet values
  declare  TestSheetResult cursor for
    select Tab, CellRange, Operator, Operand, RecID
    from   #TestData
    where  CreateTestResultRec=1 or FactorCategory='TestHeaderData'

  open TestSheetResult
  fetch next from TestSheetResult into @Tab, @Range, @Operator, @Operand, @RecID
  while(@@fetch_status=0)
    begin
      -- retrieve, aggregate, and save current datum
      -- clear past data for each test datum cfg
      truncate table #SheetData
      if(isnull(@Tab,'')<>'' and isnull(@Range,'')<>'')
        begin
          -- tab and range specified
          -- query specified sheet tab and range
          -- identify beginning col and row of specified range
          if(charindex(':',@Range)>0)
            select @Cell=left(@Range,charindex(':',@Range)-1)
          else
            select @Cell=@Range
          if(substring(@Cell,2,1) between '0' and '9')
            select @row0=right(@Cell,len(@Cell)-1), @col0=lower(left(@Cell,1))
          else
            select @row0=right(@Cell,len(@Cell)-2), @col0=lower(left(@Cell,2))
          -- identify ending col and row of specified range
          if(charindex(':',@Range)>0)
```

```
      begin
        select @Cell=right(@Range,charindex(':',reverse(@Range))-1)
        if(substring(@Cell,2,1) between '0' and '9')
          select @row1=right(@Cell,len(@Cell)-1), @col1=lower(left(@Cell,1))
        else
          select @row1=right(@Cell,len(@Cell)-2), @col1=lower(left(@Cell,2))
      end
    else
      select @row1=@row0, @col1=@col0
    -- to facilitate aggregate operations, append individual cols from sheet (rows @row0 through @row1 from cols @col0
through @col1)
    -- to single col x of #SheetData
    -- done when last col in range appended
    select @done=0
    while(@done=0)
      begin
        select @sqltext='select *
                         from   openrowset(''Microsoft.ACE.OLEDB.12.0'',
                                           ''excel 12.0;database=' + @TestSheet + ';hdr=no;imex=1'',
                                           ''select * from [' + @Tab + '$' + @col0 + @row0 + ':' + @col0 + @row1 + ']'' )'
        --print @sqltext
        insert into #SheetData(x) exec(@sqltext)
        -- quit when last col is appended
        -- otherwise, advance to next col in range (A->Z->AA->ZZ)
        if(@col0=@col1)
          select @done=1
        else if(len(@col0)=1)
          if(@col0<'z')
            select @col0=char(ascii(@col0)+1)
          else
            select @col0='aa'
        else if(right(@col0,1)<'z')
          select @col0=left(@col0,1) + char(ascii(right(@col0,1))+1)
        else
          select @col0=char(ascii(left(@col0,1))+1)+'a'
      end
  end
else
  -- no tab or range specified - use data from cfg record (operand)
  insert into #SheetData(x) values(@Operand)
-- update results
-- note the requirement that numeric operation have numeric operands
-- non-enforcement of format rule causes factor to be ignored (no data are uploaded for it)
if(@Operator='*')
  -- require numeric multiplicand (assume configured operand is numeric, you configured it, after all!)
  if(not exists(select * from #SheetData where x is not null and isnumeric(x)=0))
    update #TestData set SheetValue=convert(varchar(20),convert(real,#SheetData.x)*convert(real,@Operand))
    from   #TestData cross join #SheetData
    where  #TestData.RecID=@RecID and #SheetData.x is not null
  else if(@EnforceRuleFormat=0)
    update #TestData set CreateTestResultRec=0 where RecID=@RecID
  else
    update #TestData set msg='Error:  Invalid numeric data format in cell range (' + FactorCategory + '/' + Factor + ')'
    where  RecID=@RecID
else if(@Operator='/')
```

```
  -- require numeric dividend (assume configured divisor is numeric and non-zero, who configured it?)
  if(not exists(select * from #SheetData where x is not null and isnumeric(x)=0))
    update #TestData set SheetValue=convert(varchar(20),convert(real,#SheetData.x)/convert(real,@Operand))
    from   #TestData cross join #SheetData
    where  #TestData.RecID=@RecID and #SheetData.x is not null
  else if(@EnforceRuleFormat=0)
    update #TestData set CreateTestResultRec=0 where RecID=@RecID
  else
    update #TestData set msg='Error:  Invalid numeric data format in cell range (' + FactorCategory + '/' + Factor + ')'
    where  RecID=@RecID
else if(@Operator='sum')
  -- require at least one non-null value and all non null values of numeric format
  -- note that the test (Excel) sheets employ iif() results for tabular data (blocks of cells)
  -- the iif() results, typically, are '' to indicate that no data exist for a cell
  if(exists(select * from #SheetData where isnull(x,'')<>'')
     and not exists(select * from #SheetData where isnumeric(x)=0 and isnull(x,'')<>''))
    update #TestData set SheetValue=convert(varchar(20),SheetData.x)
    from   #TestData cross join (select sum(convert(real,x)) as x from #SheetData where isnull(x,'')<>'') SheetData
    where  #TestData.RecID=@RecID
  else if(@EnforceRuleFormat=0)
    update #TestData set CreateTestResultRec=0 where RecID=@RecID
  else
    update #TestData set msg='Error:  Missing data or invalid numeric format in cell range (' + FactorCategory + '/' +
Factor + ')'
    where  RecID=@RecID
else if(@Operator='avg')
  -- require at least one non-null value and all non null values of numeric format
  -- note that the test (Excel) sheets employ iif() results for tabular data (blocks of cells)
  -- the iif() results, typically, are '' to indicate that no data exist for a cell
  if(exists(select * from #SheetData where isnull(x,'')<>'')
     and not exists(select * from #SheetData where isnumeric(x)=0 and isnull(x,'')<>''))
    update #TestData set SheetValue=convert(varchar(20),SheetData.x)
    from   #TestData cross join (select avg(convert(real,x)) as x from #SheetData where isnull(x,'')<>'') SheetData
    where  #TestData.RecID=@RecID
  else if(@EnforceRuleFormat=0)
    update #TestData set CreateTestResultRec=0 where RecID=@RecID
  else
    update #TestData set msg='Error:  Missing data or invalid numeric format in cell range (' + FactorCategory + '/' +
Factor + ')'
    where  RecID=@RecID
else if(@Operator='std')
  -- require at least 2 non-null values (sample std has n-1 in denominator) and all non-null values of numeric format
  if((select count(1) from #SheetData where isnull(x,'')<>'')>1
     and not exists(select * from #SheetData where isnumeric(x)=0 and isnull(x,'')<>''))
    update #TestData set SheetValue=convert(varchar(20),SheetData.x)
    from   #TestData cross join (select stdev(convert(real,x)) as x from #SheetData where isnull(x,'')<>'') SheetData
    where  #TestData.RecID=@RecID
  else if(@EnforceRuleFormat=0)
    update #TestData set CreateTestResultRec=0 where RecID=@RecID
  else
    update #TestData set msg='Error:  Invalid numeric data format in cell range (' + FactorCategory + '/' + Factor + ')'
    where  RecID=@RecID
else if(@Operator='concat')
  begin
    -- concat all values in cell range
```

```
                    select @a=''
                    declare catcurs cursor for select x from #SheetData
                    open catcurs
                    fetch next from catcurs into @b
                    while(@@fetch_status=0)
                      begin
                        if(len(@a)+len(@b)<255)
                          select @a=@a+@b
                        fetch next from catcurs into @b
                      end
                    update #TestData set SheetValue=@a where RecID=@RecID
                  end
                else
                  update #TestData set SheetValue=#SheetData.x
                  from   #TestData cross join #SheetData
                  where  #TestData.RecID=@RecID and #SheetData.x is not null
                fetch next from TestSheetResult into @Tab, @Range, @Operator, @Operand, @RecID
              end
            deallocate TestSheetResult

            -- test for missing values
            update #TestData set msg='Error:  Data missing in specified sheet location (' + FactorCategory + '/' + Factor + ')(' + Tab +
')(' + CellRange + ')'
            where  CreateTestResultRec=1 and isnull(msg,'') not like 'Error:%' and isnull(SheetValue,'')=''

            -- test for proper data format
            if(@EnforceRuleFormat=1)
              update #TestData set msg='Error:  Improper data format (' + FactorCategory + '/' + Factor + ')(' + isnull(SheetValue,'') +
')'
              where  CreateTestResultRec=1 and DataFormat='n' and isnumeric(SheetValue)=0 and isnull(msg,'') not like 'Error:%'
            else
              update #TestData set CreateTestResultRec=0
              where  CreateTestResultRec=1 and DataFormat='n' and isnumeric(SheetValue)=0 and isnull(msg,'') not like 'Error:%'

            -- test for missing reactor ID
            if(not exists(select * from #TestData where Factor='InstrumentID' and isnull(SheetValue,'')<>''))
              insert into #TestData(msg) values('Error:  No reactor ID in test sheet or test cfg')

            -- test for missing test start and end times
            if(not exists(select * from #TestData where Factor='TestStartTime' and isdate(SheetValue)=1)
               and not exists(select * from #TestData where Factor='TestEndTime' and isdate(SheetValue)=1))
              insert into #TestData(msg) values('Error:  Missing test start or end time in test sheet or test data map cfg')

            -- test applicable data integrity rules
            if(@EnforceRuleBoundaries=1 and not exists(select * from #TestData where isnull(msg,'') like 'error%'))
              update #TestData set msg=case when(DataRule.Operator='NumBetween')then
                                             'Error:  Response Value Outside Data Integrity Rule Limits.  '
                                           else
                                             'Error:  Unrecognized Factor Data Integrity Rule in ReactorDataManagementCodes.  '
                                      end + isnull(msg,'')
              from   #TestData join ReactorDataManagementCodes DataRule on #TestData.FactorCategory=DataRule.FactorCategory
                     and #TestData.Factor=DataRule.Factor
              where  #TestData.CreateTestResultRec=1
                     and DataRule.ClassA='Map' and DataRule.ClassB='FactorDataRule'
                     and DataRule.ID1 in(@TemplateID, '*') and DataRule.ID2 in(@TestID, '*')
```

```
                        and ( DataRule.Operator='NumBetween'
                              -- parse least and greatest (left and right numeric values separated by a comma) from rule operand
                              -- then compare to value read from test sheet
                              and convert(real,#TestData.SheetValue)
                                  not between convert(real,left(DataRule.Operand,charindex(',',DataRule.Operand)-1))
                                          and convert(real,right(DataRule.Operand,len(DataRule.Operand)-charindex(',',DataRule.Operand)))
                              or DataRule.Operator<>'NumBetween' )

                  -- create test header and result records
                  if(not exists(select * from #TestData where isnull(msg,'') like 'error%'))
                    begin try
                      begin tran
                      -- test header
                      insert into LabTest(SampleID, TestCfgID, MeasurementGroup, LRFID, InstrumentID, DataSource, TestStartTime, TestEndTime)
                      select @SampleID, @TestCfgID, @MeasurementGroup, LRFID, InstrumentID, @TestSheet + ' [' + @TemplateID + '(' + @TestID +
')]',
                             convert(smalldatetime,TestStartTime,101), convert(smalldatetime,TestEndTime,101)
                      from   (select top 1 SheetValue as InstrumentID from #TestData where Factor='InstrumentID') Instrument
                             cross join (select top 1 SheetValue as TestStartTime from #TestData where Factor='TestStartTime') TestStartTime
                             cross join (select top 1 SheetValue as TestEndTime from #TestData where Factor='TestEndTime') TestEndTime
                             left join (select top 1 SheetValue as LRFID from #TestData where Factor='LRFID') LRF on 1=1
                      -- retain TestID
                      select @RecID=scope_identity()
                      -- results
                      insert into LabResult(TestID, ResultCfgID, MeasurementGroup, ResponseValue)
                      select @RecID, ResultCfgID, 1, case when(DataFormat='n')then convert(real,SheetValue) else SheetValue end
                      from   #TestData
                      where  CreateTestResultRec=1
                      commit tran
                    end try
                    begin catch
                      rollback tran
                      insert into #TestData(msg) values('Error (A):  ' + left(error_message(),230))
                    end catch

                end

              end try

              begin catch

                insert into #TestData(msg) values('Error (B):  ' + left(error_message(),230))

              end catch

            else

              insert into #TestData(msg) values('Error:  Data Exist for Sample/Test Cfg/Measurement Group')

          end

        else

          insert into #TestData(msg)
          select 'Error:  Invalid Item/Sample/Test Cfg (' + convert(varchar(10),ItemMaster.ItemCfgID) + '/' +
```

```
              convert(varchar(10),SampleMaster.SampleCfgID) + '/' + convert(varchar(10),@TestCfgID) + ')'
        from   SampleMaster join ItemMaster on SampleMaster.ItemID=ItemMaster.ItemID
        where  SampleMaster.SampleID=@SampleID

    else

      insert into #TestData(msg) values('Error:  Unknown Sample ID')

    -- return results
    if(exists(select * from #TestData where msg like 'err%'))
      select @RetStatus='Error'
    else
      select @RetStatus='ok'
    if(@RetStyle='RowSet')
      select   FactorCategory, Factor, '[' + Tab + '][' + CellRange +']' as Cell, SheetValue, isnull(msg,'ok') as msg
      from     #TestData
      order by case when(msg like 'error%')then 1 else 2 end, FactorCategory, Factor, Tab, CellRange

  end

drop table #SheetData
drop table #TestData
drop table #SheetDescriptorData
GO
/****** Object:  StoredProcedure [dbo].[LabDataImportReactorReload]    Script Date: 04/21/2015 10:31:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE proc [dbo].[LabDataImportReactorReload] @LabTestID int as

declare @TestSheet varchar(255), @TemplateID varchar(50), @TestID varchar(50), @SampleID varchar(50)='', @TestCfgID int=0,
        @ExistAction varchar(20), @DataSource varchar(511), @MeasurementGroup smallint, @p1 smallint, @p2 smallint, @RetStatus varchar(25)

select @TestCfgID=TestCfgID, @SampleID=SampleID, @MeasurementGroup=MeasurementGroup, @DataSource=DataSource
from   LabTest
where  TestID=@LabTestID

select @TemplateID=ID1 from ReactorDataManagementCodes where ClassA='Map' and ClassB='TestID' and @DataSource like '%'+ID1+'%'

if(@TemplateID is not null)
  begin
    select @p1=charindex('(',@DataSource,charindex(@TemplateID,@DataSource)+1)+1,
           @p2=charindex(')',@DataSource,charindex(@TemplateID,@DataSource)+1)-1
    if(@p1<@p2)
      begin
        select @TestID=substring(@DataSource,@p1,@p2-@p1+1)
        if(exists(select * from ReactorDataManagementCodes where ID1=@TemplateID and ID2=@TestID))
          begin
            select @TestSheet=left(@DataSource,charindex(@TemplateID,@DataSource)-2)
            if(@TestSheet is not null)
              begin
                begin tran
                  delete LabResult where TestID=@LabTestID
                  delete LabTest where TestID=@LabTestID
```

```
                exec LabDataImportReactor @Operation='ImportTestData', @TestSheet=@TestSheet, @TemplateID=@TemplateID,
                                          @TestID=@TestID, @SampleID=@SampleID, @TestCfgID=@TestCfgID,
                                          @MeasurementGroup=@MeasurementGroup, @RetStyle='ErrVar', @RetStatus=@RetStatus output
                if(@RetStatus not like 'err%')
                  commit tran
                else
                  rollback tran
                select @RetStatus as RetStatus
              end
            else
              select 'Error:  Invalid Test Sheet (File Name)' as msg
          end
        else
          select 'Error:  Unrecognized or Missing Test ID in LabTest DataSource' as msg
      end
    else
      select 'Error:  Unrecognized or Missing Test ID in LabTest DataSource' as msg
  end
else
  select 'Error:  LabTest Record does not Exist or Unrecognized or Missing Template ID in LabTest DataSource' as msg
```