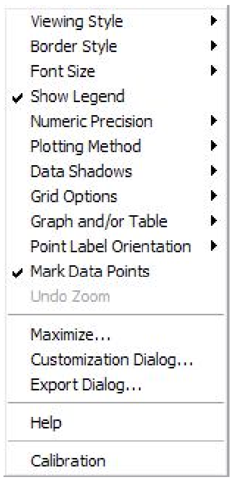
The result of each sample is represented by a 'red dot' on the funnel plot. The samples go in order by the time of testing (first, second, third, etc.) The optimum is to have all the samples fall within the funnel of upper and lower limits. The individual graphs can be customized by pointing the arrow to the graph to be edited, and right clicking on the mouse. This will bring up the following options menu:



From this menu one can change the look and feel of each individual graph. The 'Parameter' option is unique to this graph. This option enables the user to change the IDF default limits (upper and lower) of the graph. Right clicking on 'parameter' brings up the parameter screen, allowing the limits to be changed. See section 5.2. for more information on the available options in the options window.

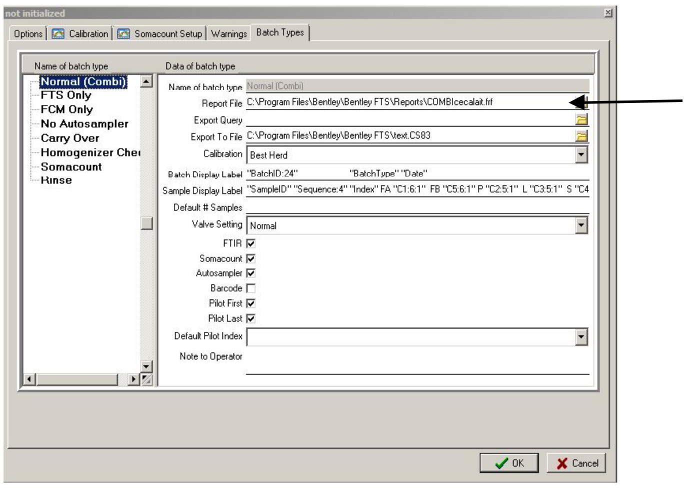
A stop sign STOP will appear in a tab if any of the values in that plot fall outside the limits (upper/lower) set.

3.6. Formatting Reports

The reporting system is based on a system called Fastreport. This provides the user with a great deal of possibilities in customized reports and reporting functions. The instrument has preformatted reports tied directly to specific Batch Types. Individual and customized reports can also be created by using the Fastreport software. To do this, go to “Tools”, and selecting “Report Designer” from the drop down menu.

3.6.1. Setup

The reports used for the batches are tied directly the specific batch types. To select a report to match a specific Batch Type, go to the '*Tools*' – '*Options*' – '*Batch Types*' tab. The report is selected at the line labeled '*Report File*'. This information can also be found in Section 3.3.1.


not initialized

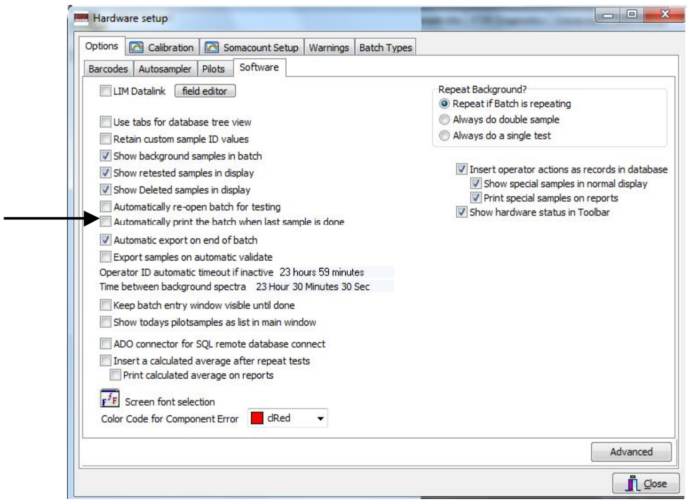
Options | Calibration | Somacount Setup | Warnings | Batch Types

Name of batch type	Data of batch type
Normal (Combi)	
FTS Only	
FCM Only	
No Autosampler	
Carry Over	
Homogenizer Che	
Somacount	
Rinse	

Name of batch type	Normal (Combi)
Report File	C:\Program Files\Bentley\Bentley FTS\Reports\COMBIcecalait.frf
Export Query	
Export To File	C:\Program Files\Bentley\Bentley FTS\text.CS83
Calibration	Best Herd
Batch Display Label	"BatchID:24" "BatchType" "Date"
Sample Display Label	"SampleID" "Sequence:4" "Index" FA "C1:6:1" FB "C5:6:1" P "C2:5:1" L "C3:5:1" S "C4
Default # Samples	
Valve Setting	Normal
FTIR	✓
Somacount	✓
Autosampler	✓
Barcode	☐
Pilot First	✓
Pilot Last	✓
Default Pilot Index	
Note to Operator	

The NexGen software, uses a customizable reporting tool which reads '*.FR3' file types and then generates the report. In the above example, the file 'mdref.FR3' was selected as the default report file for the Batch Type 'Routine Herd'.

The printer for the instrument is setup under '*Tools*' – '*Options*' – '*Options*' – '*Software*'. The printer can be set to automatically print out a report when a batch has completed its testing (enable the '*Automatically print the batch when last sample is done*').


Hardware setup

Options | Calibration | Somacount Setup | Warnings | Batch Types

Barcodes | Autosampler | Pilots | Software

- ☐ LIM Datalink [field editor]
- ☐ Use tabs for database tree view
- ☐ Retain custom sample ID values
- ✓ Show background samples in batch
- ✓ Show retested samples in display
- ✓ Show Deleted samples in display
- ☐ Automatically re-open batch for testing
- ☐ Automatically print the batch when last sample is done
- ✓ Automatic export on end of batch
- ☐ Export samples on automatic validate

Operator ID automatic timeout if inactive 23 hours 59 minutes
Time between background spectra 23 Hour 30 Minutes 30 Sec

- ☐ Keep batch entry window visible until done
- ☐ Show todays pilotsamples as list in main window
- ☐ ADO connector for SQL remote database connect
- ☐ Insert a calculated average after repeat tests
  - ☐ Print calculated average on reports

Screen font selection
Color Code for Component Error 🟥 clRed

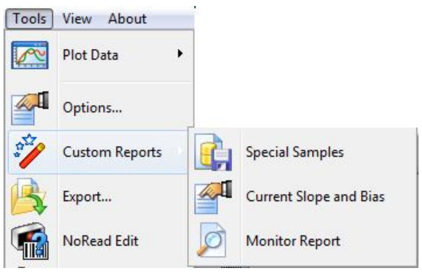
Repeat Background?
- ● Repeat if Batch is repeating
- ○ Always do double sample
- ○ Always do a single test

- ✓ Insert operator actions as records in database
  - ✓ Show special samples in normal display
  - ✓ Print special samples on reports
- ✓ Show hardware status in Toolbar

If the '*Automatically print the batch when last sample is done*' option is not selected, the user will have to manually select a batch and press the print button.

3.6.2. Custom Reports

A variety of custom reports can be created and saved to the Custom Report option under '*Tools*'. The use of custom reports provides a means to set up queries for the creation of reports that are used on a frequent basis.



The Custom Reports option provides three specific functions depending on the type of report required.

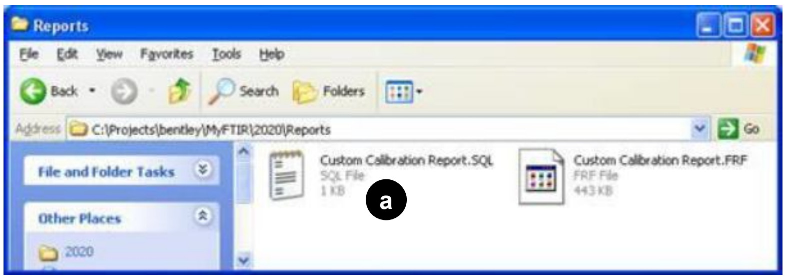
▨	Opens the 'File Save' window from where the results of the query file can be saved. Verification prompt will appear when saving a file.
▨	Opens the SQL query result screen.
▨	If a report file has been created, the report will open when selecting this option. If no .INI file exists, the query screen will open.

In addition, report files may be automatically saved, and the name of the file will appear in the pull down menu for later use. The files are created using the standard query manager and saved with the SQL or FR3 extension.

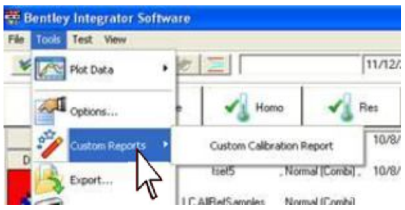
3.6.2.1. Creating a custom report

One way to create a custom report is to save the desired report or query into the Reports subdirectory with the SQL extension. It is strongly recommended to create the export queries as SQL files rather than .qry files as this speed up the database access.

When a file such as *Custom Calibration Report.SQL* (a) is saved into the Reports subdirectory, the program will register the action and create the name for the menu item.


Reports
File Edit View Favorites Tools Help
Back Search Folders
Address C:\Projects\bentley\MyFTIR\2020\Reports
Go
File and Folder Tasks
Other Places
2020
Custom Calibration Report.SQL
SQL File
1 KB
a
Custom Calibration Report.FRF
FRF File
443 KB

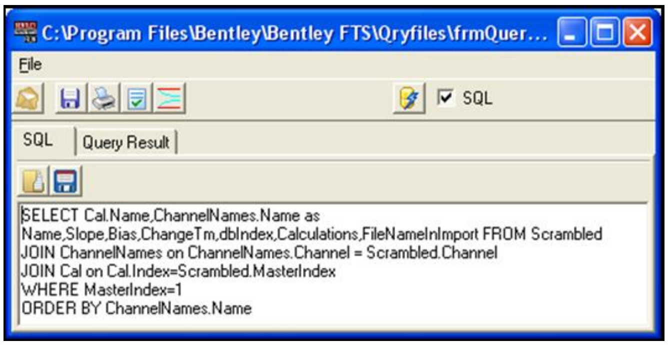
Once this is done, the name of the file will appear in the pull down menu when clicking on *Custom Reports*.



The "Custom Calibration Report.SQL" file from the above example contains the following SQL statements:

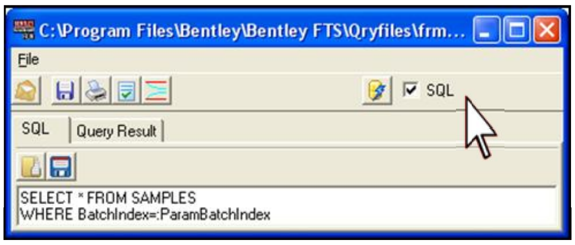
```
SELECT Cal.Name,ChannelNames.Name as
Name,Slope,Bias,ChangeTm,dbIndex,Calculations,FileNameInImport
FROM Scrambled
JOIN ChannelNames on ChannelNames.Channel = Scrambled.Channel
JOIN Cal on Cal.Index=Scrambled.MasterIndex
WHERE MasterIndex=1
ORDER BY ChannelNames.Name
```

The file in this example is used to create data for evaluating calibration changes. When selected, the program will look for a file with the extension FR3 that has the same name as the SQL file. This is a FastReport report definition file. If the FR3 file exists the program will display the formatted report. If the FR3 file does not exist the program will display the query manager with the SQL statements already loaded.



```
SELECT Cal.Name,ChannelNames.Name as
Name,Slope,Bias,ChangeTm,dbIndex,Calculations,FileNameInImport FROM Scrambled
JOIN ChannelNames on ChannelNames.Channel = Scrambled.Channel
JOIN Cal on Cal.Index=Scrambled.MasterIndex
WHERE MasterIndex=1
ORDER BY ChannelNames.Name
```

To design new SQL and FR3 files for the Custom Report option go to the standard query manager. Check the SQL button and the system will default to the SQL screen.



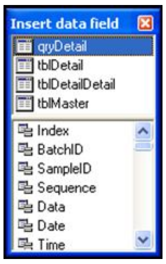
```
SELECT * FROM SAMPLES
WHERE BatchIndex=:ParamBatchIndex
```

From here a new query can be created and saved, and new report files designed as needed. The custom query language includes the following variables:

:*ParamBatchIndex* : index into batch currently selected in the display of results
:*ParamSampleIndex* : index to the sample currently selected in the display
:*ParamDate* : the system DATE
:*ParamTime* : the system Time
:*ParamSerialNumber* : Serial number of the instrument
:*ParamOperator* : name of the operator

These variables might be utilized in a query to extract the desired information from the database. See Section *6.5.2. SQL* for more information on the Structured Query Language.

When designing a report using the custom SQL file it is necessary to place the fields in the detail band from the qryDetail, not the tblDetail. If this is not done the report will consist of the first sample in the tblDetail being printed multiple times, resulting in each line on the report being identical.



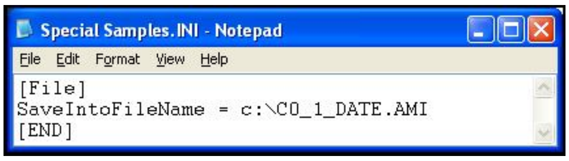
The tblDetailDetail will be inactive when using the SQL system. If the data from the SomacountData table is needed, the SQL will have to include a LEFT OUTER JOIN to this table.

3.6.2.2. Automatically Save a Custom Report

A custom report can be set up to automatically save a file onto the Custom Reports pull down menu. There are several uses for this function depending on the results desired. Following are some examples of the types of custom reports that can be set up using the .INI files and SQL queries.

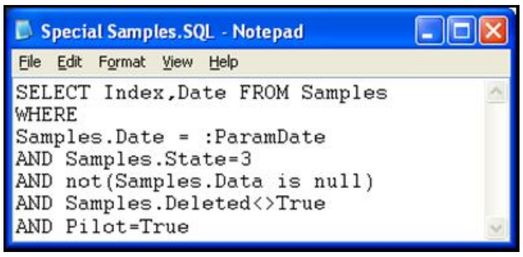
Example 1:

In the reports directory create a file named “Special Samples.INI”. The existence of the file prompts the software to open the file and look for a filename. The file may contain the following command which instructs the software to save the result of the query into the file *C0_1_DATE.AMI*. If an .INI file has not been created the query screen will appear.



```
[File]
SaveIntoFileName = c:\C0_1_DATE.AMI
[END]
```

To set up a query that will extract all the pilot samples for a 'today's date' create the following SQL file and save it into the reports directory.

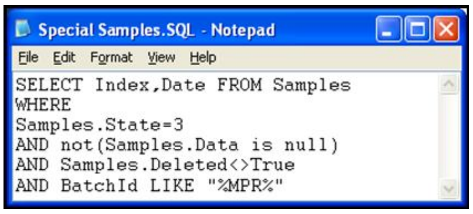


```
SELECT Index,Date FROM Samples
WHERE
Samples.Date = :ParamDate
AND Samples.State=3
AND not(Samples.Data is null)
AND Samples.Deleted<>True
AND Pilot=True
```

When this custom report is selected in the programs menu structure, the result will be an AMI file named c:\C0_1_20101509.AMI containing all the pilot samples from today.

Example 2:

To set up a query that will extract all samples results from batches containing the letters MPR create the following SQL file and save it into the reports directory.

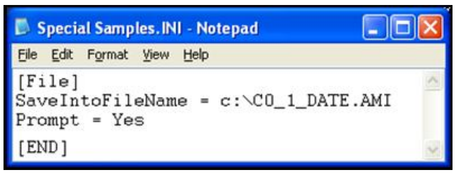


```
SELECT Index,Date FROM Samples
WHERE
Samples.State=3
AND not(Samples.Data is null)
AND Samples.Deleted<>True
AND BatchId LIKE "%MPR%"
```

When selecting the *Special Samples* report from the *Custom Reports* pull down menu, the result will be an AMI file named c:\C0_1_20101509.AMI containing all samples from any batch with a name that includes the letters MPR.

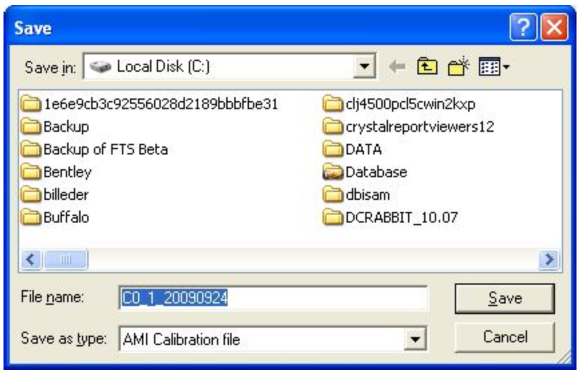
Example 3:

If the following .INI file is created, the system will prompt the operator for the filename before executing the save function.



```
[File]
SaveIntoFileName = c:\C0_1_DATE.AMI
Prompt = Yes
[END]
```

This defaults to the SaveIntoFileName location and name.



File name: C0_1_20090924

Save as type: AMI Calibration file

3.7. Pascal Script

3.7.1. PascalScript file formats

The system supports FastScript Pascal scripting and has an icon in the area of saving data into custom files. The use of PascalScript file formats allows third party programming of custom file formats.

To use this function create a text file with the name myExportFunction.PAS. The extension PAS signifies the use of the Pascal style scripting. An example of the content of this file could be:

```
var
 TS : TStringList;
 st, batch: String;
 n : integer;
begin
 With cdsSource.Dataset do
 Begin

  first;
  batch := FieldByName('batch').AsString;
  // send a debug message to the log system
  CodeSiteSend('SaveToFileName='+batch);

  TS := TStringList.Create;
  try
   // place code here to initialize any header in the file
   TS.add(format('Batch,%s,',[FieldByName('batch').AsString]));
   TS.Add(format('Batch Date,%s,',[FieldByName('date').AsString]));
   TS.add(format('Total,%s,',[FieldByName('Total').AsString]));
   TS.Add(format('Lab Date,%s,',[FieldByName('LabDate').AsString]));
   TS.Add(format('Lab1,%s,',[FieldByName('Lab1').AsString]));
   TS.Add(format('Lab2,%s,',[FieldByName('Lab2').AsString]));
```

```
    TS.Add(format('Ext1,%s,',[FieldByName('Ext1').AsString]));
    TS.Add(format('Ext2,%s,',[FieldByName('Ext2').AsString]));
    TS.Add(format('Ext2,%s,',[FieldByName('Ext3').AsString]));
    TS.Add(format('BatchType,%s,',[FieldByName('BatchType').AsString]));
    TS.Add(format('Program,%s,',[FieldByName('Program').AsString]));
    TS.Add('');

TS.Add('#inWS,#inJob,#Sub,FatA(T),Prot(T),ID,Vol.Avail,ResultType,BottleType');
    TS.Add('');
    n := 0;
    while not(eof) do
    Begin
      n := n+1;
      ProgressBar(round(100*n/RecordCount));

      // place code here to create each line in the file with data
      st := '';
      st := st + format(' %d',[fieldbyname('index').AsInteger]);
      st := st + format(',%d',[fieldbyname('Sequence').AsInteger]);
      st := st + format(',%d',[fieldbyname('SubSequence').AsInteger]);
      st := st + format(',%.2f',[fieldbyname('C1').AsFloat]);
      st := st + format(',%.2f',[fieldbyname('C2').AsFloat]);
      st := st + format(',%s',[fieldbyname('SampleID').AsString]);
      st := st + format(',%s',['20.00']);
      st := st + format(',%s',[fieldbyname('ResultType').AsString]);
      st := st + format(',%s',[fieldbyname('BottleType').AsString]);
      st := st+',';
      TS.add(st);
      Next;
    End;
  finally
    TS.saveToFile('c:\'+batch+'.csv');
    TS.free;
  end;
 End;
 // send a debug message to the log system
 CodeSiteSend('close thread');
end.
```

The above script will generate a simple text file with the SampleID, C1 and C2 on each line followed by the entire spectra recorded for that sample. The example is a CSV compatible format with a comma (,) separating each field. The text following the // is a comment and will not be processed by the script. The scripting is almost unlimited in scope and flexibility. The end-user has access to the data from the database directly and can generate text and binary files per custom requirements.

The following special functions are used to access the raw data from the samples database.

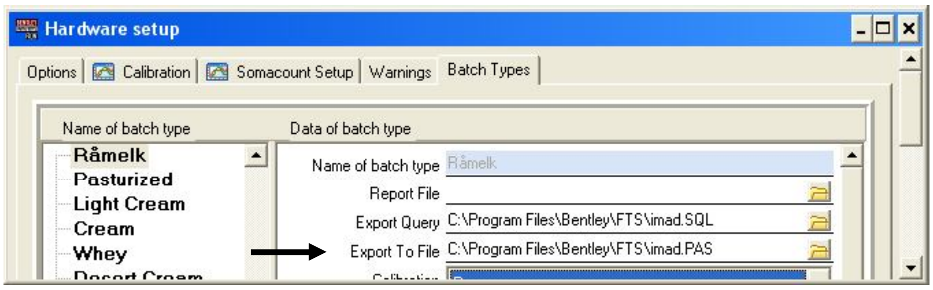
Function	
Spectra(i):single	Data point number ***i*** in the infrared spectra
DataPointsInSpectra:Integer	The number of data points stored in a given spectra
Spectra_Begin:Single	The wavelength of the first data point in the spectra array
Spectra_End:Single	The wavelength of the last data point in the spectra array
Imp_rFFT(i):single	Impedance data point ***i*** from the real of the FFT
Imp_iFFT(i):single	Impedance data point ***i*** from the imaginary of the FFT
Imp_pFFT(i):single	Impedance data point ***i*** from the power of the FFT
Imp_frequency(i):single	Impedance excitation frequency for data point ***i***

Imp_Pts:Integer	Number of data point in the impedance data array
Scc_Intensity(i):integer	Somacount intensity (or height) distribution data point
Scc_Duration(i):integer	Somacount duration (or width) distribution data point
Scc_Pts:Integer	Somacount number of data point in each histogram

The Pascal scripting system comes with its own help file, FS.HLP, which can be obtained at the Bentley Instruments FTP server. The PascalScript is executed within the context of its own thread so the execution time of the Script will not affect the running of the instrument.

3.7.2. Using PascalScript

When using PascalScript, the batch type used should be set up with a .PAS file as the export file.



Name of batch type: Råmelk
Report File:
Export Query: C:\Program Files\Bentley\FTS\imad.SQL
Export To File: C:\Program Files\Bentley\FTS\imad.PAS

The system will then load the SQL from the “Export Query” and the “PascalScript” from the “Export to File”. Note that in the above example the SQL and PAS files have the same name, but this is not a requirement.

When a batch name has been entered in the batch identification window the system will search for a PascalScript file (**onBatchName**.Pas). If the file exists it will be executed. This is an example of what the file '*onBatchName.Pas'* might contain:

```
Begin
 if uppercase(BatchName.Text)=uppercase('1') then
 Begin
  BatchName.Text := 'macro1';
  lab1.text := '';
  lab1.items.clear;
  lab1.items.add('choice 1 in drop down');
  lab1.items.add('choice 2 in drop down');
  lab2.text := 'Text in Lab2';
  ext1.text := 'Text in ext1';
  ext2.text := 'Text in ext2';
  ext3.text := 'Text in ext3';
  nRepeat := 2;                 // each sample in this batch is tested twice
  nSamplesInBatch := 10;        // there are 10 samples in this batch
  AutoPrint := false;   // do not automatically print this batch when   closed.
  PilotFirst := false;    // no pilot in the first position of this batch
  PilotLast := false;           // no pilot in the last position of this batch
  StartInNextRack := false;     // rackposition 1 is sequence 1
  useAutosampler := true;       // use the autosampler to advance the rack
  useBarcode := false;          // do not use the barcode reader
  useFTIR := true;              // test samples using the component FTIR
  useSomacount := true;         // test samples using the Somacount
 End;
End.
```

When the entry of a batch has been completed and “save’ or ‘done’ pressed, the system will look for a PascalScript file named ‘**onBatchCreate**.pas’. This script can be used to verify that the user has entered correct data. The following is an example of a '*onBatchCreate.pas*’ file.

```
Begin
OperatorMessage := ‘’;
if uppercase(BatchName.Text)='GARBAGE' then
Begin
BatchName.Text := ‘pg13 please’;
OperatorMessage := 'Bad language does not work for me'; // message to the operator
End else
If uppercase(BatchName.Text)=’MACRO1’ then
Begin
If (nSamplesInBatch<10) then OperatorMessage := ‘must have more than 10 samples’;
If (nRepeat>1) then OperatorMessage := ‘no repeating allowed’;
If (Lab1.Text=’’) then OperatorMessage := ‘Make Selection in Lab1’;
End;
End.
                    CanClose := false; // prevents the creation of the batch
```

Setting the ***OperatorMessage*** will prevent the batch from being created and the user is returned to the batch entry form after being presented with the indicated message.

Once a result has been analyzed by the instruments, but before the data is output on serial ports, hard drives or printers, the system will call for the script file **onSampleDone**.pas which allows the programmer to interface with the database directly and provide required custom input. For example:

```
var
  idx : integer;
  S : TStringList;
Begin
  idx := cdsSource.Dataset.fieldbyname('index').AsInteger;
  S := TStringList.Create;
  S.Add('UPDATE Samples SET C1=9.99');
  S.Add('WHERE [index]='+IntToStr(idx));
  sqlExecute(S.text);
  ResultMessage := '';
  S.free;
End.
```

The use of the function sqlExecute takes an SQL and executes it on the server.

NOTE: there is no limit to what can be executed in this statement and the only filter is the server itself. The programmer is fully responsible for what this piece of code will do. Any damage done to the database is outside the scope of support from Bentley Instruments.

If the available customizations for serial port output prove insufficient to solve a particular case, the system will look for a script named '**onBeforeSerialPort**.pas' which can be used to overwrite the output format. The following is an example of the '*onBeforeSerialPort.pas'* script.

```
var
  idx,i : integer;
  sub,nm : String;
Begin
  idx := cdsSource.Dataset.fieldbyname('index').AsInteger;
  sub := format('%d',[idx]);
  for i := 1 to 30 do
  Begin
    nm := format('C%d',[i]);
   sub := sub + ',' + cdsSource.Dataset.fieldbyname(nm).AsString;
  End;
  ResultMessage := sub;
End.
```

All results returned in the "ResultMessage" variable will be sent to the serial port.

APIPascalScript.exe debug
APIPascalScript.exe is a special program that has been created to debug a custom solution consisting of an SQL statement and the PAS script file used in the export.

The filename shown in the above example is a fixed name used for a debug in the final version of the script where it is often needed to name the exported files with the name of the batch. Change the code to include the following:

```
With DataSource.DataSet do
    Begin
      First;
      Fs := TfileStream.Create('c:\'+FieldByName('BatchID').AsString+'.TXT',fmCreate);
    End;
```

For the SQL, the following example shows how to get the batch information to synchronize with the output such that the SQL returns only the data from the current batch

```
SELECT * FROM Samples
Where BatchIndex=:paramBatchIndex
```

Note that the API development program for the Pascal-Script does not support this parameter.

3.7.3. FastReport Template Updates

In order to be compatible with scripts, the reporting engine has new features such as variable types, classes, functions, common language constructions, which provide significant improvements in performance. These updates do require checking of existing script code and changes to be made in order to ensure proper functionality.

3.7.3.1 Script

The following are updates that should be considered when writing script in FR2 and FR4 or converting from one version to another. Examples are included for clarification.

- In FR4 variables must be declared (Pascal style) while this is not supported in FR2 (Basic style).

FR2 code	**FR4 code**
a := 1;	var a: Integer;
	a := 1;

- In FR4 < > must be used to surround report variables, data fields and system variables. In FR2 [ ] must be used for the same purpose as in FR4, but can be omitted if the variable has a "correct Pascal identifier" name.

FR2 code	**FR4 code**
a := [dataset."Field"];	a := <dataset."Field">;
b := MyReportVariable;	b := <MyReportVariable>;

- In FR2 [ ] may be used even if it is not necessary. In FR4 the use of < > is restricted to data fields, report variables, and system variables only.

FR2 code	**FR4 code**
a := [Copy('a', 1, 1)];	a := Copy('a', 1, 1);

- In FR4 Pascal-style arrays can be used in addition to TfrxArray. FR2 uses tricky arrays without predefined bounds.

FR2 code

```
ar[0] := 1;
ar[10] := 2;
ar['test'] := 3;
ar[i] := 5;
```

FR4 code

```
var ar: TfrxArray;
ar := TfrxArray.Create;
ar[0] := 1;
ar[10] := 2;
ar['test'] := 3;
ar[i] := 5;
```

- FinalPass, FreeSpace, CurY and other engine variables must be changed to Engine.FinalPass, Engine.FreeSpace, Engine.CurY. This is partially solved by the converter which adds "with Engine do" to each code block.

3.7.3.2. Report Layout

Text and RichText objects.

- In FR2 [ ] can be used to define an expression in the text and inside the expression. In FR4 [ ] are used to define an expression and < > used inside the expression:

FR2 TfrMemoView text:	**FR4 TfrxMemoView text:**
Total: [Sum([table1."Field1"])]	Total: [Sum(<table1."Field1">)]

- It is important to check all text for inconsistent use of the [ ]:

FR2 TfrMemoView text:	**FR4 TfrxMemoView text:**
[[Copy(s, 1, 1)]]	[Copy(s, 1, 1)]
[Copy([s], 1, 1)]	either [Copy(s, 1, 1)]
	or [Copy(<s>, 1, 1)] if "s" is a report variable

- Check Condition property in the Group Header:

FR2 group condition:	**FR4 group condition:**
Copy(MyReportVariable, 1, 1)	Copy(<MyReportVariable>, 1, 1)
[Copy([MyReportVariable], 1, 1)]	Copy(<MyReportVariable>, 1, 1)

- Redesign cross-tab layout due to significant changes in FR4