Ostendo®

Scripting Reference Guide

4th February 2022

Table of Contents

1 Introduction

This reference help is intended to provide a guide as to what scripting functions and procedures are available in Ostendo. Accompanying each function is a simple script that you can copy and paste into the appropriate scripting area and then run the function to see how it works and what results are obtained.

At the end of this Help are some practical examples of useful scripts that you can use and/or adapt

2 Scripting Use in Ostendo

This section shows you where scripts can be run along with examples to demonstrate each option

- **Custom Menu Script**: A Custom Menu Script can be run in the following ways:
    - **Ostendo's Main Menu**: Adds the Script Name to the drop-down under ‘Custom’ on Ostendo’s main menu
    - **Command Line Script**: This allows you to run a script from the Command Line of your computer. This is very useful if, for example, you wish to run the script as part of your computer’s overnight batch process.
    - **Desktop Icon Script**: This allows you to run a script from a Desktop Icon without the necessity of starting up Ostendo.
- **Related Menu Script**: Adds the Script Name to the drop-down Menu held against the ‘**Related**’ button within specific screens.
- **Order Script**: This allows you to create a Script specific to a preselected Order
- **Screen Data Script**: Related to Master, Order, Receiving, and Invoicing Screens where the action of Adding or Deleting a record or changing any field within the record will automatically run this Script
- **Workflow Script**: These are scripts that can be used to change the appearance and/or Text of a Workflow Object.
- **Report Layout Editor**: Scripts to control many printing options such as printing fields under certain data conditions, etc

2.1 Custom Menu Script

This Type of script can be used in many areas of Ostendo such as:

- Main Menu Script
- Desktop Icon Script
- Command Line Script

2.1.1 Main Menu Script

Adds the Script Name to the drop-down under ‘Custom’ on Ostendo’s main menu. Selecting this will run the Script. This style of Script can be:

- Available to all
- Restricted to Administrator
- Restricted to specific Users

Examples of the types of script are:

- Updating your existing system with information from Ostendo
- Updating Ostendo with information from your existing system
- Importing Supplier Catalogues
- Send user-defined KPI information via email, or to a Mobile phone

To see how this works you can try the following exercise in which we will get an existing Job that was inadvertently created as an Order and we want to change the status to '**Quote**'.

Go into ***File>Custom Scripts*** and add a new Custom Script called (say) **Job Quote**. 'Check' the '**All Users**' radio button to make this script available to all users. In the 'Script' tab enter the following:

```
var
 TheJobNumber: String;
 TheCurrentStatus: String;
 AreYouSure: String;
begin
 TheJobNumber := AskMandatoryQuestionWithLookup('Job Number','Select Job Number',
1075);
 TheCurrentStatus := GetStringFromTable('JOBHEADER','ORDERSTATUS',
'ORDERNUMBER',TheJobNumber);
 AreYouSure := AskQuestion('Confirm this is the Job','TEXT','Are you sure you wish to
change ' + TheJobNumber + ' to a Quote','[d]Yes,No');
 if AreYouSure = 'Yes' then
   begin
    executeSQL('update JobHeader set OrderStatus = ''Quote'' where OrderNumber = ''' +
TheJobNumber + '''');
    executeSQL('delete from OrderRequirements where DemandOrderType = ''Job'' and
DemandOrderNumber = ''' + TheJobNumber + '''');
   end;
end.
```

Save and exit the Custom Script screen

You will see a new entry '**Custom**' on the top toolbar of Ostendo. If you click on this then your Script '**Job Quote**' will be displayed. If you click on this you will be taken through the script after which the Job Status will be updated to '**Quote**'

2.1.2 Desktop Icon Script

This allows you to run a script from a Desktop Icon without the necessity of starting up Ostendo.

To see how this works you can try the following exercise in which we will determine the number of Sales Orders created today.

Go into ***Sales>Sales Orders*** and create a couple of Orders

Go into ***File>Custom Scripts*** and add a new Custom Script called (say) **SalesToday**.

In the '**Script**' tab enter the following:

```
// Define the Variable
Var
 OrdersToday: String;
// You can then get the number of Sales Orders created today
Begin
   OrdersToday := GetSQLResult('Select Count(*) from SalesHeader where OrderDate =
''now''');
   Showmessage('There are ' + OrdersToday + ' Orders Today');
End.
```

To run the script from a Desktop Icon carry out the following. On your Desktop 'Right Mouse' on your existing Ostendo Icon that points to this '**DEMO**' Company database and then copy and paste to create another Desktop Icon. On the copied Icon 'Right Mouse' and select '**Properties**'. In the '**Target**' field extend the target reference to the following

```
"C:\Program Files\Ostendo\ostendo.exe" STARTUPID:DEMO SCRIPT=SalesToday
```

and save the changes

If you now double-click on the Icon it will run the script

2.1.3 Command Line Script

This allows you to run a script from the Command Line of your computer. This is very useful if, for example, you wish to run the script as part of your computer's overnight batch process.

To see how this works you can try the following exercise in which we will determine the number of Sales Orders created today. We will then run this script from the Command Line on your PC

Go into ***Sales>Sales Orders*** and create a couple of Orders

Go into ***File>Custom Scripts*** and add a new Custom Script called (say) **SalesToday**.

In the '**Script**' tab enter the following:

```
// Define the Variable
Var
 OrdersToday: String;
// You can then get the number of Sales Orders created today
Begin
  OrdersToday := GetSQLResult('Select Count(*) from SalesHeader where OrderDate =
"now"');
  Showmessage('There are ' + OrdersToday + ' Orders Today');
End.
```

Open Notepad on your PC then copy and paste the following

```
"C:\Program Files\Ostendo\ostendo.exe" STARTUPID:DEMO SCRIPT=SalesToday
```

where "C:\Program Files\Ostendo\ostendo.exe" points to the Ostendo executable
STARTUPID:DEMO defines the Database to look at
SCRIPT=SalesToday defines the script name within the Database

'**Save**' this as a **.bat** file (for example **SalesToday.bat**) directly under '**C**' drive

To run the .bat file from a Command Line click on '**Start**' on your Windows screen and select '**Run**'. On the presented panel enter '**CMD**' and click the '**OK**' Button. Go back to the Root Directory (Hint: Enter **cd ..** to go back one level.). You should end up with **C:\>**

Type in the .bat file name (Example: **SalesToday**) to give **C:\>SalesToday** and hit the '**Enter**' key on your keyboard to run your script

2.2 System Action

This allows you to create a Script that is run prior to displaying a specified screen. This could, for example, enable you to define what the User can or cannot view in the nominated screen

To see how this works you can try the following exercise in which we will substitute the Inquiry - Items screen in place of the Item Master screen. This will allow the User to View Item Data but not maintain it

Go into ***File>Custom Scripts*** and add a new System Action Script called (say) **ItemView**. From the drop-down list against field **System Action** select **Inventory/Items**

In the '**Script**' tab enter the following:

```
begin
 If CurrentUser = 'ADMIN' then
 OstendoInquiry('Inquiry - Items');
 Abort ;
end.
```

- The first line defines the user to which this action applies
- The second line states that the Inquiry screen 'Inquiry - Items' should be run
- The third line aborts the request to display of the Item Master screen.

It is important to always include the ABORT statement in the System Action script so that the usual action is not executed.
If this statement is not included, then the normal action will still run AFTER the System Action script is executed.

Now go into ***Inventory>Items*** and you will see that, because you are currently signed on as ADMIN, you will be directed to the Inquiry - Items screen

2.3 Custom Screens

There are two types of Custom Screens that you can create

1. Custom Data Screens Enable the User to create Custom Screens that provide the following options

Display Types: The Screen itself can be Inquiry only, Data Entry only or a combined Inquiry/Data Entry
Data Entry Styles: You have the option to use a Computer Keyboard, Graphical Keyboard, Touch Screen, or Barcode Scanner. All four can be utilised in the same Custom Data Screen.
Data Options: This can include Open format entry, validated against data within the Custom Data Script, or validated against Ostendo data.
Data Storage Options. The data being entered can be held in temporary storage for 'Batch Posting' into Ostendo or posted immediately - a single record at a time. If the data is held in temporary storage it can be held in non-Ostendo files and subsequently recalled into the data screen.

This style of screen requires the use of Ostendo's Graphical generator.

2. Data Entry Screens where you can define what data is to be collected and create a multi-line Data entry form into which the data is entered. The resultant records can be actioned as required

You can get more information on creating these types of screen in the Training Guide found under

Help on Ostendo's top toolbar

2.4 Related Menu Script

Adds the Script Name to the drop-down Menu held against the '**Related**' button within specific screens. The Related script can optionally take key information from the current screen and show specific - related - information. In the following example we will add a Related Script called '**Job Customer**' and whenever you are in a Job Screen this script will take you to the Customer Detail screen related to this Order's Customer

Go into ***File>Custom Scripts*** and create a new Script called '**Job Customer**'. In the '**Detail**' tab 'check' the '**Add to this Screen**' checkbox and select '**Job Orders**' from the drop-down list in the adjacent field. In the '**Script**' tab add the following script

```
begin
  RunSystemAction('Sales', 'Customers');
  SetScreenParameter('KEYFIELD=CUSTOMER');
  SetScreenParameter('KEYVALUE=' + GetSourceFieldValue('CUSTOMER'));
  SetScreenParameter('TABINDEX=1');  // This will open up in the Customer 'Detail'
screen rather than the 'List' screen
end.
```

Save and exit the Custom Script screen.

Go into ***Jobs>Job Orders*** and click on the '**Related**' button. You will find that the '**Job Customer**' Custom Script appears in the drop-down list. If you select this then the script will bring up the Customer Screen relating to this Job Order.

2.5 Order Script

This allows you to create a Script that is run against an Order as a whole in the following areas:

- Assembly
- Jobs
- Sales
- Purchase
- POS

and enables you to add extra specific functionality such as:

- Total Order Value discounting based on Order content
- Freight calculations based on Order content
- Order Authorisation Levels (Example: User Purchasing levels)
- Order Margin Control with User-defined Margin levels
- Order Validation and/or Checks
- Workflow actions (Example: send Email regarding this Order)
- Promotions (Example: 3 for price of 2, etc)

To see how this works you can try the following exercise in which we will create a Sales Order Script that will return the Order Customer's Name which when confirmed will allow the order to proceed.

Go into ***File>Custom Scripts*** and add a new Custom Script called (say) **OrderCust**. 'Check' the '**This is an Order Script**' checkbox. To define if this script must be acknowledged before the Order can proceed you should also 'check' the '**Mandatory**' checkbox.

In the '**Script**' tab enter the following:

```
var
TheCustomer: string;
begin
   TheCustomer := GetSourceFieldValue('CUSTOMER');
   showmessage(TheCustomer);
   OrderScriptRun(True);
end.
```

- The first line defines a variable
- The 3rd line populates this variable with the contents of the CUSTOMER field in the SALESHEADER record
- The 4th line displays the content of the variable when the script is run
- The 5th line updates the **OrderScriptRun** field in the SALESHEADER record with '**True**' to denote that the Order Script has been run

The next step is to tell Ostendo that the script is linked to a Sales Order. To do this go into ***File>System Configuration>Order Scripts*** and create a new record containing the following
Screen: Select '**Sales Orders**' from the drop-down
Script Name: Select **OrderCust**

Now go into ***Sales>Sales Orders*** and create a Sales Order then add a line to the Order. If you try and pick a line then you will be presented with an error message stating that **OrderCust needs to be run'**. I.e. **OrderScriptRun** field in the SALESHEADER record is currently set to '**False**'.

You will see a new button (**OrderCust**) on the **Batch Entry Bar** of the Sales Order Lines screen. If you click on this button then the script will be run. This will return the Customer Name to the screen in addition to amending the **OrderScriptRun** field in the SALESHEADER record to '**True**'.

You can now continue with picking the Sales Order lines

2.6 Screen Data Script

This is related to Master, Order, Receiving, and Invoicing Screens where the action of Adding or Deleting a record or changing any field within the record will automatically run the Script to provide a resultant action. For example

- Zero Price Check on Sales Order Lines
- Update Sell Price based on Last receipt Cost
- In Purchasing check for best price from all Suppliers
- Have a pop-up Sales Message appear
- Specify a minimum order quantity
- Show active Promotion when Sales Line Entered

To see how this works we will create a Custom Script that will block any Price or Cost change to an Order Line if the resultant margin falls below the value defined in the System Settings screen.

Go into ***File>System Configuration>System Settings*** and amend field '**Min Allowable Margin%**' to **50**.

Now, go to ***File>Custom Scripts*** and add a new Custom Script called (say) **Margin Check** and 'check' the '**This is a Screen Data script**' checkbox. Click on the 'Script' tab and add the following script

```
var
 TheMinMargin,TheUnitPrice,TheUnitCost,TheCalcMargin: double;
 TheIntValueofMargin: integer;
begin
  TheMinMargin := GetDoubleFromTable(
'SYSTEMMASTER','MINMARGINPERCENT','COMPANYACCOUNTINGID','100');
  TheUnitCost := GetCost(queryvalue('CODETYPE'),queryvalue('LINECODE'));
  TheUnitPrice := strtofloat(queryvalue('ORDERUNITPRICE'));
  if (TheUnitPrice <> 0) then
   begin
     TheCalcMargin := (((TheUnitPrice - TheUnitCost)/ TheUnitPrice) * 100);
     if (TheCalcMargin < TheMinMargin) then
      begin
       TheIntValueofMargin := int(TheCalcMargin * 100);
       TheCalcMargin := (TheIntValueofMargin /100);
       messagedlg('A ' + floattostr(TheCalcMargin) + ' % margin is below the Company
Minimum of ' + floattostr(TheMinMargin) + ' %',mtinformation,mbOK,0);
       {Comment out the abort function below if you just want to display a message only}
       abort;
      end
   end
  else
   begin
     if (TheUnitCost > 0) then
      messagedlg('This line has a Zero Price with a Cost',mtinformation,mbOK,0);
   end;
end.
```

Save and exit the Custom Script screen.

The next step is to tell Ostendo that the script is linked to a Sales Order Line. To do this go into ***File>System Configuration>Screen Data Scripts*** and create a new record containing the following

Screen: Select '**Sales Orders**' from the drop-down
Table Name: Select '**SALESLINES**' from the drop-down list
SQL Type: Select '**Insert**'
Script Name: Select the above script Name

Save and exit

Now go into ***Sales>Sales Orders*** and create a Sales Order then add a line to the Order. If you amend the sell price against the Item such that the resultant price falls below the above Margin then you will get a message returned and you will be prevented from saving the line.

This covers the situation where you are adding a new Sales Order Line but what about if you are amending an existing line. This can be covered by going back into ***File>System Configuration>Screen Data Scripts*** and create another record containing the following

Screen: Select '**Sales Orders**' from the drop-down
Table Name: Select '**SALESLINES**' from the drop-down list
SQL Type: Select '**Update**'
Script Name: Select the above script Name

Save and exit

2.7 Workflow Script

These are scripts that can be used to change the appearance and/or Text of a Workflow Object. These include amending the Text, Colour, Hint, Visible/Hide, etc.

To see this in action we will change the colour of a Workflow Object.

You should start by bringing a workflow to the desktop. If you don't have a workflow of your own then you should use the workflow supplied with Ostendo. To bring this to the desktop carry out the following.

Click on ***File>System Configuration>User Security and Options*** and click on the 'Workflow' tab. In that screen 'check' the 'Enable Workflow' checkbox then click in the centre of the lower panel. Click the 'Add' button and add the following line:

> **Caption**: **Workflow**
> **Filename**: Locate **SampleWorkflow.dat** which can be found under the Ostendo folder
> **Sequence**: Leave 'as is'

Save and close out of the screen. You will find that the workflow now displays on your Ostendo Desktop.

In the workflow screen (NOT on an object) 'right mouse' and select '**Edit Workflow**'. The Workflow Editor will start up. In the Editor click on and object (to select it) then note the ID number in the Inspector Options (down the right hand side). If you have used Workflow **SampleWorkflow.dat** then you will find that the 'Customers' Object will have an ID of **32**.

Go into ***File>Custom Scripts*** and add a new Custom Script called (say) **Workflow**. In the 'Script' tab enter the following:

```
Begin
        SetWorkflowObjectColour(32,claqua);
        Showmessage('Colour Updated');
End.
```

Where 32 is the Object ID

Save and exit

If you now run the Script (**Custom>Workflow**) you will see that the fill colour of the Object will take place.

2.8 Report Layout Editor

Scripting can be used in the Report Layout Editor to control many printing options such as printing fields under certain data conditions, etc. There are two main elements in the Report that controls the script.

> When is the action to take place
> What action is to take place

To demonstrate this go into ***File>Reporting Configuration>Report and View Developer*** and click the '**Add**' button. Copy the '**Item Detail Sheet**' to your Company Reports folder. Select the copied report and click on the '**Master Settings**' tab then on the '**Edit**' button. On the report layout scroll down to the '**Child4**' Band. You will see a small red triangle (?) in the band. This denotes that it has some code linked to it. If you click on the Band then we will address this linked code as follows

When is the action to take place: If you click on the '**Events**' tab to the left of the screen you will see that the 'when' is '**OnBeforePrint**' and it relates to a script procedure in the adjacent field. (In this instance **Child4OnBeforePrint**.)

What action is to take place: If you double-click on the **Child4OnBeforePrint**.it will take you to the ‘**Code**’ tab and position you at the **Child4OnBeforePrint**. Procedure. You will see that script provides a visible = **True** or **False** depending upon the content of the Notes field

3 Basic Script Structure

The Basic Script Structure is broken down into 4 main sections.

The Variables Section

A variable is a user-defined attribute against which you can evaluate and store an evaluated result for output or further action. Multiple variables can be declared under a heading **var**. For Example:

```
Var
  TheCustomerName: String;
  TheInvoiceCount: integer;
  ErrorFound : Boolean;
```

Each variable is declared using the same standard format;

VariableName	your defined name – No Spaces
:	Colon to denote the end of the VariableName
Format	The Format of the data
;	Semi-colon to denote end of variable

The more common **Formats** are:

String – AlphaNumeric characters
Double – Number with decimals
Integer – Number with no decimals
Boolean – True or False

However you can also use the following:

Byte: Same as Integer
Word: Same as Integer
Integer: Basic Integer (Whole numbers only)
Longint: Same as Integer
Cardinal: Same as Integer
Tcolor: Same as Integer
Boolean: Returns True or False
Real: Same as Extended
Single: Same as Extended
Double: Same as Extended
Extended: A floating point type with the highest capacity and precision
TDate: Data type holding a date value (Same as Extended type)
TTime: Data type holding a time value (Same as Extended type)
TDateTime: Data type holding a date and time value (Same as Extended type)
Char: Variable type holding a single character
String: A data type that holds a string of characters
Variant: A variable type that can hold changing data types
Array: An Array type

The Constants Section

A Constant is a user-defined value that can be referenced at any time within the script Multiple Constants can be declared under a heading **const**. For Example:

```
Const
  pi = 3.14159;
   e = 2.71828496
```

Each constant is declared using the same standard format;

ConstantName	your defined name – No Spaces
=	Equals sign
Value	The defined value of the constant
;	Semi-colon to denote end of declared constant

The Main Process

The Main Process is a user-defined routine than performs a required activity. A simple example of a Main Process is:

```
var
  MyValue: Integer;
begin
  MyValue := 1;
  Showmessage(Myvalue + Myvalue);
End.
```

You should note the following

A **Var** is only required if you are storing a result as opposed to the script (say) updating a record.
The process itself has a **Begin** and **End** Statement
Each instruction within the process ends with a semi-colon
The final End statement ends in a full-stop

The Procedure Section

If a process is used more than once within the script you can avoid retyping the process by declaring a Procedure. A Procedure is a user-defined routine than performs a required activity. This allows you to simply call the Procedure Name whenever you require this process to be carried out.

A procedure can comprise of its own declared Variables as well as the process itself, For example

```
procedure FirstProcedure;
var
  MyValue: Integer;
begin
  MyValue := 1;
  Showmessage(Myvalue + Myvalue);
End;

Begin
  FirstProcedure;
End.
```

Comments

A couple of notes to complete this introduction:

1. You can place comments in the script in one of two ways

   // Anything on this line after the 'double slash' is ignored

   { Anything that is contained within 'squiggly Brackets' is ignored. This can go over many lines}

2. Pre-defined functions are 'Procedures' designed by Development-X that allow you enter simple information to perform a complex activity. These are shown in the next section

4 Constants, Variables, Functions, Procedures

The following script Constants are designed specifically for use in Ostendo.

4.1 Abort

For use with
  Screen Data Script

Format: **Abort;**

This function is used where the script carries out a user-defined check and, if a condition occurs, abort the current transaction. This can be used for example to check if a Sales Line margin has been achieved and - if not - stop the current Sales order Line entry. The element that makes up this function is a simple '**Abort'** statement. In the following example the Order Sell Price is compared against the default margin set up in System Settings and if the Margin is not met the line entry is ended with the 'Abort' function

```
var
 TheMinMargin,TheUnitPrice,TheUnitCost,TheCalcMargin: double;
 TheIntValueofMargin: integer;
begin
  TheMinMargin := GetDoubleFromTable(
'SYSTEMMASTER','MINMARGINPERCENT','COMPANYACCOUNTINGID','100');
  TheUnitCost := GetCost(queryvalue('CODETYPE'),queryvalue('LINECODE'));
  TheUnitPrice := strtofloat(queryvalue('ORDERUNITPRICE'));
  if (TheUnitPrice <> 0) then
   begin
     TheCalcMargin := (((TheUnitPrice - TheUnitCost)/ TheUnitPrice) * 100);
     if (TheCalcMargin < TheMinMargin) then
      begin
       TheIntValueofMargin := int(TheCalcMargin * 100);
       TheCalcMargin := (TheIntValueofMargin /100);
       ShowMessage('A ' + floattostr(TheCalcMargin) + ' % margin is below the Company
Minimum of ' + floattostr(TheMinMargin) + ' %');
       {Comment out the abort function below if you just want to display a message only}
       abort;
      end
   end
  else
   begin
     if (TheUnitCost > 0) then
      Showmessage('This line has a Zero Price with a Cost');
   end;
end.
```

4.2 AddContactsToOutlook

This procedure enables you to add Ostendo Contacts to your Microsoft Outlook Address Book

For use with
  General Custom Scripts
  Screen Data Script
  Order Script
  Custom Product Script

Format: AddContactsToOutlook(ContactDetails);

ContactDetails: These are the field details from Ostendo

This will copy selected Ostendo Contacts (found under CRM>Contacts) to Outlook Address Book. This example shows 'hard-coded' fields but you can, of course, populate these with selected records from your Contacts List

```
begin
    AddContactsToOutlook('ContactName=Ken_Jolly'
  + ',FirstName=Ken'
  + ',Title=Mr'
  + ',Position=Driver'
  + ',LastName=Jolly'
  + ',Phone=555-1111'
  + ',Fax=444-2222'
  + ',Mobile=333-1111'
  + ',CompanyName=KJ'
  + ',EmailAddress=kj@kj.com'
  + ',ContactNotes=Good Driver'
  );
  Showmessage('Copy Done');
end.
```

4.3 AddRelatedMenuItem

For use with
Edit View Scripts

Format: Function AddRelatedMenuItem('Screen Name');

This Function is used in combination with procedure RelatedMenuItemClicked and allows you to add a Related Screen to the 'Related' Button in the Edit View Panel

The elements that make up this function are:
Screen Name: The Name of the Screen to be called

In the following example we will assume that you already have created an Edit View and we are going to add a report to the 'Related' button in that view.

Procedure determines the 'Index Number' in the List of Related Items displayed under the 'Related' button.
To get the Index Number add the following to the end of the Edit View Script.

```
Begin
  AddRelatedMenuItem('Screen Name');
End.
```

Where you should replace 'Screen Name' with the actual name of your screen.

If you go into the Edit View then you will see this option presented when you click on the 'Related' button. Now let us run Procedure RelatedMenuItemClicked.

If it is the only report under the Related button then its Index will be 0 therefore add this to the Edit View Script under the RelatedMenuItemClicked Procedure.

```
procedure RelatedMenuItemClicked(MenuIndex: Integer);
begin
 If MenuIndex = 0 then
 begin
  RunSystemAction('General','Knowledge Base');
 end;
end;
```

If you go into the Edit View and select this under the 'Related' button you will find that the Knowledge Base screen will be run.

4.4 AddReportMenuItem

For use with
Edit View Scripts

Format: Function AddReportMenuItem('Report Name');

This Function is used in combination with procedure ReportMenuItemClicked and allows you to add your Report or Analysis View to the 'Reports' Button in the Edit View Panel

The elements that make up this function are:
Report Name: The Name of the Report as it appears in the Report and View Deloper

In the following example we will assume that you already have created an Edit View and we are going to add a report to the 'Reports' button in that view.

Procedure determines the 'Index Number' in the List of Reports displayed under the 'Reports' button.
To get the Index Number add the following to the end of the Edit View Script.

```
Begin
 AddReportMenuItem('Report Name');
End.
```

Where you should replace 'Report Name' with the name of your Report.

If you go into the Edit View then you will see this option presented when you click on the 'Reports' button. Now let us run Procedure ReportMenuItemClicked.

If it is the only report under the Reports button then its Index will be 0 therefore add this to the Edit View Script under the ReportMenuItemClicked Procedure.

```
procedure ReportMenuItemClicked(MenuIndex: Integer);
begin
 If MenuIndex = 0 then
 begin
  OstendoReport('Report Name');
 end;
end;
```

If you go into the Edit View and select this under the 'Reports' button you will find that the report will be run.

4.5 AllocateToJobOrder

For use with

General Custom Scripts

Format: AllocateToJobOrder(OrderNumber, TaskName, LineCode, CodeType, LineUnit, LineDescription, LinkedCatalogueName, AllocSysID);

Within Ostendo when you create a Purchase Order, Receive Goods without a prior Purchase Order, or Receive an Invoice without a prior Receipt you have the option to create a Job Order Line and a 'Line Source' record that is linked to this Order, Receipt, or Invoice. This function:

- Allows you to carry out the above action when importing Invoices from a file or document received from your Supplier.
- Allows you to also allocate a Purchase receipt and/or Purchase Invoice to a Job Order subsequent to the Receipt or Invoice being entered into Ostendo

The elements that make up this function are:

OrderNumber: The existing Job Order to which the Purchase Order, Receipt, or Invoice is being allocated.
TaskName: The Task Name currently held within the Job Order
LineCode: The Item, Descriptor, Catalogue Code, or Kitset Code that is being allocated
CodeType: This relates to the Line Code and must be one of 'Item Code', 'Descriptor Code', 'Kitset Code', or 'Catalogue Code'
LineUnit: The Unit of Measure of the Line Code. This must be valid for the Line Code.
LineDescription: A Description of the Line Code. This can be any description and is not validated against the base LineCode record
LinkedCatalogueName: If the CodeType is 'Catalogue Code' then you must also provide a valid Catalogue Code Name that currently exists in Ostendo.
AllocSysID: This is the SysUniqueID of the Order, Receipt, or Invoice Allocation line. This is passed through to the Job Line's 'Line Source' record

An example showing how you can allocate a previously entered Purchase Invoice to a Job Order can be seen as 'Reallocate Purchase Invoice Lines' in the 'Useful Scripts' section of this Help Guide

4.6 AskMandatoryQuestion

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskMandatoryQuestion(Question, QuestionType, FullExplanation, [d]ValueList, DefaultValue);

This type of question includes a defined list from which the answer must be selected The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Question: Define the question (max 200chars) enclosed in 'commas'
Question Type: This is INTEGER, TEXT, NUMERIC, DATE, BOOLEAN, or TEXT LIST
Full Explanation: A longer explanation that can be referenced during configuration
[d]: Optional entry to define a default display entry from the List
Value List: Allowable options - separated by a comma
Default Value: Optional Entry to prefill question with current answer if re-doing questions

The following example defines a Variable ‘TimberType’ and then asks the question from which either Pine or Rimu is selected. After selection a message will be presented showing the value now held in variable TimberType

```
// Define the Variable
Var
 TimberType: String;
// You can then ask the question to answer TimberType using
  Begin
           TimberType := AskMandatoryQuestion('Please select the type of Timber ','TEXT',
           'Rimu provides a better finish','[d]Pine,Rimu',TimberType);
           Showmessage('Your selection is ' + TimberType);
  End.
```

4.7 AskMandatoryQuestionWithLookup

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskMandatoryQuestionWithLookup(Question, FullExplanation, LookupIndex, DefaultValue);

This type of question includes a link to an Ostendo Table from which the answer must be selected

The elements that make up this function are:
Variable: The defined variable against which the result will be held.
Question: Define the question (max 200chars) enclosed on ‘commas’
Full Explanation: A longer explanation that can be referenced during configuration
LookupIndex: The Index Reference of the Ostendo Table
Default Value: Optional Entry to prefill question with current answer if re-doing questions

The following example defines a Variable ‘SelectedItem’ and then asks the question. The drop-down list shows data from the defined Ostendo table (See Appendix A).   After making a selection a message will be presented showing the value now held in variable SelectedItem

```
// Define the Variable
Var
 SelectedItem: String;
// You can then ask the question to answer SelectedItem using
  Begin
           SelectedItem := AskMandatoryQuestionWithLookup('Please select the Item
           Number','User Defines what is to be used',1004,SelectedItem);
           Showmessage('Your selection is ' + SelectedItem);
  End.
```

4.8 AskQuestion

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskQuestion(Question, QuestionType, FullExplanation, ValueList, DefaultValue);

This type of question can optionally include a defined list from which the answer may optionally be selected   The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Question: Define the question (max 200chars) enclosed on 'commas'
QuestionType: This is INTEGER, TEXT, NUMERIC, DATE, BOOLEAN, or TEXT LIST
FullExplanation: A longer explanation that can be referenced during configuration
[d]: Optional entry to define a default display entry from the List
ValueList: Allowable options - separated by a comma
Default Value: Optional Entry to prefill question with current answer if re-doing questions

The following example defines a Variable 'OptionSelect' and then asks the question from which you can (optionally) type in an answer. After keying in the option a message will be presented showing the value now held in variable OptionSelect

```
// Define the Variable
Var
 OptionSelect: String;
// You can then ask the question to answer OptionSelect using
  Begin
      OptionSelect := AskQuestion('Please Type in your option','TEXT','You can leave
      blank if you wish','');
      Showmessage('Your selection is ' + OptionSelect);
  End.
```

4.9 AskQuestionNumericRange

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskQuestionNumericRange(Question, QuestionType, FullExplanation, MinValue, MaxValue, NumberInc, DefaultValue);

This type of question includes a minimum and maximum range along with - optionally - incremental steps. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Question: Define the question (max 200 chars) enclosed on 'commas'
Question Type: this is either NUMERIC or INTEGER
Full Explanation: A longer explanation that can be referenced during configuration
MinValue: The numeric value denoting the minimum value allowed during data entry
MaxValue: The numeric value denoting the maximum value allowed during data entry

NumberInc: A numeric value denoting the incremental steps within the range. If nothing is entered then any value in the range will be accepted.
Default Value: Optional Entry to prefill question with current answer if re-doing questions

The following example defines a Variable ‘DeskLength’ and then asks the question from which you enter the length. After keying in the Length a message will be presented showing the value now held in variable DeskLength

```
// Define the Variable
Var
 DeskLength: Integer;
// You can then ask the question to answer DeskLength using
  Begin
         DeskLength := AskQuestionNumericRange('Please enter the Length (mm) of the
         Desk', 'INTEGER','We can only make desks between 1000mm - 3000mm in
         length',1000,3000,10,DeskLength);
         Showmessage('Your Desk Length is ' + inttostr(DeskLength));
// Note how we converted from Integer to String to complete the message
  End.
```

4.10 AskQuestionWithChecklist

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskQuestionWithChecklist(Question, FullExplanation, ValueList);

This type of question includes a pre-defined list from which multiple selections can be made. The elements that make up this function are:
Variable: The defined variable against which the result will be held.
Question: Define the question (max 200chars) enclosed on ‘commas’
FullExplanation: A longer explanation that can be referenced during configuration
ValueList: Allowable options - separated by a comma

The following example defines a Variable ‘OptionsSelect’ and then asks the question from which you can ‘check’ one or more entries. After clicking the ‘Answer’ button a message will be presented showing the values now held in variable OptionsSelect

```
// Define the Variable
Var
  OptionsSelect: String;
// You can then ask the question to answer OptionsSelect using
Begin
  OptionSelect := AskQuestionWithChecklist('Please Select your options','You can leave
blank if you wish','Blue,Green,Red');
  Showmessage('Your Selections are ' + OptionsSelect);
End.
```

4.11 AskQuestionWithDBChecklist

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskQuestionWithDBChecklist(Question, FullExplanation, CheckListSQL);

This type of question includes a link to an Ostendo Table from which multiple selections can be made. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Question: Define the question (max 200chars) enclosed on ‘commas’
Full Explanation: A longer explanation that can be referenced during configuration
CheckListSQL: The Query that extracts the fields. These are displayed in a drop-down list during the question/Answer process

The following example defines a Variable ‘SelectedDescriptors’ and then asks the question. The drop-down list shows data extracted via the Query. After making selection(s) a message will be presented showing the value(s) now held in variable SelectedDescriptors

```
// Define the Variable
Var
 SelectedDescriptors: String;
// You can then ask the question to answer ‘SelectedDescriptor’ using
  Begin
        SelectedDescriptors := AskQuestionWithDBChecklist('Please select the
        Descriptor Codes','User selects what is to be used', 'Select * from
        DescriptorMaster');
        Showmessage(‘Your selections are ’ + SelectedDescriptors);
  End.
```

4.12 AskQuestionWithLookup

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskQuestionWithLookup(Question, FullExplanation, LookupIndex, DefaultValue);

This type of question includes a link to an Ostendo Table from which the answer may optionally be selected

The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Question: Define the question (max 200chars) enclosed on ‘commas’
Full Explanation: A longer explanation that can be referenced during configuration
LookupIndex: The Index Reference of the Ostendo Table
Default Value: Optional Entry to prefill question with current answer if re-doing questions

The following example defines a Variable ‘SelectedDescriptor’ and then asks the question. The drop-down list shows data from the defined Ostendo table (See section on 'Lookup Numbers'). After making a selection a message will be presented showing the value now held in variable SelectedDescriptor

```
// Define the Variable
Var
 SelectedDescriptor: String;
// You can then ask the question to answer ‘SelectedDescriptor’ using
  Begin
        SelectedDescriptor := AskQuestionWithLookup('Please select the Descriptor
        Code','User selects what is to be used',1010,SelectedDescriptor);
        Showmessage('Your selection is ' + SelectedDescriptor);
  End.
```

4.13 AskQuestionWithUserDefinedLookup

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := AskQuestionWithUserDefinedLookup(LookupSQL,Question, FullExplanation, LookupIndex, DefaultValue);

This type of question includes a Query that allows you to extract data to select records from which the specific Variable value can be selected

The elements that make up this function are:

LookupSQL: A standard Query enclosed in single quotes
Question: Define the question (max 200chars) enclosed on ‘commas’
Full Explanation: A longer explanation that can be referenced during configuration
Default Value: Optional Entry to prefill question
LookupTitle: The title that will appear in the displayed panel
Result Field: The field from a selected record that returns the value
LookupHeight: Height of the displayed panel in pixels. Default = 320
LookupWidth: Width of the displayed panel in pixels. Default = 440

The following example defines a Variable ‘TheLostQuote’ and then asks the question. The drop-down list shows data from the Query. After making a selection a message will be presented showing the value now held in variable TheLostQuote

```
// Define the Variable
Var
 TheLostQuote: String;
// You can then select the data and ask the question for answering ‘TheLostQuote’ using
  Begin
        TheLostQuote := AskQuestionWithUserDefinedLookup('Select * from
        SalesHeader where OrderStatus = "Lost"','Select the Lost Sales Quote','','','Select
        Quote','OrderNumber',350,500);
        Showmessage('Your selection is ' + TheLostQuote);
  End.
```

4.14 ClearValueStore

For use with
- *General Custom Scripts*
- *Screen Data Script*
- *Order Script*
- *Custom Product Script*

Format: **ClearValueStore;**

This function will clear all values in store previously created by the SaveValueToStore function.

```
Begin
        ClearValueStore;
End.
```

4.15 CloseOstendo

For use with
- *System Action Script*
- *Standard Script*
- *Related Script*
- *Screen Data Script*
- *Order Script*
- *Accounting Link Script*
- *Custom Data Script*
- *API Script*
- *Edit View Script*
- *Web Service Script*

Format: **CloseOstendo**

As the procedure suggests it closes the current copy of Ostendo.

```
Begin
  CloseOstendo;
End.
```

4.16 CloseScreen

For use with
- *System Action Script*
- *Standard Script*
- *Related Script*
- *Screen Data Script*
- *Order Script*
- *Accounting Link Script*
- *Custom Data Script*
- *API Script*
- *Edit View Script*
- *Web Service Script*

Format: **CloseScreen(Screen)**

As the procedure suggests it closes the nominated Ostendo screen. In this example we will close the Items screen.

```
Begin
  CloseScreen('Items');
End.
```

4.17 ConvertToUniversalTime

Universal Time (UT) is a timescale based on the rotation of the Earth and is a modern continuation of Greenwich Mean Time (GMT). All emails are sent using the Universal Time and adjusted by the local time at the recipients email location. All Ostendo-generated emails, therefore, should be converted to Universal Time before they are sent.

For use with
Order Scripts for use with email options

Format: Variable:= ConvertToUniversalTime(DateTime);

Variable: The defined variable against which the result will be held.
DateTime: The Date and Time to be converted to Universal Time

For example this script will take the current Date and Time from your computer and convert it to Universal Time.

```
Var
  CurrentDateAndTime: TDateTime;
  DateAndTime: TDateTime;
begin
 CurrentDateAndTime:= NOW;
 DateAndTime:= ConvertToUniversalTime(NOW);
 Showmessage('Current Date and Time   = ' + datetimetostr(CurrentDateAndTime) + #13
+ 'Universal Date and Time = ' + datetimetostr(DateAndTime));
end.
```

4.18 CopyFile

This procedure enables you to copy any file on your network to any other location. This can optionally be used in combination with Procedures ***DeleteFile, MoveFile, RenameFile*** and ***CreateDir***

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: CopyFile(Source File, Destination File, Overwrite);

Source File: The full path of the file to be copied
Destination File: The full path of the file at its destination
Overwrite: ‘True’ if you wish to overwrite any existing file of the same name, else ‘False’

In this example we will copy a file from the ‘C’ Drive to the ‘D’ Drive and overwrite the file if it currently exists at the Destination.

```
begin
  CopyFile('C:\Temp\CopyScript.doc','D:\CopyScript.doc',True);
  Showmessage('File Copied');
end.
```

4.19 CreateDir

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script
Custom Data Screens
Data Entry Script

Format: CreateDir(Directory);

This will create a Directory on you network into which you would store information, etc, possibly from other instructions within the whole script. Note: This function returns True to denote if the Directory was created, or False if it was not created - for example due to permissions)

Directory: The full path of the Directory being created

In this example we will create a Directory ‘Scripts’ under the Ostendo Folder

```
Begin
        CreateDir(‘C:\Program Files\Ostendo\Scripts’);
End.
```

Of course you would replace C:\Program Files\Ostendo\Scripts with the configuration you use on your network

4.20 CurrentUser

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script
Custom Data Screens
Data Entry Script

Format: CurrentUser;

This is a Constant that is used to determine the User who is running the script and could include that User ID in any actions derived from the script

In this example we will use the path related to your PC

```
Begin
        Showmessage(CurrentUser);
End.
```

4.21 CustomLineNotes

For use with
Custom Product Script

Format: CustomLineNotes;

This Custom Product Script variable allows you to construct Notes that will populate the Notes field in the Sales or Job Order line created for this Custom Product.

Example:

```
CustomLineNotes('This product is made from' + #13 +
        'Front: ' + FaciaMaterial + #13 +
        'Side: ' + SideMaterial + #13 +
        'Legs: ' + LegMaterial);
```

4.22 CustomOrderLineID

For use with
Custom Product Script

Format: CustomOrderLineID;

When used in a script this Custom Product Constant holds the current Line’s SYSUNIQUEID of the Sales or Job Order. You may wish to use this in the remainder of the script.

For example, create the following Custom Product Script linked to an Item and then add the Item to a Sales Order. The Sales Order Line’s SYSUNIQUEID will be held against this Constant and displayed in the Showmessage

```
Begin
        Showmessage(CustomOrderLineID);
End.
```

4.23 CustomProductCode

For use with
Custom Product Script

Format: CustomProductCode;

This Custom Product Script variable allows you to construct a Custom Product Code – usually from answers made during the Configuration process. This is used in conjunction with CustomProductCodeType

Example:

```
CustomProductCode(‘PC’, + Answer1 + ‘-‘ + Answer2 + ‘Custom’);
```

4.24 CustomProductCodeType

For use with
Custom Product Script

Format: CustomProductCodeType;

This Custom Product Script variable allows you to specify what the Code Type of the generated Product will be. The allowable options are ‘Item Code’, ‘Descriptor Code’, ‘Kitset Code’, or ‘ Catalogue Code’. This is used in conjunction with CustomProductCode to allow you to make a Customised Product ID.

Example:

```
CustomProductCodeType('Item Code');
```

4.25 CustomProductCustomer

For use with

Custom Product Script

Format: CustomProductCustomer;

When used in a Custom Product script this Constant holds the current Customer for the configured Product. This can then be used within the remainder of the script to determine Customer specific values when designing a Custom Product

For example, create the following Custom Product Script linked to an Item and then add the Item to a Sales Order. The Sales Order Customer will be held against this Constant and displayed in the Showmessage

```
Begin
        Showmessage(CustomProductCustomer);
End.
```

4.26 CustomProductDescription

For use with

Custom Product Script

Format: CustomProductDescription;

This Custom Product Script variable allows you to construct a Description of up to 100 characters that will populate the Description field of the generated Sales or Job Order Line

Example:

```
CustomProductDescription('Stainless Steel Shelf Assembly ' + Length + 'Mtrs Long and ' + Width + ' Mtrs Wide');
```

4.27 CustomSalesOrderNumber

For use with

Custom Product Script

Format: CustomSalesOrderNumber;

When used in a Custom Product Script this Constant holds the current Sales or Job Order Number. You may wish to use this in the remainder of the script.

For example, create the following Custom Product Script linked to an Item and then add the Item to a Sales Order. The Sales Order Number will be held against this Constant and displayed in the Showmessage

```
Begin
        Showmessage(CustomSalesOrderNumber);
End.
```

4.28 CustomSalesLineID

For use with
> ***Custom Product Script***

Format: CustomSalesLineID;

When used in a Custom Product Script this Constant holds the current Sales or Job Order Line Number. You may wish to use this in the remainder of the script.

For example, create the following Custom Product Script linked to an Item and then add the Item to a Sales Order. The Sales Order Line Number will be held against this Constant and displayed in the Showmessage

```
Begin
        Showmessage(CustomSalesLineID);
End.
```

4.29 DataEntryCancel

For use with
> ***Data Entry Scripts***

This closes the Data Entry screen by setting the result of function DataEntryShow to False from within the script. This, effectively, performs the same action as clicking the ‘X’ at the top of Data Entry screen.

> **Format: DataEntryCancel;**

4.30 DataEntryCellValueGet

For use with
> ***Data Entry Scripts***

This is used to get the value from a specific field within a record in the Table

> **Format: DataEntryCellValueGet(RecordIndex, ColumnIndex);**

The elements that make up this function are:
> **RecordIndex**: The record number being addressed (starting from 0)
> **ColumnIndex**: The field number in the record (starting from 0)

4.31 DataEntryCellValueSet

For use with
> ***Data Entry Scripts***

This function enables you to populate a specific field within a record with a defined value. The format of this function is:

Format: DataEntryCellValueSet(RecordIndex,ColumnIndex,CellValue);

The elements that make up this function are:
RecordIndex: The record number being addressed (starting from 0)
ColumnIndex: The field number in the record (starting from 0)
CellValue: The value to populate the field

4.32 DataEntryColumnCount

For use with
Data Entry Scripts

This function enables you to determine the number of columns in the Grid. The format of this function is:

Format: DataEntryColumnCount;

4.33 DataEntryColumnCreate

For use with
Data Entry Scripts

This function is used to define the sequence and format of the data entry columns. The format of this function is:

Format: DataEntryColumnCreate(Caption, ColumnWidth{Opt Default=100}, EditorType{Opt Default='TEXT'}, LookupIndex{Opt Default=0});

The elements that make up this function are:
Caption: The caption that populates the column heading
ColumnWidth: The width of the column
EditorType: Defines the format of the entry field. The options are
TEXT: Open format entry
DATE: Entry format as your Regional Settings
CALC: This allows you to enter a decimal number. You also have the option to bring up a calculator to calculate the entered value.
CURRENCY: Format as your Regional Settings
SPIN: An Integer only entry allowed. This option also shows arrows from which you can incrementally increase or decrease the displayed Integer
TIME: Entry format as your Regional Settings
CHECKBOX: A 'Yes/No 'Boolean' checkbox
LOOKUP: Used in combination with ***LookUpIndex***

LookUpIndex: This allows you to reference any Ostendo table that has a LookUpIndex. (For a list of these refer to section 'Condition Indexes' in Ostendo Help covering 'Reporting '.)
When using a LookUpIndex the EditorType must be LOOKUP. When answering the question a drop-down list is presented showing the current entries in that Table from which a selection can be made.

4.34 DataEntryCreate

For use with

Data Entry Scripts

This function will create the Data Entry panel. The elements that make up this function are:

Format: DataEntryCreate(Title{Opt Default='Data Entry'}, FormHeight{Opt Default=400}, FormWidth{Opt Default=550});

Where the elements represent the following

Title: The Title that will appear at the top of the form
Height: The Height of the Form in Pixels
Width: The Width of the Form in Pixels

4.35 DataEntryFocusedColumnIndex

For use with

Data Entry Scripts

This function returns the Column Index where the cursor is located within the grid. This can be used, for example, to provide a reference for further processing data in a column using (for example) DataEntryCellValueGet.

Format: DataEntryFocusedColumnIndex;

4.36 DataEntryFocusedRecordIndex

For use with

Data Entry Scripts

This function returns the Record Index of the record where the cursor is located within the grid. This can be used, for example, to return the selected record index when OK is clicked, thus emulating a ‘Lookup’ selection.

Format: DataEntryFocusedRecordIndex;

4.37 DataEntryNewRecordValuesSet

For use with

Data Entry Scripts

This function enables you to prefill fields when creating records. The format of this function is:

Format: DataEntryNewRecordValuesSet(RecordValues);

The elements that make up this function are:
RecordValues: The default values in each filed separated by a comma

4.38 DataEntryOK

For use with
Data Entry Scripts

This closes the Data Entry screen and sets result of DataEntryShow to True from with script. This is the same as the user clicking the OK button on screen. This function can be used to force the close of the Data Entry if needed prior to the user closing it and the OK/Cancel.

Format: DataEntryOK;

4.39 DataEntryRecordCount

For use with
Data Entry Scripts

When you enter the records they are stored in a temporary table until you tell Ostendo what to do with them. This function simply counts the number of records in this temporary table. The format of this function is:

Format: DataEntryRecordCount;

4.40 DataEntrySetLabel

For use with
Data Entry Scripts

This function enables you to create a Label that appears across the top of the grid. The format of this function is:

Format: DataEntrySetLabel(Title);

The elements that make up this function are:
Title: The Title of the Grid

4.41 DataEntryShow

For use with
Data Entry Scripts

This function simply instructs Ostendo to display the Entry Form

Format: DataEntryShow;

4.42 DataScreenActiveScheme

For use with
Data Screen Scripts

This function allows you to determine the Identity of the current Active Scheme

The elements that make up this function are:

Format: DataScreenActiveScheme(SchemeID);

SchemeID: The Identifier of the Scheme which you are changing to

4.43 DataScreenChangeScheme

For use with
Data Screen Scripts

This function allows you to go to another Scheme in the same Ostendo Graphic.

The elements that make up this function are:

Format: DataScreenChangeScheme(SchemeID);

SchemeID: The Identifier of the Scheme which you are changing to.

4.44 DataScreenClose

For use with
Data Screen Scripts

This function is used to close the screen display.

The format of this function is:

Format: DataScreenClose:

4.45 DataScreenGetObjectText

For use with
Data Screen Scripts

This function is used to get the current content of a Graphical Object's 'Text'.

Format: DataScreenGetObjectText(ObjectID);

The element that makes up this function is:
ObjectID: The Identifier of the Object in the Graphical and View Developer. The returned Text from this Object would populate a defined Variable

To find the Object ID go into the Graphic and click on the Object. You will find the ID of the Object in the adjecent '**Inspector**' panel

4.46 DataScreenObjectLoadPicture

For use with
Data Screen Scripts

This function is used to load a Graphical Object with a picture.

Format: DataScreenObjectLoadPicture(ObjectID,Picture);

The elements that make up this function are:

ObjectID: The Identifier of the Object in the Graphical and View Developer. This MUST be a 'Picture' Object

Picture: The full path of the picture enclosed in single quotes. For example 'C:\Program Files\Ostendo\MyPicture.jpg'

You should note that the Graphical Object must be a picture object.

4.47 DataScreenQuestion

For use with

Data Screen Scripts

This function is used to prompt for data entry. The result of the data entry populates the 'Value' held against the Question Index.

Format: DataScreenQuestion(QuestionIndex, Question, EditorType, FullExplanation, ValueList, DefaultValue, LookupIndex);

The elements that make up this function are:

QuestionIndex: A sequential number, commencing at zero to denote the unique question and the order in which the question is presented

Question: Define the question (max 200chars) enclosed in 'Single Quotes'

EditorType: Defines the format of the answer. The options are

TEXT: Open format entry

DATE: Format as your Regional Settings

COMBOBOX: Creates a drop-down list of entries in ***Value List***

CALC: This allows you to enter a decimal number. You also have the option to bring up a calculator to calculate the entered value.

CURRENCY: Format as your Regional Settings

SPIN: An Integer only entry allowed. This option also shows arrows from which you can incrementally increase or decrease the displayed Integer

TIME: Format as your Regional Settings

LOOKUP: Used in combination with ***LookUpIndex***

If this is defined as blank (two single quotes) then TEXT is assumed.

FullExplanation: A longer explanation that is displayed during data entry

ValueList: If you wish to select entries from a pre-defined list then you should enter the allowable options - separated by a comma. When using a Value List the EditorType must be COMBOBOX. When answering the question a drop-down list is presented showing these options from which a selection can be made. You should note that this does NOT prevent the user you from entering a value that is not in this list. You should exclude these within your script

If you are not using this feature then define this with two single quotes

DefaultValue: If you are using a ValueList then you can enter a default Value that prefills the data entry field

LookUpIndex: This allows you to reference any Ostendo table that has a LookUpIndex. (For a list of these refer to section 'Condition Indexes' in Ostendo Help covering 'Reporting '.)

When using a LookUpIndex the EditorType must be LOOKUP. When answering the question a drop-down list is presented showing the current entries in that Table from which a selection can be made.

4.48 DataScreenSaveGraphicalFile

For use with
Data Screen Scripts

This function will save the current changes by overwriting the source Graphical File. If this is not run then, if the Data Screen is amended during the execution of the Script, any changes will not be saved

Format: DataScreenSaveGraphicalFile;

4.49 DataScreenSetEditText

For use with
Data Screen Scripts

This function will move data into the input field. This could be, for example, a selected line from an Ostendo Table, a hardcoded text, a 'Touch Screen' button, etc. The format is:.

Format: DataScreenSetEditText('YourText');

The element that makes up this function is:
YourText: The text being set. This can be text entered here (defined in quotes) or can refer to a Var

4.50 DataScreenSetObjectColour

For use with
Data Screen Scripts

This function allows you to amend the fill colour of an Object within the Graphical Data Screen. The Object must not be currently set to 'Gradient Fill'. The elements that make up this function are:

Format: DataScreenSetObjectColour(ObjectID, ObjectColour);

Object ID: The Identifier of the Object in the Graphical Developer
ObjectColour: See the defined colours in the Graphical Editor. The Colour selection is the standard colour preceded with the letters cl. For example you can enter either clAqua

4.51 DataScreenSetObjectGradientColour

For use with
Data Screen Scripts

This function allows you to amend the Gradient fill colour of an Object within the Graphical Data Screen. The Object must be currently set to 'Gradient Fill'. The elements that make up this function are:

Format: DataScreenSetObjectGradientColour(ObjectID, BeginColour,EndColour);

Object ID: The Identifier of the Object in the Graphical Developer

BeginColour: The first colour in the Gradient. See the defined colours in the Graphical Editor. The Colour selection is the standard colour preceded with the letters cl. For example you can enter either clAqua
EndColour: The second colour in the Gradient. See the defined colours in the Graphical Editor.

4.52 DataScreenSetObjectHint

For use with
Data Screen Scripts

This function allows you to add or amend a Hint to an Object. A 'Hint' appears whenever the cursor is passed over the object. The elements that make up this function are:

Format: DataScreenSetObjectHint(ObjectID,Hint);

Object ID: The Identifier of the Object in the Graphical Developer
Hint: The hint that will replace the current hint.

4.53 DataScreenSetObjectText

For use with
Data Screen Scripts

This function is used to populate the text in a defined Graphical Object. The format of this function is:

Format: DataScreenSetObjectText(ObjectID,Value);

It is used in conjunction with the ***DataScreenOnValueEntered*** Procedure. In that Procedure a variable 'Value' has been declared which stores the entered Value against the QuestionIndex number.

This function (which is used against a specific QuestionIndex) takes the content of 'Value' and adds it to the 'Text' for the defined Object ID.

The elements that make up this function are:
ObjectID: The Identifier of the Object in the Graphical Developer
Value: Takes the content of the variable 'Value' to populate the Object

4.54 DataScreenSetObjectTransparency

For use with
Data Screen Scripts

This function allows you to define the 'opaqueness' of the Object. The elements that make up this function are:

Format: DataScreenSetObjectTransparency(ObjectID,Transparency);

Object ID: The Identifier of the Object in the Graphical Developer
Transparency: The amount of 'opaqueness. This ranges from 0 = solid to 100 = Invisible

4.55 DataScreenSetObjectVisible

For use with

Data Screen Scripts

This function allows you to turn off or on the visibility of an Object. The elements that make up this function are:

Format: DataScreenSetObjectVisible(ObjectID,Value);

Object ID: The Identifier of the Object in the Graphical Developer
Value: This can be True or False

4.56 DataScreenShow

For use with

Data Screen Scripts

This function is used to activate the Data Screen and accept entries.

The format of this function is:

Format: DataScreenShow:

If this is not included in the script then the Data Screen will not be displayed.

4.57 DBValueExists

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: DBValueExists('TableName','FieldName',FieldValue,CaseSensitive);

This type of question allows you to check data in the database, which returns a True/False (Boolean) response.

The elements that make up this function are:
Table Name: The Ostendo Table Name.
Field Name: The Ostendo field Name within the above Table
Field Value: The definitive value within the field being checked
Case Sensitive: A 'True' or 'False' entry to carry out a case sensitivity check on the data

In this example we are checking that a specific Item Code exists and return a message stating what was found

```
Begin
  If DBValueExists('Itemmaster','ItemCode','100-2000',True) then
    Showmessage('Item Exists')
  Else Showmessage('Item Does Not Exist');
End.
```

4.58 DeleteFile

This procedure enables you to delete any file on your network. This can optionally be used in combination with Procedures ***CopyFile, MoveFile, RenameFile*** and ***CreateDir***

For use with
- *General Custom Scripts*
- *Screen Data Script*
- *Order Script*
- *Custom Product Script*

Format: DeleteFile(FileToBeDeleted);

FileToBeDeleted: The full path of the file to be deleted

In this example we will delete file MyScript.doc

```
begin
  DeleteFile('D:\Temp\CopyScript.doc');
  Showmessage('File Deleted');
end.
```

4.59 DeleteFileFTP

For use with
General Custom Scripts

Format: DeleteFileFTP(Host, User, Password, FileName, PassiveMode{Opt Default=False}, Port{Opt Default=21});

This is used to delete a file in a remote FTP folder. Should be used with care.

Related script functions are ***FTPList*** and ***DownloadFileFTP***.

Example:

```
Const

TheHost = 'ftp.test.info' ;  // enter your ftp host name here
TheUser = 'testuser' ;  // enter your ftp username here
Thepassword = 'testpassword' ;  // enter your ftp password here


var

TheFilename : string;

begin
try

  TheFileName := 'public_html/testfolder/testfile.txt' ;  // enter your ftp
folder/filename here
```

```
  DeleteFileFTP(TheHost, TheUser, ThePassword, TheFileName, False,21);

  showmessage(TheFilename + ' is deleted.');

except
  showmessage(exceptionmessage) ;
end;

end.
```

Here is another example - how to delete the contents of a FTP folder:

```
Const

TheHost = 'ftp.test.info' ;  // enter your ftp host name here
TheUser = 'testuser' ;  // enter your ftp username here
Thepassword = 'testpassword' ;  // enter your ftp password here

var
 FileList: TStringList;
TheFolder : String ;
 i: Integer;
begin
  FileList := TStringList.Create;
 TheFolder :=  'public_html/testfolder' ;  // enter your ftp folder name here
  try
   FileList.Text := FTPList(TheHost,TheUser,ThePassword,TheFolder,True,21) ;
   for i := 0 to FileList.Count-1 do
   begin
     if trim(FileList.Strings(i)) <> '.' then
     if trim(FileList.Strings(i)) <> '..' then
     DeleteFileFTP
(TheHost,TheUser,ThePassword,TheFolder+'/'+FileList.Strings(i),True,21) ;
  end;
  finally
    FileList.Free;
  end;

end.
```

4.60 DetailCustomLookup

For use with
Edit View Scripts

Format: Procedure DetailCustomLookup(FieldName: String);

This Procedure is used in combination with Procedure DetailCustomLookupButtonClick and allows you to define that a field value in the Detail tab is derived from a drop-down Lookup where the Lookup content is defined in the script.

In the following example we will assume that you already have created an Edit View and we are going to create a Custom Lookup against (say) field 'Customer'

Procedure DetailCustomLookup enbles you to nominate the Detail record field against which the lookup will appear. Add the following to the end of the Edit View Script.

```
Begin
  DetailCustomLookup('Customer');
End.
```

The next step is to declare the Custom Lookup under Procedure DetailCustomLookupButtonClick as follows:

```
procedure DetailCustomLookupButtonClick(DisplayValue: Variant; AField: TField);
var
  LookupResult: String;
begin
   if AField.Fieldname = 'Customer' then
  begin
    LookupResult := DisplayData('Select CUSTOMER from CUSTOMERMASTER', '
Customer Lookup', 'CUSTOMER');
  if trim(LookupResult) <> '' then
  begin
    AField.DataSet.Edit;
    AField.AsString := LookupResult;
  end;
 end;
end;
```

If you go into the Edit View you will see a 'spyglass' symbol in the Customer field which, when clicked, will bring up the Customer listing

4.61 DetailCustomLookupButtonClick

For use with
Edit View Scripts

Format: Procedure DetailCustomLookupButtonClick(DisplayValue: Variant; AField: TField);

This Procedure is used in combination with Procedure DetailCustomLookup and allows you to define that a field value in the Detail tab is derived from a drop-down Lookup where the Lookup content is defined in the script.

In the following example we will assume that you already have created an Edit View and we are going to create a Custom Lookup against (say) field 'Customer'

Procedure DetailCustomLookup enbles you to nominate the Detail record field against which the lookup will appear. Add the following to the end of the Edit View Script.

```
Begin
  DetailCustomLookup('Customer');
End.
```

The next step is to declare the Custom Lookup under Procedure DetailCustomLookupButtonClick as follows:

```
procedure DetailCustomLookupButtonClick(DisplayValue: Variant; AField: TField);
var
  LookupResult: String;
begin
   if AField.Fieldname = 'Customer' then
  begin
    LookupResult := DisplayData('Select CUSTOMER from CUSTOMERMASTER', '
Customer Lookup', 'CUSTOMER');
  if trim(LookupResult) <> '' then
  begin
    AField.DataSet.Edit;
    AField.AsString := LookupResult;
  end;
 end;
end;
```

If you go into the Edit View you will see a 'spyglass' symbol in the Customer field which, when clicked, will bring up the Customer listing

4.62 DetailDomainCombo

For use with
Edit View Scripts

Format: DetailDomainCombo(FieldName, Domain);

This function allows you to create an entry field in the Detail record containing data that currently exists against Ostendo's Domains. The elements that make up this function are:

FieldName: The FieldName in the 'Detail' record. This should refer to the re-defined name in the Edit View Master Query and not the database field name.
Domain: The Domain Name.

This is an example of how you would declare the Values in a 'Detail' screen against domain CUSTOMER_STATUS

```
Begin
  DetailValuesCombo('Status','CUSTOMER_STATUS');
End.
```

4.63 DetailFocusedItemChanged

For use with
Edit View Scripts

Format: DetailFocusItemChanged(FocusedFieldName: String; PrevFocusedFieldName: String);

This procedure enables you to define what to do with related fields based upon the selection made in a nominated field in the Detail record. In this example we will look at the Resource Type field in the Resource Master Table (Asset or Employee) and that selection will dictate what drop-down appears in the next field in the Edit View.

```
Procedure DetailFocusedItemChanged(FocusedFieldName: String;
PrevFocusedFieldName: String);

begin
```

```
    if (FocusedFieldName = 'ResourceName’) then
    begin
     if Detailquery.fn('ResourceType').AsString = 'Asset' then
       Begin
        DetailLookup('ResourceName’,1053);
       end;
     if Detailquery.fn('ResourceType').AsString = 'Employee' then
       Begin
        DetailLookup('ResourceName’,1052);
       end;
    end;
end;
```

This states that if the cursor is placed in field **'ResourceName'** then look at the current entry in field **'ResourceType'** and show the relevant drop-down

4.64 DetailLookup

For use with
Edit View Scripts

Format: DetailLookup(FieldName, Lookup Number);

This function allows you to create an entry field whose data is populated from another table in Ostendo using the Lookup Reference Number defined for the Table. The elements that make up this Function are:

FieldName: The FieldName in the ‘Detail’ record. This should refer to the re-defined name in the Edit View Master Query and not the database field name.
Lookup Number: The Lookup Numbers as defined in the Scripting Help that refers to the relevant Tables

This is an example of how you would declare the Values in a ‘Detail’ screen against field (say) Customer_Type

```
Begin
  DetailLookup('Customer_Type',1028);
End.
```

4.65 DetailNewRecord

For use with
Edit View Scripts

Format: DetailNewRecord(DataSet: TpFIBDataSet);

This procedure enables you to prefill a field in the Detail record whenever a new record is being added. Of course you could amend this in the record unless it has been nominated as a ‘Read Only’ field. You should define the field in the ‘Detail’ record and the value that you wish to pre-populate it with. For example:

```
procedure DetailNewRecord(DataSet: TpFIBDataSet);
begin
  DataSet.FN('CustomerStatus').AsString := 'Active';
end;
```

4.66 DetailReadOnly

For use with
Edit View Scripts

Format: DetailReadOnly(FieldName,True);

This function allows you to define that a field is Read Only and cannot be amended. This is useful (a) when a field is copied from the main Ostendo Tables and cannot be amended in the Edit View, or (b) an Edit View has been created specifically for a User in which the data can be seen but not amended

The elements that make up this function are:

FieldName: The FieldName in the 'Detail' record. This should refer to the re-defined name in the Edit View Master Query and not the database field name.

True: If this is 'True' then this field cannot be amended in the Edit View. If set to 'False' then it can be amended.

This is an example of how you would define that the Detail field covering a Customer Address cannot be amended

```
Begin
  DetailRead Only('CUSTOMERADDRESS1',True);
End.
```

4.67 DetailValidate

For use with
Edit View Scripts

Format: DetailValidate(DisplayValue: Variant; AField: TField);

This procedure enables you to populate other Detail Fields based upon the selection made in a nominated field

This procedure allows you to carry out actions against a 'Detail' Level field based upon the entry in the nominated field. In the following example the nominated field in the 'Detail' record is linked to a Company Asset in Ostendo. Having selected the Company Asset this procedure will populate the Status, Description, and Type fields of the Edit View with data from the Asset Master based upon the selection made in the 'Name' field

```
procedure DetailValidate(DisplayValue: Variant; AField: TField);
begin
  if (AField.fieldname = 'Name') then
   begin
    Detailquery.fn('Status').AsString := GetSQLResult('Select ResourceStatus from
ResourceMaster where (ResourceName = ''' + Afield.fieldname + ''') and (AssetType =
''Asset'')');
    Detailquery.fn('Description').AsString := GetSQLResult('Select AssetDescription from
ResourceMaster where (ResourceName = ''' + Afield.fieldname + ''') and (AssetType =
''Asset'')');
    Detailquery.fn('Type').AsString := GetSQLResult('Select AssetType from
ResourceMaster where (ResourceName = ''' + Afield.fieldname + ''') and (AssetType =
''Asset'')');
   end;
end;
```

4.68 DetailValuesCombo

For use with

Edit View Scripts

Format: DetailValuesCombo(FieldName, Values, True);

This function allows you to create an entry field containing pre-defined data which is selected from a drop-down list.   The elements that make up this function are:

FieldName: The FieldName in the 'Detail' record. This should refer to the re-defined name in the Edit View Master Query and not the database field name.
Values: The Values that are to appear in the drop-down list; separated by a comma
True: If 'True' then only the defined Values can be selected. If False then the values are still displayed for selection but you may type in your own if required

This is an example of how you would declare the Values in a 'Detail' screen against field (say) Asset_Type

```
Begin
  DetailValuesCombo('Asset_Type','Plant,Vehicles,Tools',True);
End.
```

4.69 DirectoryExists

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script
CustomDataScreens
DataEntryScript

Format: DirectoryExists(Directory);

This will interrogate your network to determine if the specified Direcotry exists and returns a True/False message

Directory: The full path of the Directory being sought

In this example we will check if the Ostendo Directory exists

```
 Var
  DirectoryIsThere: Boolean;
Begin
        DirectoryIsThere := DirectoryExists('C:\Program Files\Ostendo');
        Showmessage(VarToStr(DirectoryIsThere));
End.
```

4.70 DisplayData

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format:
DisplayData(SQL, Title, ResultField, FormHeight, FormWidth);

This allows you to create an inquiry screen to list selected records extracted via an SQL. From the resultant list (which has additional search facilities included) you can select any record and return a specific field value for further action

The elements that make up this function are:

SQL: A standard Query enclosed in single quotes
Title: The title that will appear in the displayed panel
Result Field: The field from a selected record that returns the value
FormHeight: Height of the displayed panel in pixels
FormWidth: Width of the displayed panel in pixels

In this example all Labour Codes are extracted and displayed, from which a specific Labour Code is selected. This will be held against Variable ‘SelectedLabourCode’. In this example we have included an additional line to run another function that will update the Charge Rate of that Labour Code.

```
// Define the Variable to be populated by the selected Labour Code
Var
SelectedLabourCode: String;
// You can then ask the question to answer 'SelectedLabourCode' using
Begin
  SelectedLabourCode :=  DisplayData('Select * from LabourMaster','Labour Codes',
'LabourCode',500,1200);
  ExecuteSQL('update LabourMaster set StdSellRate = 55 where LabourCode = ''' +
SelectedLabourCode + '''');
  Showmessage('Labour Code ' + SelectedLabourCode + ' updated');
End.
```

4.71 DownloadFileFTP

For use with
General Custom Scripts

Format: DownloadFileFTP(Host, User, Password, SourceFileName, DestFileName, CanOverwrite, PassiveMode{Opt Default=False}, Port{Opt Default=21});

This is used to download a file from a remote FTP folder.

Related script functions are ***FTPList*** and ***DeleteFileFTP***.

Example:

```
Const

TheHost = 'ftp.test.info' ;  // enter your ftp host name here
TheUser = 'testuser' ;  // enter your ftp username here
Thepassword = 'testpassword' ;  // enter your ftp password here


var
SourceFile, DestFile  : string ;
```

```
begin
try
  sourcefile := 'public_html/testfile.txt' ;  // enter your source file path here
  destfile := OstendoPath+'/testfile.txt' ;  // enter your destination file path here

  DownloadFileFTP
(TheHost,TheUser,Thepassword,sourceFile,destFile,true,false,21) ;

  Run(destfile) ;

except
   showmessage(ExceptionMessage) ;
end;

end.
```

4.72 EndProgress

For use with

General Custom Scripts
Screen Data Script

Format: EndProgress;

This terminates the display of the Progress Bar.   This is used in combination with functions ***ShowProgress*** and ***UpdateProgress*** (and optionally ***UpdateProgressCaption***). The following example uses the four available functions related to the Progress Bar

```
Const
  ProgressCount = 2000;
Var
  x: Integer;
begin
  ShowProgress('My Progress Bar',ProgressCount);
   For x := 1 to progressCount do
   Begin
    if x >= (ProgressCount / 2) then
     Begin
      UpdateProgressCaption('Getting There');
     end;
    UpdateProgress(x);
   end;
  EndProgress;
  Showmessage ('Progress Display Completed');
End.
```

4.73 ExecuteSQL

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: ExecuteSQL(SQLStatement);

This allows you to run a Query

The elements that make up this function are:
SQL Statement: The Query, which can include Select, Update, or Delete functions

This example copies the first 30 characters of the Customer Name in the 'Customer' Table to the AdditionalField_1 in the same Table

```
Begin
   Executesql('update customermaster set ADDITIONALFIELD_1 = Substring(Customer
from 1 for 30)');
   Showmessage('Customer Names copied');
End.
```

4.74 ExportData

For use with
- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: ExportData(SQL, ExportType);

This allows you to export data selected via a Query. The export is added to the your Documents and Settings folder as csvfile.csv

The elements that make up this function are:
SQL Statement: The Query that would use the Select function
ExportType: 'HTML','XLS','XML','CSV'

```
var
 TheFileName: String;
Begin
  TheFileName := ExportData('select Customer,CustomerType from Customermaster',
'CSV');
  Showmessage('Data Exported to ' + TheFileName);
End.
```

4.75 FileExists

For use with
- ***System Action Script***
- ***Standard Script***
- ***Related Script***
- ***Screen Data Script***
- ***Order Script***
- ***Accounting Link Script***
- ***Custom Data Script***
- ***API Script***
- ***Edit View Script***
- ***Web Service Script***

Format: FileExists(FileName)

This will interrogate your network to determine if the specified Directory exists and returns a True/False message

FileName: The full path of the File being sought

In this example we will check if file Ostendo.exe exists

```
Var
  FileIsThere: Boolean;
Begin
  FileIsThere := FileExists('C:\Program Files\Ostendo\Ostendo.exe');
  Showmessage(VarToStr(FileIsThere));
End.
```

4.76 FinishQuestions

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: FinishQuestions;

This is used to clear the current display of question,   As an example, during a Custom Product configuration you may wish to first define a 'Model Style' after which the screen is cleared and the questions related to that Model Style are presented.

The following example asks two questions then, upon the answer to a third question will clear the display of not. You should note that the values against Variables answered in the first two questions are retained during the whole script run

```
// Define the Variables
Var
   FirstName: String;
   LastName: String;
   ClearQuestion: String;
   YourAge: Integer;
// You can then ask the questions regarding the name
  Begin
    FirstName := AskQuestion('What is your first name','TEXT','','');
    LastName := AskQuestion('What is your last name','TEXT','','');
    ClearQuestion := AskQuestion('Do you wish to clear the previous questions','TEXT','',
'Yes,No');
    If (ClearQuestion = 'Yes') then
      begin
       {The following function clears the Question Grid}
        FinishQuestions;
      end;
    {Now we continue with the question}
    YourAge := AskQuestion('What is your age','INTEGER','','');

Showmessage(FirstName + ' ' + LastName + ' you are ' + IntToStr(YourAge) + ' years of
age');
```

```
End.
```

4.77 FTPList

For use with
General Custom Scripts

Format: **FTPList(Host: String; User: String; Password: String; RemoteFolder: String; PassiveMode: boolean = False; Port: Integer = 21): Strings;**

This is used to list all the files in a remote FTP folder. Useful when you wish to download some files from an FTP site.

Related script functions are ***DownloadFileFTP*** and ***DeleteFileFTP***.

Example:

```
Const

TheHost = 'ftp.test.info' ;  // enter your ftp host name here
TheUser = 'testuser' ;  // enter your ftp username here
Thepassword = 'testpassword' ;  // enter your ftp password here


var
FileList  : TStringList ;
TheFolder : string ;
TheFilename : string;

begin
try
  FileList        := TStringList.Create;
  TheFolder :=  'public_html/testfolder' ;  // enter your ftp folder name here

  try
    FileList.Text := FTPList(TheHost,TheUser,ThePassword,TheFolder,false,21) ;

    showmessage(FileList.Text) ;

  except
    showmessage(exceptionmessage) ;
  end;
finally
  FileList.Free ;

end;
end.
```

4.78 GetBooleanFromTable

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := GetBooleanFromTable('TableName','FieldName','KeyField','KeyValue');

This will access the Ostendo Database and get the Boolean Value of a field in a nominated record. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
TableName: The name of the Ostendo Table
FieldName: The Field Name within the Table containing the Boolean Value
KeyField: The Field against which you are identifying the specific record
KeyValue: The Specific record identity within the Key Field

The following example check a specific Item in the Item Master table to see if a Warranty record is to be generated when it is sold. This example will use the AskQuestionWithLookup function to get the Item Code.

```
// Define the Variable to be populated by the result
Var
SelectedItem: String;
PartYesNo1: Boolean;
// You then construct the enquiry to answer ‘PartYesNo1’
Begin
  SelectedItem := AskQuestionWithLookup(‘Item Code’,’Please select the Item Code’,
1004,);
  PartYesNo1 := GetBooleanFromTable ('ItemMaster','WarrantyApplies','ItemCode'
,SelectedItem);
  Showmessage(PartYesNo1);
End.
```

4.79 GetCompanyName

This Function enables you to pull back the Company Name from the encrypted Licence.

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := GetCompanyName;

Variable: The defined variable against which the result will be held.

This example returns the Company Name from your Ostendo Licence

```
Var
  TheName: String;
begin
  TheName:= GetCompanyName;
  Showmessage(TheName);
```

```
        end.
```

4.80 GetCost

For use with
- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: Variable := GetCost('CodeType','Code');

This will access the Ostendo Database and get the Cost of the selected Item, Descriptor, or Labour Code. The Cost for Items and Descriptors is related to the Cost Method defined in System Settings

The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Code Type: Descriptor, Item or Labour
Code: The actual Code of the Descriptor, Item or Labour

The following example will get the cost of the Item and will display it for your information

```
// Define the Variable
Var
 PartCost1: Double;
// You can then get the cost using
  Begin
          PartCost1 := GetCost('Item','100-2000');
          Showmessage('Item Cost is $' + floattostr(PartCost1));
  End.
```

4.81 GetCurrencyFormat

For use with
- ***System Action Script***
- ***Standard Script***
- ***Related Script***
- ***Screen Data Script***
- ***Order Script***
- ***Accounting Link Script***
- ***Custom Data Script***
- ***API Script***
- ***Edit View Script***
- ***Web Service Script***

Format: GetCurrencyFormat

This will interrogate the CURRENCY Table set up in Ostendo and return:
- The Currency Symbol identified against the Currency record
- The Regional Settings for Decimal and Thousands separator
- The Number of decimal places set in Ostendo's System Configuration screen

```
Var
  CurrencyFormat: String;
```

```
Begin
  CurrencyFormat := GetCurrencyFormat('STL');
  Showmessage(CurrencyFormat);
End.
```

4.82 GetCustomerSellPrice

For use with

Custom Product Script

Format: Variable := GetCustomerSellPrice('CodeType','Code');

This is used within the Custom Product area to derive a Sell Price taking into account the Price Level and Quantity Breaks offered the Customer during Custom creation in Sales or Job Orders. It will access the Ostendo Database and get the Sell Price of the selected Item, Descriptor, or Labour linked to the Customer defined in the Order. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Code Type: Descriptor, Item or Labour
Code: The actual Code of the Descriptor, Item or Labour

The following example will get the price of the Item and display it for your information. You should first create the Custom Product Item (Example: DOOR') and then add the Item to a Sales Order. The script will then be run within the Order

```
// Define the Variable
Var
 SellPrice1: Double;
// You can then get the Price using
  Begin
        SellPrice1 := GetCustomerSellPrice('Item','DOOR');
        Showmessage('Item Sell Price is $' + floattostr(SellPrice1));
  End.
```

4.83 GetDateFromTable

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := GetDateFromTable('TableName','FieldName','KeyField','KeyValue');

This will access the Ostendo Database and get the date currently held in a field in a nominated record. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
TableName: The name of the Ostendo Table
FieldName: The Field Name within the Table containing the date
KeyField: The Field against which you are identifying the specific record
KeyValue: The Specific record identity within the Key Field

This will access the Ostendo Database and get the 'Last Cost Date' from a specified Item record

```
// Define the Variable
Var
```

```
  TheLastCostDate: TDate;
// You can then get the date using
Begin
  TheLastCostDate :=    GetDateFromTable('ItemMaster','Lastcostdate','Itemcode',
'100-2000');
  Showmessage('The Last Cost Date is ' + datetostr(TheLastCostDate));
End.
```

4.84 GetDoubleFromTable

For use with

- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: Variable := GetDoubleFromTable('TableName','FieldName','KeyField','KeyValue');

This will access the Ostendo Database and get the numeric value including decimals of a field in a nominated record. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
TableName: The name of the Ostendo Table
FieldName: The Field Name within the Table containing the number
KeyField: The Field against which you are identifying the specific record
KeyValue: The Specific record identity within the Key Field

This example will access the Ostendo Database and get the 'Stock on Hand' quantity from a specified Item record

```
// Define the Variable
Var
 StockOnHand: Double;
// You can then get the quantity using
  Begin
        StockOnHand := GetDoubleFromTable('ItemMaster','OnHandQty','Itemcode',
        '100-2000');
        Showmessage('Stock on hand is ' + floattostr(StockOnHand));
  End.
```

4.85 GetEmailAttachmentCount

For use with

- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: GetEmailAttachmentCount(MsgIndex);

The element that make up this function is:

MsgIndex: The evaluated number of attachments to an email.

This is used in conjunction with functions ***ReceiveEmail*** and ***GetEmailMessage*** to determine the number of attachments related to each email. Having determined the attachment count we can then download them to a defined location.

This exercise includes the ***ReceiveEmail***, ***GetEmailMessage*** and ***GetEmailAttachmentCount*** functions to demonstrate how to download and extract email information

```
Const
  Host = 'pop3.myemailhost.com';
  User = 'YourUserName';
  Password = 'yourpassword';
Var
  MessageCount,AttachmentCount,x,y: Integer;
  MessageBody, MessageFrom,MessageSubject,AttachmentFileName: String;
begin
  MessageCount := ReceiveEmail(Host,User,Password);
  Showmessage('Number of Messages is ' + inttostr(MessageCount));
  For x := 1 to MessageCount do
   Begin
    MessageSubject := GetEmailMessage('SUBJECT',x);
    MessageFrom := GetEmailMessage('FROMADDRESS',x);
    MessageBody := GetEmailMessage('BODY',x);
    Showmessage('The contents for email ' + inttostr(x) + ' is:' + #13 +
                'Subject: ' + MessageSubject + #13 +
                'From: ' + MessageFrom + #13 +
                'Body: ' + MessageBody);
    AttachmentCount := GetEmailAttachmentCount(x);
    For y := 1 to AttachmentCount do
     Begin
      AttachmentFileName := GetEmailMessage('ATTACHMENT',x,y);
      Showmessage(AttachmentFileName);
     end;
   end;
end.
```

4.86 GetEmailMessage

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: GetEmailMessage(MsgPart, MsgIndex, AttachIndex);

This will access a specific message previously downloaded via function ***ReceiveEmail*** and extract data from various elements that make up that email. The elements that make up this function are:

MsgPart: The part of the email that you are extracting. The options are:
FROMNAME - Name of the Sender
SUBJECT - The content of the message Header
BODY - The content of the message body
ATTACHMENT - the Attachment (Attachment Index also required)
BODYHTML - The content of the message body in html format
FROMADDRESS - The senders address

MsgIndex: Message Number derived from function ***ReceiveEmail***
AttachIndex: Attachment Index derived from function ***GetEmailAttachmentCount***

The email receipt process comprises of:

- **ReceiveEmail**; This receives the email(s) and gives each email a Message Number
- **GetEmailMessage**: This allows you to reference a specific Message Number and extract data from that message
- **GetEmailAttachmentCount**: This counts the number of attachment to a referenced Message Number

This exercise repeats the ***ReceiveEmail*** function and extends it so that you can extract data from specific emails

It is suggested that you first send a couple of emails using the ***sendmailmessage*** function and then run this routine which includes the ***ReceiveEmail*** and ***GetEmailAttachment*** functions.

Create a new Custom Script with the following. Amend the '**Const**' values to suit your environment

```
Const
 Host = 'pop3.myemailhost.com';
 User = 'YourUserName';
 Password = 'yourpassword';
Var
 MessageCount,AttachmentCount,x,y: Integer;
 MessageBody, MessageFrom,MessageSubject,AttachmentFileName: String;
begin
 MessageCount := ReceiveEmail(Host,User,Password);
 Showmessage('Number of Messages is ' + inttostr(MessageCount));
 For x := 1 to MessageCount do
  Begin
   MessageSubject := GetEmailMessage('SUBJECT',x);
   MessageFrom := GetEmailMessage('FROMADDRESS',x);
   MessageBody := GetEmailMessage('BODY',x);
   Showmessage('The contents for email ' + inttostr(x) + ' is:' + #13 +
         'Subject: ' + MessageSubject + #13 +
         'From: ' + MessageFrom + #13 +
         'Body: ' + MessageBody);
   AttachmentCount := GetEmailAttachmentCount(x);
   For y := 1 to AttachmentCount do
    Begin
     AttachmentFileName := GetEmailMessage('ATTACHMENT',x,y);
     Showmessage(AttachmentFileName);
    end;
  end;
end.
```

4.87 GetFieldNames

For use with

System Action Script
Standard Script
Related Script
Screen Data Script
Order Script
Accounting Link Script
Custom Data Script
API Script

Edit View Script
Web Service Script

Format: GetFieldNames(TableName)

This will return a string list containing all fields in the specified Table

TableName: The Table against which the Fields are being extracted

In this example we will create a string list from the ITEMMASTER and display the result using the Showmessage function

```
Var
  FieldNames: String;
Begin
  FieldNames := getfieldnames('ITEMMASTER');
    Showmessage(FieldNames);
End.
```

4.88 GetFileList

For use with
- *General Custom Scripts*
- *Screen Data Script*
- *Order Script*
- *Custom Product Script*
- *CustomDataScreens*
- *DataEntryScript*

Format: **GetFileList(Directory,FileExtension);**

This will go to the specified Directory and return a list of all the Files with the specified extension. The files in the string are separated by a ‘Carriage Return’ and is therefore ideal for display in a stringlist.

The elements that make up this function are

Directory: The full path of the Directory being interrogated
FileExtension: This is the file extension that you wish to be returned. You should specify this without the dot segregator. For example, txt and not .txt. If you wish to return all extensions then use an asterisk.

In this example we will get a list of fr3 files from the Ostendo Reports Directory.

```
 Var
  MyStringList: String;
Begin
        MyStringList := GetFileList( 'C:\Program Files\Ostendo\Reports\' , 'fr3' );
        Showmessage(MyStringList);
End.
```

4.89 GetGenerator

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: **GetGenerator(Generator);**

This will access Ostendo Number Generator table and obtain the next sequential number for the relevant Table Name. You should note that this function also increments the number in preparation for the next number allocation. The elements that make up this function are:

Generator: The name of the record against which the number is held.

This example will go to the Job Issue Generator and get the next Job Issue Batch Number

```
Var
 TheJobIssueNo: Integer;
// You can then get the next number using
Begin
 TheJobIssueNo := getgenerator('JOBISSUENO');
 showmessage('The Next Job Issue Number is ' + inttostr(TheJobIssueNo));
End.
```

List of Generators:

ASSEMBLYORDERNO
ASSIGNMENTID
ASSYISSUENO
ASSYRECEIPTNO
ASSYTIMESHEETNO
AUDITLOGNO
BANKINGNO
BATCHFILENO
BUYPRICECONTRACTNO
CALLTICKETID
CONNECTIONNO
COSTINGBATCHNO
CURRENCYNUMBER
CUSTOMERDEPOSITNO
CUSTOMERPAYMENTNO
GOODSDOCKETNO
INVENTORYADJUSTNO
INVENTORYBATCHNO
INVENTORYCOUNTNO
INVENTORYRESTOCKNO
INVENTORYTRANSFERNO
INVOICECONTRACTNO
ITEMCODENO

JOBISSUENO
JOBORDERNO
JOBTIMESHEETNO
JOURNALNO
KBARTICLEID
PRICEINQUIRYNO
PRICINGBATCHNO
PRICINGCONTRACTNO
PRICINGFILENO
PURCHASEINVOICENO
PURCHASEORDERNO
PURCHASERECEIPTNO
PURCHASESHIPMENTNO
REPORTARCHIVENO
SALESDELIVERYNO
SALESINVOICENO
SALESORDERNO
SALESPOSENDOFDAY
SALESPOSNO
SPECIALPRICINGNO
SUPPLIERCATALOGUENO
SUPPLIERCONTRACTNO
SYSUNIQUEID
TIMESHEETNO
WARRANTYID

4.90 GetIntegerFromTable

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := GetIntegerFromTable('TableName','FieldName','KeyField','KeyValue');

This will access the Ostendo Database and get the Integer Value of a field in a nominated record. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
TableName: The name of the Ostendo Table
FieldName: The Field Name within the Table containing the number
KeyField: The Field against which you are identifying the specific record
KeyValue: The Specific record identity within the Key Field

This will access the Ostendo Database and get the 'Lead Time' from a specified Item record

```
// Define the Variable
Var
 TheLeadTime: Integer;
// You can then get the value using
```

```
Begin
     TheLeadTime := GetIntegerFromTable('ItemMaster','LeadTime','Itemcode',
     '100-2000');
     Showmessage('The Leadtime is ' + inttostr(TheLeadTime) + ' day');
End.
```

4.91 GetOstendoCompanies

For use with
- *System Action Script*
- *Standard Script*
- *Related Script*
- *Screen Data Script*
- *Order Script*
- *Accounting Link Script*
- *Custom Data Script*
- *API Script*
- *Edit View Script*
- *Web Service Script*

Format: GetOstendoCompanies

This will return a string List of all Ostendo Company Names and Database Locations currently registered within Ostendo.

```
Var
  Companies: String;
Begin
  Companies:= GetOstendoCompanies;
  Showmessage(Companies);
End.
```

4.92 GetSourceFieldValue

For use with
- *Order Script*
- *Related Menu Script*
- *Screen Data Script*

Format: GetSourceFieldValue(FieldName,HEADER (or LINE));

This is used specifically against Orders (Sales, Jobs, Purchase, or Assembly). It places a Button on the Order Line Batch Entry Bar which when selected returns the value in the field defined here

FieldName: The Name of a field in the source record
HeaderOrLine: This defines if the source is the Order HEADER or LINE.

In this exercise we will create a script that will get the current total order value and display this whilst still in the Order Lines Screen.

To perpare and run this exercise carry out the following steps

Go to ***File>Custom Scripts*** and add a new Custom Script called (say) **OrdPrice** and 'check' the '**This is anOrder script**' checkbox. Click on the 'Script' tab and add the following script

```
var
```

```
 OrdAmount: String;
begin
 OrdAmount := GetSourceFieldValue('ORIGINALORDERAMOUNT','HEADER');
 showmessage('The Order Amount is $' + OrdAmount);
end.
```

Save and exit the Custom Script screen.

The next step is to tell Ostendo that the script is linked to a Sales Order. To do this go into ***File>System Configuration>Order Scripts*** and create a new record containing the following

Screen: Select '**Sales Orders**' from the drop-down
Script Name: Select the above script Name

Save and exit

Now go into ***Sales>Sales Orders*** and create a Sales Order and add a couple of lines. While you are in the Lines tab you will see a Button '**OrdPrice**' on the Batch Entry Bar. If you click on this button the current Order Value will be returned

4.93 GetSQLResult

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: GetSQLResult(SQL Statement);

This will allow you to run a Query and then populate the result in a Variable. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
SQL Statement: The Query

In this example we will access the Item Master and carry out a count of the number of records

```
// Define the Variable to be populated by the selected Query result
Var
 QueryResult: String;
// You then specify the Query using
Begin
   QueryResult := GetSQLResult('Select Count(*) from ItemMaster');
   Showmessage('There are ' + QueryResult + ' Item Records');
End.
```

4.94 GetStdBuyPrice

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := GetStdBuyPrice('CodeType','Code');

This will access the Ostendo Database and get the Standard Buy Price of the selected Item, Descriptor, or Labour. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Code Type: Descriptor, Item or Labour
Code: The actual Code of the Descriptor, Item or Labour

In this example we will access the Item Master and extract the Standard Buy Price against Item 100-2000

```
// Define the Variable
Var
 PartBuy1: Double;
// You can then get the Buy Price using
  Begin
        PartBuy1 := GetStdBuyPrice('Item','100-2000');
        Showmessage('Item Buy Price is $'' + floattostr(PartBuy1));
  End.
```

4.95 GetStdSellPrice

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := GetStdSellPrice('CodeType','Code');

This will access the Ostendo Database and get the Standard Sell Price of the selected Item, Descriptor, or Labour. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
Code Type: Descriptor, Item or Labour
Code: The actual Code of the Descriptor, Item or Labour

In this example we will access the Descriptor Master and extract the Standard Sell Price against Descriptor GENERALTIME

```
// Define the Variable
Var
 PartSell1: Double;
// You can then get the Standard Sell Price using
  Begin
        PartSell1 := GetStdSellPrice('Descriptor','GENERALTIME');
        Showmessage('Descriptors Standard Sell Price is $' + floattostr(PartSell1));
  End.
```

4.96 GetStringFromTable

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := GetStringFromTable('TableName','FieldName','KeyField','KeyValue');

This will access the Ostendo Database and get the contents of a field in a nominated record. The elements that make up this function are:

Variable: The defined variable against which the result will be held.
TableName: The name of the Ostendo Table containing the string
FieldName: The Field Name within the Table containing the string
KeyField: The Field against which you are identifying the specific record
KeyValue: The Specific record identity within the Key Field

In this example we will access the Item Master and extract the Primary Supplier against Item 5000-1010

```
// Define the Variable
Var
   PrimSupp: String;
// You can then get the PrimarySupplier using
Begin
 PrimSupp := GetStringFromTable('ITEMMASTER','PRIMARYSUPPLIER','ITEMCODE',
'5000-1010');
 Showmessage('Primary Supplier is ' + PrimSupp);
End.
```

4.97 GetTableNames

For use with

System Action Script
Standard Script
Related Script
Screen Data Script
Order Script
Accounting Link Script
Custom Data Script
API Script
Edit View Script
Web Service Script

Format: GetTableNames(IncludeSystem)

This will return a string list containing all Ostendo Tables. The following example returns the String List using a 'Showmessage' function

```
Var
  TableNames: String;
Begin
  TableNames := gettablenames;
  Showmessage(TableNames);
End.
```

4.98 GetValueFromStore

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

This function retrieves values that have been previously stored in memory via the SaveValueToStore function. The elements that make up this function are:

Name: The Name of the String that is currently in memory

This process will use the ***SaveValueToStore*** function to store the value in memory after which we will use this function to retrieve and display it

```
Var
 TheStoredValue: String;
Begin
  SaveValueToStore('FirstName=Fred');
  TheStoredValue:= GetValueFromStore('FirstName');
  Showmessage(TheStoredValue);
End.
```

4.99 GetWorkflowObjectTag

For use with

General Custom Scripts linked to Workflows

Format: GetWorkflowObjectTag(ObjectID)

ObjectID: Right-Mouse on the Object in the Workflow to get the ObjectID.

This function gets the Tag Reference Number contained in the Object

This example will return the Tag Reference Number of the Object. Add an Object to a Workflow and – in the Inspector panel - enter a numeric value in the Object's 'Tag' field. Create the following script in Ostendo and link the Object to it

```
Var
Tag: Integer;
begin
  Tag:= GetWorkflowObjectTag(4);
  Showmessage('Objects Tag Reference is ' + IntToStr(Tag));
end.
```

If you now click on the Object in the Workflow it will return the Object's Tag reference.

4.100 GetWorkflowObjectText

For use with

General Custom Scripts linked to Workflows

Format: GetWorkflowObjectText(ObjectID)

ObjectID: Right-Mouse on the Object in the Workflow to get the ObjectID.

This function gets the Text held against the Object

This example will return the Text currently in an Object. Add a Text Object to a Workflow and amend the Text as required. Create the following script in Ostendo and link the Object to it

```
Var
TheText:  String;
begin
  TheText:= GetWorkflowObjectText(4);
  Showmessage('Objects Text is ' + TheText);
end.
```

If you now click on the Object in the Workflow it will return the Text contained in the Object.

4.101 InsertAssembly

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertAssembly(OrderNumber, OrderDate, RequiredDate, ItemCode, Qty, SourceType, SourceReference, SourceID, SourceName, CreateDefaultStep);

This will create an AssemblyHeader record. The elements that make up this function are:

OrderNumber: The Assembly Order Number. Note: If Order Numbering is automatic (Assembly Rules) then this should be left blank.
Order Date: The date the Order was raised. You can enter the word ‘date’ to denote the system date or specify strtodate(‘01/07/2008’) to convert an entered date (Format dd/mm/yyyy) into a date field
RequiredDate: The date the Order is required. You can enter the word date to denote the system date or specify strtodate(‘01/07/2008’) to convert an entered date (Format dd/mm/yyyy) into a date field
ItemCode: The parent Item for the Assembly Order. You should note that upon generation of the Assembly Order Ostendo will check to see if a BOM exists for this Item and, if so, add the components and routing from the BOM.
Qty: The quantity being ordered
Source Type: The source of the order (Example: Manual, Sales, Jobs)
Source Reference: The number of the Source Order. This can be blank for Manual Source Type
SourceID: Other source such as Drawing Number. Can be left blank
SourceName: Other source Name. Can be left blank
CreateDefaultStep: Set to True if you wish to add the default Assembly step to the Assembly Order. If set to False then you should also run function InsertAssemblyStep to add the relevant steps.

In this example we will create an order for 1105-2184.

```
Begin
      InsertAssembly('',date, strtodate('01/07/2008'),'1105-2184',1,'Manual','',0,'To
      Stock',False);
      Showmessage('Assembly Order Created');
End.
```

You should note that, in addition to creating the Assembly Order, you can also return the Order Number itself to a variable. This variable can then be used in other parts of the whole script. For example the above script can be extended as follows

```
Var
  TheOrderNumber: String;
```

```
Begin
   TheOrderNumber := InsertAssembly('',date, strtodate('01/07/2008'),'1105-2184',1,'
Manual','',0,'To Stock',False);
   Showmessage('The Assembly Order created is ' + TheOrderNumber);
End.
```

4.102 InsertAssemblyLine

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertAssemblyLine(OrderNumber, Qty, LineNumber, CodeType, LineCode, Description, StepName, LineUnit, PositionReference, RunorSetup, LineNotes);

This will create an Assembly Line record against an Assembly Order Header that has been previously generated. The elements that make up this function are:

OrderNumber: The Assembly Order Number.that must already exist
Qty: The quantity of the line that is being added
LineNumber: The Assembly Order Line Number.to be given to this line
CodeType: The type of line being added. The options are Item Code, Descriptor Code, or Labour Code.
LineCode: The identity of the line being added.
Description: Description of the line being added.
StepName: The Step Name that this line is being added to. The Step must already exist against the Assembly Order.
LineUnit: The Unit of Measure for the Line. This must be a valid Unit of Measure currently held against the LineCode
PositionReference: The position Reference (Example: Location on Drawing that the component appears in this order. Can be left blank.
RunOrSetup: Must have either Run or Setup as an entry
LineNotes: Any required notes. Can be left blank

In this example we will add a line to an existing order. Therefore you should first create the Order Header and add any Steps. The Lines can then be added using the following function (Replace WO200027 with the Order Header Number)

```
Begin
        InsertAssemblyLine('WO200027',3,60,'Item Code','100-2000','Washer-Mild
        Steel-8mm','Assembly','Each','','Run','');
        Showmessage('Assembly Line Added');
End.
```

You should note that you can add the Assembly Order Line at the same time as you create the Assembly Order Header by using a declared variable. For example we will combine the InsertAssembly and InsertAssemblyLine scripts as follows

```
Var
  TheOrderNumber: String;
Begin
  TheOrderNumber := InsertAssembly('',date, strtodate('01/07/2008'),'1105-2184',1,'
Manual','',0,'To Stock',False);
   InsertAssemblyLine(TheOrderNumber,3,60,'Item Code','100-2000','Washer-Mild
Steel-8mm','Assembly','Each','','Run','');
```

```
  Showmessage('The Assembly Order and Line created is ' + TheOrderNumber);
End.
```

4.103 InsertAssemblyOutput

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertAssemblyOutput(OrderNumber, OutputStyle, OutputCode, OutputDescription, OutputQty, OutputUnit, OutputCostPercent, OutputRecQty, ScrapQty);

When an Assembly Order is created the main Output Item is automatically included. This function will create additional output records covering Co-Products and Bi-Products. The elements that make up this function are:

OrderNumber: The Assembly Order Number.that must already exist
Output Style: This must be either CoProduct or BiProduct
OutputCode: Must be a valid Item Code
OutputDescription: Description of the Co-Product or Bi-Product.
OutputQty: The quantity of the line in the Assembly Order
OutputUnit: The base unit of the OutputCode
OutputCostPercentage: The percentage of the component costs that this Co-Product or Bi-Product will consume
OutputRecQty: The quantity already received against this line
ScrapQty: The quantity already scrapped against this line

In this example we will add a Co-Product to an existing order. Therefore you should first create the Order Header. The Co-Product can then be added using the following function. You should replace WO200027 with the actual Order Header Number)

```
Begin
        InsertAssemblyOutput('WO200027','CoProduct','100-2000','Washer-Mild
        Steel-8mm',2,'Each',40,0,0);
        Showmessage('Co-Product Line Added');
End.
```

You should note that you can add the Co-Product or Bi-Product at the same time as you create the Assembly Order Header by using a declared variable. In this example we will combine the InsertAssembly and InsertAssemblyOutput scripts as follows

```
Var
  TheOrderNumber: String;
Begin
   TheOrderNumber :=  InsertAssembly('',date, strtodate('01/07/2008'),'1105-2184',1,'
Manual','',0,'To Stock',False);
    InsertAssemblyOutput('WO200027','CoProduct','100-2000','Washer-Mild  Steel-8mm',2,'
Each',40,0,0);
    Showmessage('Assembly Order and Co-Product Line Added');
End.
```

4.104 InsertAssemblyStep

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertAssemblyStep(OrderNumber, StepName, StepSeq, StepInstructions);

This will create an Assembly Step record. The elements that make up this function are:

OrderNumber: The Assembly Order Number.
StepName: The name of the Step. This MUST already exist in ***Assembly>Settings>StepNames***
StepSequence: Any Integer
Instructions: Extended Instruction on the Step Process

In this example we will add a step to an existing Assembly Order. You should replace WO200027 with the actual order number.

```
Begin
        InsertAssemblyStep('WO200027','QA',20,'');
        Showmessage('Assembly Step Added');
End.
```

You should note that you can add the Step at the same time as you create the Assembly Order Header by using a declared variable. In this example we will combine the InsertAssembly and InsertStep scripts as follows

```
Var
  TheOrderNumber: String;
Begin
   TheOrderNumber := InsertAssembly('',date, strtodate('01/07/2008'),'1105-2184',1,'
Manual','',0,'To Stock',False);
   InsertAssemblyStep(TheOrderNumber,'QA',20,'');
   Showmessage('Assembly Order and additional Step Added');
End.
```

4.105 InsertBOMHeader

For use with

Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when creating the BOM Header for that configuration

Format: InsertBOMHeader('Description');

This is required to define that a BOM is being created. The elements that make up this function are:

Description: A description of the created BOM (max 50 chars). Blank if not specified

Example:

```
InsertBOMHeader('The BOM Header for this configuration');
```

4.106 InsertBOMLine

For use with

Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when adding a BOM Line to that configuration

Format:
InsertBOMLine('StepName','CodeType','LineCode',LineQty,LineNumber,ScrapPercent,'RunOrSetup','PosReference'','LineInstructions');

This will add a Line to a step in the Bill of Material. The elements that make up this function are:

StepName: The Step Name **-** The step should be created as defined above
CodeType: Must be Descriptor, Item or Labour
LineCode: The name of the Descriptor, Item or Labour (max 50 chars)
LineQty: The quantity required against this line
LineNumber: Any Integer to define a Line number
ScrapPercent: Any Number (incl decimals). Zero if it does not apply
RunOrSetup: Enter either Run or Setup
PosReference: Enter a position reference if applicable
LineInstructions: Unlimited amount of text to add Instructions

Example: InsertBOMLine('Cut','Item','RIMUPANEL1600X600',1,10,0,'Run','','');

4.107 InsertBOMProperty

For use with

Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when creating a BOM Property for that configuration

Format: InsertBOMProperty('PropertyName','PropertyValue');

If you wish to add Properties to the BOM then use this function. The elements that make up this function are:

PropertyName: The name of the Property (max 20 chars)
PropertyValue: The value being applied to the Property Name (max 50 chars)

Example:

InsertBOMProperty('Length',inttostr(DeskLength));

Note: The instruction inttostr(DeskLength converts a numeric variable to a string format

4.108 InsertBOMResource

For use with

Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when adding a BOM Resource for that configuration

Format: InsertBOMResource('StepName','ResourceType','ResourceName');

This will add a Resource to a process Step. The elements that make up this function are:
StepName: The Step Name - The step should be created as defined above
ResourceType: Must be either 'ASSET' or 'EMPLOYEE'
ResourceName: The name of the Asset or Employee (max 30 chars)

Example:
InsertBOMResource('Cut','Employee','John');

4.109 InsertBOMStep

For use with
Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when creating a BOM Step for that configuration

Format: InsertBOMStep('StepName',StepSequence,'StepDescription','StepInstructions');

This will create a Bill of Material Step. The elements that make up this function are:
StepName: The name that you are applying to the Step (max 20 chars)
StepSequence: A number defining the sequence in which the Step will be carried out
StepDescription: A brief description of the Step (max 50 chars)
StepInstructions: Unlimited Text defining what happens in this Step

Example:
InsertBOMStep('Cut',10,'Cut and Trim all Timber','Check the measurements');

4.110 InsertJob

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertJob(OrderNo, OrderDate, RequiredDate, JobType, Customer, Description, OrderNotes, CustomerPO, CustomerAsset, UseTemplate, Template, EstimatedDuration, DurationScale, ProjectName);

This will create a Job Header record. The elements that make up this function are:
OrderNumber: The Job Order Number. Note: If the Job Type's Order Numbering is automatic then this should be left blank.
Order Date: The date the Order was raised. You can enter the word 'date' to denote the system date or specify strtodate('01/07/2008') to convert an entered date (Format dd/mm/yyyy) into a date field
RequiredDate: The date the Order is required. You can enter the word 'date' to denote the system date or specify strtodate('01/07/2008') to convert an entered date (Format dd/mm/yyyy) into a date field
JobType: The Job Type as defined in Jobs>Settings>Job Types.
Customer: The Order Customer
Description: Description of the Order. Can be blank.
Order Notes: Extended Notes against the Order. Can be blank
Customer PO: The Customer's Purchase Order. This must have an entry if the Customer 's Master record is flagged as 'Purchase Order Mandatory'
CustomerAsset: The Identity of the Customer Asset is the Job Style of the Order Type is

defined as ‘Customer Asset’
UseTemplate: Set to True if you wish to create the Job Order using a Job Template..
Template: If the previous parameter is ‘True’ then a valid Template should be entered here
EstimatedDuration: The estimated duration that the Job will take. Can be zero
DurationScale: The scale of the duration. The options are ‘Minutes’ or ‘Hours’,
ProjectName: If the Job Order is linked to a Project then enter the Project ID here

In this example we will create a Job Order for Customer ‘Jim Gold & Co Ltd’

```
Begin
        InsertJob('',date, strtodate('01/07/2008'),'Progress','Jim Gold & Co Ltd','Phone
        Order','','','',False,'',0,'Hours');
        Showmessage('Job Order Created');
End.
```

You should note that, in addition to creating the Job Order Header, you can also return the Job Order Number itself to a variable. This variable can then be used in other parts of the whole script. For example the above script can be extended as follows

```
Var
  TheJobOrderNumber: String;
Begin
   TheJobOrderNumber := InsertJob('',date, strtodate('01/07/2008'),'Progress','Jim Gold &
Co Ltd','Phone Order','','','',False,'',0,'Hours');
   Showmessage('The Job Order created is ' + TheJobOrderNumber);
End.
```

4.111 InsertJobOrderLine

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertJobOrderLine(OrderNumber, CodeType, LineCode, LineQty, OverridePrice, UnitPrice);

This will create a Job Order Line record against a Job Order Header that has been previously generated. The elements that make up this function are:
OrderNumber: The Job Order Number.that must already exist
CodeType: The type of line being added. The options are Item Code, Descriptor Code, Labour Code, Kitset Code, or Catalogue Code.
LineCode: The identity of the line being added.
LineQty: The Order quantity of the line being added
OverridePrice: Enter ‘True’ if the next field has an override Price. Enter ‘False’ if you are using the Base Price
UnitPrice: If the previous field is ‘True’ then enter the override Price

In this example we will add a line to an existing order. Therefore you should first ensure that a Job Order Header exists (This should replace PRO400010 - below - with the actual Job Order number). The Lines can then be added using the following function

```
Begin
        InsertJobOrderLine('PRO400005','Item Code','100-2000',3,False,0);
        Showmessage('Job Line Added');
End.
```

You should note that you can add a Job Order Line at the same time as you create the Job Order Header by using a declared variable. For example we will combine the InsertJob and InsertJobOrderLine scripts as follows

```
Var
   TheJobOrderNumber: String;
Begin
    TheJobOrderNumber := InsertJob('',date, strtodate('01/07/2008'),'Progress','Jim Gold & Co Ltd','Phone Order','','','',False,'',0,'Hours');
    InsertJobOrderLine(TheJobOrderNumber,'Item Code','100-2000',3,False,0);
    Showmessage('Job Order and Line Added');
End.
```

4.112 InsertOrderLine

For use with
Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when creating an additional Sales Or Job Order Line for the configuration

Format: InsertOrderLine('CodeType','LineCode',LineQty,UnitPrice);

This function allows you to add new lines to the originating Sales or Job Order. This emulates an ' Add-On Sale' type function such as you also supply a 'Chair' when you configure a Desk. The elements that make up this function are:

CodeType: Must be Descriptor, Item or Labour
LineCode: The name of the Descriptor, Item or Labour (max 50 chars)
LineQty: The quantity required against this line (incl Decimals)
UnitPrice: The Unit Sell Price for this Line (incl Decimals)

Example:
```
InsertSOLine('Item','OFFICECHAIR',1,135);
```

4.113 InsertPurchaseOrder

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertPurchaseOrder(OrderNo, OrderType, OrderDate, RequiredDate, Supplier, Description, OrderNotes);

This will create a Purchase Order Header record. The elements that make up this function are:

OrderNumber: The Purchase Order Number. Note: If the Purchase Type's Order Numbering is automatic then this should be left blank.
OrderType: The Purchase Order Type as defined in Purchasing>Settings>Purchase Types.
Order Date: The date the Order was raised. You can enter the word 'date' to denote the

system date or specify strtodate('01/07/2008') to convert an entered date (Format dd/mm/yyyy) into a date field
RequiredDate: The date the Order is required. You can enter the word 'date' to denote the system date or specify strtodate('01/07/2008') to convert an entered date (Format dd/mm/yyyy) into a date field
Supplier: The Order Supplier
Description: Description of the Order. Can be blank.
Order Notes: Extended Notes against the Order. Can be blank

In this example we will create a Purchase Order against Supplier 'Bruce Wilson'

```
Begin
    InsertPurchaseOrder('','Standard',date,strtodate('01/07/2008'),'Bruce
    Wilson','Urgent Order','');
    Showmessage('Purchase Order Created');
End.
```

You should note that, in addition to creating the Purchase Order Header, you can also return the Purchase Order Number itself to a variable. This variable can then be used in other parts of the whole script. For example the above script can be extended as follows

```
Var
  ThePONumber: String;
Begin
    ThePONumber := InsertPurchaseOrder('','Standard',date,strtodate('01/07/2008'),'
    Bruce Wilson','Urgent Order','');
    Showmessage('The Purch Order created is ' + ThePONumber);
End.
```

4.114 InsertPurchaseOrderLine

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertPurchaseOrderLine(OrderNumber, CodeType, LineCode, LineQty, OverridePrice, UnitPrice);

This will create a Purchase Order Line record against a Purchase Order Header that has been previously generated. The elements that make up this function are:
OrderNumber: The Purchase Order Number.that must already exist
CodeType: The type of line being added. The options are Item Code, Descriptor Code, or Catalogue Code.
LineCode: The identity of the line being added.
LineQty: The order quantity of the line being added
OverridePrice: Enter 'True' if the next field has an override Price. Enter 'False' if you are using the Base Price
UnitPrice: If the previous field is 'True' then enter the override Price

In this example we will add a line to an existing order. Therefore you should first ensure that a Purchase Order Header exists. Replace PO100016 with the actual Purchase Order Number. The Lines can then be added using the following function

```
Begin
    InsertPurchaseOrderLine('PO100017','Item Code','100-2000',3,False,0);
```

```
        Showmessage('Purchase Line Added');
End.
```

You should note that you can add a Purchase order Line at the same time as you create the Purchase Order Header by using a declared variable. For example we will combine the InsertPurchaseOrder and InsertPurchaseOrderLine scripts as follows

```
Var
  ThePONumber: String;
Begin
        ThePONumber := InsertPurchaseOrder('','Standard',date,strtodate('01/07/2008'),'
        Bruce Wilson','Urgent Order','');
        InsertPurchaseOrderLine(ThePONumber,'Item Code','100-2000',3,False,0);
        Showmessage('Purchase Order and Line Added');
End.
```

4.115 InsertRecord

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertRecord(TableName, Mappings, MultiLineValues);

This will create one or more lines in any table in Ostendo. The elements that make up this function are:

TableName: The table into which the record is being inserted
Mappings:: You should identify the mappings of what is to be inserted. This can be in one of two formats dependent upon what is specified in the next parameter.
If the next parameter is '**False**' (or not specified) then the mappings are in the form of a string value using #13 after each field. (The function recognises #13 as a 'Carriage Return')

For example Field1 + #13 + Field2 + #13 + Field3

If the next parameter is '**True**' then the mappings are in the form of a list

```
For example     Field1
                Field2
                Field3
```

MultiLineValues:: If False or Blank then the supplied data is in the form of a string value. If True then the supplied data is in a list where each field is on a new line.

In this example we will add a Scrap Code to the Assembly Scrap Code Table

```
Begin
      InsertRecord('ScrapCodes',
      'ScrapCode=Broken' + #13 +
      'ScrapDescription=Item Broken' + #13 +
      'CostCentre=');
      Showmessage('Scrap Code Added');
End.
```

4.116 InsertSalesOrder

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertSalesOrder(OrderNo, OrderType, OrderDate, RequiredDate, Customer, Description, PurchaseOrder, OrderNotes);

This will create a Sales Order Header record. The elements that make up this function are:

OrderNumber: The Sales Order Number. Note: If the Sales Type's Order Numbering is automatic then this should be left blank.
OrderType: The Sales Type as defined in Sales>Settings>Sales Types.
Order Date: The date the Order was raised. You can enter the word 'date' to denote the system date or specify strtodate('01/07/2008') to convert an entered date (Format dd/mm/yyyy) into a date field
RequiredDate: The date the Order is required. You can enter the word 'date' to denote the system date or specify strtodate('01/07/2008') to convert an entered date (Format dd/mm/yyyy) into a date field
Customer: The Order Customer
Description: Description of the Order. Can be blank.
PurchaseOrder: The Customer's Purchase Order. This must have an entry if the Customer's Master record is flagged as 'Purchase Order Mandatory'
OrderNotes: Extended Notes against the Sales Order Header. This can be blank

In this example we will create a Sales Order for Customer 'Jim Gold & Co Ltd'

```
Begin
     InsertSalesOrder('','CounterSales',date, strtodate('01/07/2008'),'Jim Gold & Co
     Ltd','Fax Order','','');
     Showmessage('Sales Order Created');
End.
```

You should note that, in addition to creating the Sales Order Header, you can also return the Sales Order Number itself to a variable. This variable can then be used in other parts of the script. For example the above script can be extended as follows

```
Var
  TheSONumber: String;
Begin
  TheSONumber := InsertSalesOrder('','CounterSales',date, strtodate('01/07/2008'),'Jim
Gold & Co Ltd','Fax Order','','');
  Showmessage('The Sales Order created is ' + TheSONumber);
End.
```

4.117 InsertSalesOrderLine

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertSalesOrderLine(OrderNumber, CodeType, LineCode, LineQty, OverridePrice,

UnitPrice, Description, CatalogueName, LineNumber, LineNotes, RequiredDate);

This will create a Sales Order Line record against a Sales Order Header that has been previously generated. The elements that make up this function are:

> **OrderNumber**: The Sales Order Number, which must already exist
> **CodeType**: The type of line being added. The options are Item Code, Descriptor Code, Kitset Code, or Catalogue Code.
> **LineCode**: The identity of the line being added.
> **LineQty**: The Order quantity of the line being added
> **OverridePrice**: Enter 'True' if the next field has an override Price. Enter 'False' if you are using the Base Price
> **UnitPrice**: If the previous field is 'True' then enter the override Price
> **Description**: If you are overriding the Description then enter this here. If this is left blank then the Description will be copied from the Item Code, Descriptor Code, Kitset Code, or Catalogue Code.
> **CatalogueName**: If this Item is a Code Type 'Catalogue Code' then enter the Catalogue Name here
> **LineNumber**: Enter the Line Number of the Inserted Line. If nothing is entered then Ostendo will automatically allocate a Line Number
> **Line Notes**: Enter any Notes that you are including with this Item
> **RequiredDate**: Enter the Required Date for the Line. If this is not entered then today's date will be used

In this example we will add a line to an existing order. Therefore you should first ensure that a Sales Order Header exists. In the example below replace SO300019 with the actual Sales order Number. The Lines can then be added using the following function

```
Begin
    InsertSalesOrderLine('SO300018','Item Code','100-2000',10,False,0,",",",",");
    Showmessage('Sales Order Line Added');
End.
```

You should note that you can add a Sales Order Line at the same time as you create the Sales Order Header by using a declared variable. For example we will combine the InsertSalesOrder and InsertSalesOrderLine scripts as follows

```
Var
   TheSONumber: String;
Begin
    TheSONumber := InsertSalesOrder(",'CounterSales',date, strtodate('01/07/2008'),'Jim
Gold & Co Ltd','Fax Order',",");
     InsertSalesOrderLine(TheSONumber,'Item Code','100-2000',10,False,0,",",",");
        Showmessage('Sales Order and Line Added');
End.
```

4.118 InsertTimeSheetHeader

For use with

> ***General Custom Scripts***
> ***Screen Data Script***
> ***Order Script***
> ***Custom Product Script***

Format: InsertTimesheetHeader(Status, Reference, EntryStyle, EntrySelection, UseTimeOfDay, TimesheetDate, ApprovalStatus, ApprovalDate, ApprovalUserName,

TimesheetNotes, TimesheetBatchDate);

This will create a Timesheet Header record. The elements that make up this function are:

Status: The status of the Timesheet. This would normally be ‘InProgress’ however the status could be ‘Updated’ if you are adding Timesheet history
Reference: Reference text against the Timesheet. This can be blank
EntryStyle: The entry style which can be one of Employee, Job, Assembly, Non-Charge, or Any
EntrySelection: Mandatory entry that must be related to the Entry Style chosen in the previous field
UseTimeOfDay: If ‘True’ then any Timesheet Lines will have a start and finish time. If False then the lines will have duration only
TimesheetDate: The date the Timesheet was raised. You can enter the word ‘date’ to denote the system date or specify strtodate(‘01/07/2008’) to convert an entered date (Format dd/mm/yyyy) into a date field
ApprovalStatus: If ‘Timesheet Approvals’ under Labour Settings is checked then this must have an entry. The entry options are ‘Waiting Approval’, ‘Approved’, or ‘Approval OnHold’.
ApprovalDate: If the status is ‘Approved’ then this must contain a date. This can be the word ‘date’ to denote the system date or you can specify strtodate(‘01/07/2008’) to convert an entered date (Format dd/mm/yyyy) into a date field
ApprovalUserName: The Name of a User who has ‘Approvals’ authority
TimesheetNotes: Additional Notes that can apply to the Timesheet. This can be blank
TimesheetBatchDate: The date of the Timesheet Batch. You can enter the word ‘date’ to denote the system date or specify strtodate(‘01/07/2008’) to convert an entered date (Format dd/mm/yyyy) into a date field

In this example we will create a Timesheet for Bob Drum

```
Begin
        InsertTimesheetHeader('InProgress','','Employee','Bob Drum',False,date,'
        Approved',date,'Admin','',date);
        Showmessage('Timesheet Header Created');
End.
```

You should note that, in addition to creating the Timesheet Header, you can also return the Timesheet Batch Number to a variable. This variable can then be used in other parts of the script. For example the above script can be extended as follows

```
Var
  TheBatchNumber: Integer;
Begin
TheBatchNumber := InsertTimesheetHeader('InProgress','','Employee','Bob Drum'
,False,date,'Approved',date,'Admin','',date);
Showmessage(TheBatchNumber);
End.
```

4.119 InsertTimeSheetLine

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: InsertTimesheetLine(BatchNo, DateWorked, OrderType, ReferenceNumber,

Employee, EmployeeRateScale, TaskorStepName, LabourCode, DayStartTime, DayEndTime, Hoursworked, ChargeStyle, NonChargeCode, RecordedNotes, CopyToHistory, CopyToLines, CopyToInvLine, LineStatus);

This will create a Timesheet Line record. The elements that make up this function are:

BatchNo: The Timesheet Batch Number held against the Timesheet Header record.
DateWorked: The date the work was carried out. You can enter the word ‘date’ to denote the system date or specify strtodate(‘01/07/2008’) to convert an entered date (Format dd/mm/yyyy) into a date field
OrderType: The type order against which the work was carried out. The options are Job, Assembly, or Non-Charge
ReferenceNumber: the Reference number within the Order Type entered in the previous field. This must exist in Ostendo
Employee: The Employee Name
EmployeeRateScale: Mandatory entry that must be related to a Rate Scale maintained under Labour>Settings
TaskOrStepName: A valid Task if booking against a Job or Step Name if booking against an Assembly Order. Not required for Non-Charge
LabourCode: A valid Labour Code is required
DayStartTime: If the ‘UseTimeOfDay’ in the Timesheet Batch header is True then this must have an entry to the format HH:MM
DayEndTime: If the ‘UseTimeOfDay’ in the Timesheet Batch header is True then this must have an entry to the format HH:MM
HoursWorked: If ‘Timesheet Approvals’ under Labour Settings is not checked then this must have an entry. The entry is in the form of HH.DD
ChargeStyle: Enter the Charge Style. The options are Chargeable, Warranty, Contract, or Non-Charge
Non-Charge Code: If the Charge Style is not Chargeable then this must have an entry that relates to the Style being booked against
RecordedNotes: Additional Notes that can apply to the Timesheet Line. This can be blank
CopyToHistory: Can be True or False as required
CopyToLines: Can be True or False as required
CopyToInvLine: Can be True or False as required
LineStatus: Can be InProgress or Updated

In this example we will add a Timesheet Line to a previously created Timesheet Header. In this example replace the first number (4) with the timesheet Batch Number

```
Begin
        InsertTimesheetLine(4,date,'Job','JOB400008','Bob
        Drum','STD','Job','LAB-ASSEMBLY',0,0,5,'Chargeable','','',False,False,False,'
        InProgress');
        Showmessage('Timesheet Line Added');
End.
```

You should note that you can add a line to the Timesheet at the same time as you create the Timesheet Batch by using a declared variable. For example we will combine the InsertTimesheetHeader and InsertTimesheetLine scripts as follows

```
Var
  TheBatchNumber: Integer;
Begin
TheBatchNumber := InsertTimesheetHeader('InProgress','','Employee','Bob Drum'
,False,date,'Approved',date,'Admin','',date);
```

```
InsertTimesheetLine(TheBatchNumber,date,'Job','JOB400008','Bob
Drum','STD','Job','LAB-ASSEMBLY',0,0,5,'Chargeable','','',False,False,False,'InProgress');
Showmessage('Timesheet Header and Line Added');
End.
```

4.120 LineQty

For use with

Custom Product Script

Format: LineQty;

This Custom Product Script variable allows you to evaluate the Line quantity and attach it to this pre-defined Variable. The value in this Variable will automatically populate the Quantity in the generated Custom Product Line. If this Variable is not used then a value of 1 is assumed.

Example:

```
Var
  ReqdQty, ScrapQty: Double;
Begin
  LineQty := (ReqdQty * ScrapQty);
End.
```

4.121 LinesCustomLookup

For use with

Edit View Scripts

Format: Procedure LinesCustomLookup(FieldName: String);

This Procedure is used in combination with Procedure LinesCustomLookupButtonClick and allows you to define that a field value is derived from a drop-down Lookup where the Lookup content is defined in the script.

In the following example we will assume that you already have created an Edit View and we are going to create a Custom Lookup against (say) field 'Item_Code'

Procedure LinesCustomLookup enables you to nominate the Lines record field against which the lookup will appear. Add the following to the end of the Edit View Script.

```
Begin
  LinesCustomLookup('Item_Code');
End.
```

The next step is to declare the Custom Lookup under Procedure LinesCustomLookupButtonClick as follows:

```
procedure LinesCustomLookupButtonClick(DisplayValue: Variant; AField: TField);
var
  LookupResult: String;
begin
   if AField.Fieldname = 'Item_Code' then
  begin
    LookupResult := DisplayData('Select ITEMCODE from ITEMMASTER', 'Item Lookup', '
ITEMCODE');
```

```
  if trim(LookupResult) <> '' then
  begin
    AField.DataSet.Edit;
    AField.AsString := LookupResult;
  end;
 end;
end;
```

If you go into the Edit View you will see a 'spyglass' symbol in the Item_Code field which, when clicked, will bring up the Item listing

4.122 LinesCustomLookupButtonClick

For use with
Edit View Scripts

Format: Procedure LinesCustomLookupButtonClick(DisplayValue: Variant; AField: TField);

This Procedure is used in combination with Procedure LinesCustomLookup and allows you to define that a field value in the Lines Tab is derived from a drop-down Lookup where the Lookup content is defined in the script.

In the following example we will assume that you already have created an Edit View and we are going to create a Custom Lookup against (say) field 'Item_Code'

Procedure LinesCustomLookup enbles you to nominate the Lines record field against which the lookup will appear. Add the following to the end of the Edit View Script.

```
Begin
  LinesCustomLookup('Item_Code');
End.
```

The next step is to declare the Custom Lookup under Procedure LinesCustomLookupButtonClick as follows:

```
procedure LinesCustomLookupButtonClick(DisplayValue: Variant; AField: TField);
var
  LookupResult: String;
begin
   if AField.Fieldname = 'Item_Code' then
  begin
    LookupResult := DisplayData('Select ITEMCODE from ITEMMASTER', 'Item Lookup', '
ITEMCODE');
  if trim(LookupResult) <> '' then
  begin
    AField.DataSet.Edit;
    AField.AsString := LookupResult;
  end;
 end;
end;
```

If you go into the Edit View you will see a 'spyglass' symbol in the Item_Code field which, when clicked, will bring up the Item listing

4.123 LinesDomainCombo

For use with
Edit View Scripts

Format: LinesDomainCombo(FieldName, Domain);

This function allows you to create an entry field in the Lines record containing data that currently exists against Ostendo's Domains. The elements that make up this function are:

FieldName: The FieldName in the 'Lines' record. This should refer to the re-defined name in the Edit View Master Query and not the database field name.
Domain: The Domain Name.

This is an example of how you would declare the Values in a 'Lines' screen against domain ITEM_STATUS

```
Begin
  DetailValuesCombo('ItemStatus','ITEM_STATUS');
End.
```

4.124 LinesFocusedItemChanged

For use with
Edit View Scripts

Format: LinesFocusItemChanged(FocusedFieldName: String; PrevFocusedFieldName: String);

This procedure enables you to define what to do with related fields based upon the selection made in a nominated field in the Lines record. In this example we will define (for example) that a field in the Edit View Lines record called **LOANEDTOTYPE** contains values of Employee, Customer, Supplier and that the drop-down against the associated field **LOANEDTONAME** relates to the selection made

```
Procedure LinesFocusedItemChanged(FocusedFieldName: String;
PrevFocusedFieldName: String);

begin
  if (FocusedFieldName = 'Loaned_To_Name') then
  begin
    if linesquery.fn('Loaned_To_Type').AsString = 'Employee' then
      Begin
       LinesLookup('Loaned_To_Name',1040);
      end;
    if linesquery.fn('Loaned_To_Type').AsString = 'Customer' then
      Begin
       LinesLookup('Loaned_To_Name',1015);
      end;
    if linesquery.fn('Loaned_To_Type').AsString = 'Supplier' then
      Begin
       LinesLookup('Loaned_To_Name',1001);
      end;
  end;
end;
```

This states that if the cursor is placed in field '**Loaned_To**' then look at the current entry in field

'Loaned_To_Type' and show the relevant drop-down

4.125 LinesLookup

For use with
Edit View Scripts

Format: LinesLookup(FieldName, Lookup Number);

This function allows you to create an entry field whose data is populated from another table in Ostendo using the Lookup Reference Number defined for the Table. The elements that make up this Function are:

FieldName: The FieldName in the 'Lines' record. This should refer to the re-defined name in the Edit View Master Query and not the database field name.
Lookup Number: The Lookup Numbers as defined in the Scripting Help that refers to the relevant Tables

This is an example of how you would declare the Values in a 'Lines' screen against field (say) Inventory Adjustment_Type

```
Begin
  DetailLookup('Adjustment_Type',1005);
End.
```

4.126 LinesNewRecord

For use with
Edit View Scripts

Format: LinesNewRecord(DataSet: TpFIBDataSet);

This procedure enables you to prefill a field in the Lines record whenever a new record is being added. Of course you could amend this in the record unless it has been nominated as a 'Read Only' field. You should define the field in the 'Lines' record and the value that you wish to pre-populate it with. For example:

```
procedure LinesNewRecord(DataSet: TpFIBDataSet);
begin
  DataSet.FN('ItemSourcedBy').AsString := 'Supply from Stock';
end;
```

4.127 LinesReadOnly

For use with
Edit View Scripts

Format: LinesReadOnly(FieldName,True);

This function allows you to define that a field is Read-Only and cannot be amended. This is useful (a) when a field is copied from the main Ostendo Tables and cannot be amended in the Edit View, or (b) an Edit View has been created specifically for a User in which the data can be seen but not amended
The elements that make up this function are:

FieldName: The FieldName in the 'Lines' record. This should refer to the re-defined name in the Edit View Master Query and not the database field name.

True: If this is ‘True’ then this field cannot be amended in the Edit View. If set to ‘False’ then it can be amended.

This is an example of how you would define that the Lines field covering a Sell Price cannot be amended

```
Begin
  LinesRead Only('SELL_PRICE',True);
End.
```

4.128 LinesValidate

For use with
Edit View Scripts

Format: LinesValidate(DisplayValue: Variant; AField: TField);

This procedure allows you to carry out actions against a ‘Lines’ Level field based upon the entry in the nominated field. This example will validate that a 'Lines' field called (say) Planned_Date_Out is not earlier than the System Date.

```
procedure LinesValidate(DisplayValue: Variant; AField: TField);
begin
  if AField.fieldname = 'Planned_Date_Out' then
   begin
     if DisplayValue('Planned_Date_Out').AsDateTime < NOW then
       begin
       Showmessage(‘Date is before today’);
    end;
  end;
end;
```

4.129 LinesValuesCombo

For use with
Edit View Scripts

Format: LinesValuesCombo(FieldName, Values, True);

This function allows you to create an entry field containing pre-defined data which is selected from a drop-down list. The elements that make up this function are:

FieldName: The FieldName in the ‘Lines’ record. This should refer to the re-defined name in the Edit View Detail Query and not the database field name.
Values: The Values that are to appear in the drop-down list; separated by a comma
True: If ‘True’ then only the defined Values can be selected. If False then the values are still displayed for selection but you may type in your own if required

This is an example of how you would declare the Values in a ‘Lines’ screen against field (say) Activity_Type

```
Begin
  DetailValuesCombo('Activity_Type','Repair,Maintenance,Inspection',True);
End.
```

4.130 LineUnitPrice

For use with

Custom Product Script

Format: LineUnitPrice;

This is a Custom Product Script variable which gathers the Customer Price level, The Item Code and the Quantity being ordered to evaluate the Sell Price. The result is stored against this variable

4.131 LoadSpreadSheet

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: LoadSpreadSheet(FileName);

This will load a Spreadsheet into memory for use with spreadsheet functions found later in this document. The element that makes up this function is
:

FileName: The full path of the document. Note: This must be an xls type of spreadsheet.

In this 'paper' example we will load spreadsheet Items.xls and display the content of spreadsheet row 1, column 2 into memory and, using the SSGetCellText (shown later), extract data from a specific Cell. This -**LoadSpreadSheet** - function must always be run before any data can be extracted from that Spreadsheet.
.

```
Begin
          LoadSpreadSheet('c:\Items.xls');
          Showmessage(uppercase(SSGetCellText(0,1)));
End.
```

Note: Column and/or Row 0 in the script refers to the first Colum or Row in the spreadsheet. In the above example therefore it will extract data from the First Column and the Second Row in the spreadsheet

4.132 MessageDlg

This is not a specific Ostendo Function but has been included here as it is used extensively in scripting. It is an alternative to function ShowMessage

For use across all scripting in Ostendo

Format: MessageDlg(Message,mtStyle,Button,HelpContextID)

Message: The Message that will be displayed
MtStyle: The Style of the panel. You should use mtInformation. (Other types used in general scripting are 'mterror', or 'mtwarning')
Button: defines the button(s) to be displayed. This should be set to number '4'. (Other Button styles are 1 thru 15)
HelpContextID: Not used...leave as zero

This example will return the Message in the panel format specified
Create the following script in Ostendo and run the script

```
begin
  MessageDlg('This is the message',mtInformation,4,0);
end.
```

4.133 MoveFile

This procedure enables you to move any file on your network to any other location. This can optionally be used in combination with Procedures ***DeleteFile, CopyFile, RenameFile*** and ***CreateDir***

For use with
- *General Custom Scripts*
- *Screen Data Script*
- *Order Script*
- *Custom Product Script*

Format: MoveFile(Source File, Destination File);

Source File: The full path of the file to be copied
Destination File: The full path of the file at its destination

In this example we will move a file from the 'C' Drive to the 'D' Drive.

```
begin
  MoveFile('C:\Temp\MoveScript.doc','D:\MoveScript.doc');
  Showmessage('File Moved');
end.
```

4.134 OrderScriptRun

For use with
- ***Order Script***

Format: OrderScriptRun(BeenRun, Refresh);

This is used as part of an Order Script to annotate if the script has been run, or not. The elements that make up this function are:
BeenRun: This can be set to 'True' or 'False'
Refresh: This will refresh the screen with any changes identified within this script

In this example we will create an Order Script linked to a Purchase Order. You will notice that the Purchase Order cannot be printed until the Script has been run. Once the script has been run it also will update the Tracking Code against the Order

Go into ***File>Custom Scripts*** and add a new Custom Script called (say) **PurchSupplier**. 'Check' the '**This is an Order Script**' checkbox. To define if this script must be acknowledged before the Order can proceed you should also 'check' the '**Mandatory**' checkbox.

In the '**Script**' tab enter the following:

```
var
```

```
TheSupplier: string;
OrderNo: String;
begin
   TheSupplier := GetSourceFieldValue('SUPPLIER');
   OrderNo := GetSourceFieldValue('ORDERNUMBER');
   executeSQL('update Purchaseheader set WorkFlowStatus = ''Accepted'' where
ORDERNUMBER = ''' + OrderNo + '''');
   showmessage('The Tracking Status has been updated');
   OrderScriptRun(True,True);
end.
```

The next step is to tell Ostendo that the script is linked to a Purchase Order. To do this go into ***File>System Configuration>Order Scripts*** and create a new record containing the following

Screen: Select '**Purchase Orders**' from the drop-down
Script Name: Select **PurchSupplier**

Now go into ***Purchasing>Purchase Orders*** and create a Purchase Order then add a line to the Order. If you try and pick a line then you will be presented with an error message stating '**Please click PurchSupplier before printing'**. I.e. **OrderScriptRun** field in the Purchase Order Header record is currently set to '**False**'.

You will see a new button (**PurchSupplier**) on the **Batch Entry Bar** of the Purchase Order Lines screen. If you click on this button then the script will be run. This will return the Supplier Name to the screen in addition to amending the **OrderScriptRun** field in the Purchase Order Header record to '**True**'.

This will allow you to continue with printing the Purchase Order in addition to updating the Tracking Status against the Purchase Order to '**Accepted**'

4.135 OstendoAnalysis

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: OstendoAnalysis(AnalysisName, CondValuesCommaText);

This will run the selected Analysis View

Analysis Name: The Name of the Analysis View as known by Ostendo
CondValuesCommaText: If the Analysis View has parameters then this can either be left blank and the Parameter Entry screen will be presented, or you can enter the Parameters as part of the script to automatically prefill the Parameter Values. When used the following format must be adopted:
   'Condition1=Value1' + #13 + 'Condition2=Value2'
Note: The function recognises #13 as a 'Carriage Return'

In this example we will run the 'Analysis - Item Listing' view

```
Begin
        OstendoAnalysis('Analysis - Item Listing'),
        'From Item Category=Electrical' + #13 +
        'To Item Category=Electrical' + #13 +
        'From Item Code=' + #13 +
        'To Item Code=');
End.
```

4.136 OstendoChart

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: OstendoChart(ChartName, CondValuesCommaText);

This will run the selected Chart View

Chart Name: The Name of the Chart View as known by Ostendo
CondValuesCommaText: If the Chart View has parameters then this can either be left blank and the Parameter Entry screen will be presented, or you can enter the Parameters as part of the script to automatically prefill the Parameter Values. When used the following format must be adopted:
‘Condition1=Value1’ + #13 + ‘Condition2=Value2’
Note: The function recognises #13 as a ‘Carriage Return’

In this example we will run ‘Chart - Inventory Values’

```
Begin
        OstendoChart('Chart - Inventory Values',
        'From Warehouse=Main' + #13 +
        'To Warehouse=Secondary' + #13 +
        'From Location=' + #13 +
        'To Location=' + #13 +
        'From Category=' + #13 +
        'To Category=');
End.
```

4.137 OstendoEditView

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: OstendoEditView(EditViewName, CondValuesCommaText);

This will run the selected Edit View

EditViewName: The Name of the Edit View as known by Ostendo
CondValuesCommaText: If the Edit View has parameters then this can either be left blank and the Parameter Entry screen will be presented, or you can enter the Parameters as part of the script to automatically prefill the Parameter Values. When used the following format must be adopted:
‘Condition1=Value1’ + #13 + ‘Condition2=Value2’
Note: The function recognises #13 as a ‘Carriage Return’

In this example we will run an Edit View called 'Loan Equipment'

```
Begin
        OstendoEditView('Loan Equipment');
End.
```

4.138 OstendoInquiry

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: OstendoInquiry(InquiryName, CondValuesCommaText);

This will run the selected Pivot View

InquiryName: The Name of the Inquiry as known by Ostendo
CondValuesCommaText: If the Inquiry has parameters then this can either be left blank and the Parameter Entry screen will be presented, or you can enter the Parameters as part of the script to automatically prefill the Parameter Values. When used the following format must be adopted:
  'Condition1=Value1' + #13 + 'Condition2=Value2'
Note: The function recognises #13 as a 'Carriage Return'

In this example we will run 'Inquiry- Items'

```
Begin
        OstendoInquiry('Inquiry - Items');
End.
```

4.139 OstendoPath

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script
Custom Data Screens
Data Entry Script

Format: OstendoPath;

When used in a script this Constant will locate Ostendo in a Client/Server or Peer to Peer environment and retain the full path to Ostendo from the PC

In this example we will use the path related to your PC

```
Begin
        Showmessage(OstendoPath);
End.
```

4.140 OstendoPivot

For use with
- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: OstendoPivot(PivotName, CondValuesCommaText);

This will run the selected Pivot View

Pivot Name: The Name of the Pivot View as known by Ostendo
CondValuesCommaText: If the Pivot View has parameters then this can either be left blank and the Parameter Entry screen will be presented, or you can enter the Parameters as part of the script to automatically prefill the Parameter Values. When used the following format must be adopted:
‘Condition1=Value1’ + #13 + ‘Condition2=Value2’
Note: The function recognises #13 as a ‘Carriage Return’

In this example we will run ‘Pivot - Inventory Transactions’

```
Begin
        OstendoPivot('Pivot - Inventory Transactions',
        'From Item Category=' + #13 +
        'To Item Category=' + #13 +
        'From Item Code=AAAAAAA' + #13 +
        'To  Item  Code=WWWWW' + #13 +
        'From Date=' + #13 +
        'To Date=');
End.
```

4.141 OstendoReport

For use with
- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: OstendoReport(ReportName, DeviceIndex, CondValuesCommaText, EmailAddress, CCAddress, BCCAddress);

This will run the selected Report

Report Name: The Name of the Report as known by Ostendo
Device Index: Specify how the report is to be output. The options are:
0 or Blank = Standard output option selection panel
1 = Select a Printer for printing the report
2 = Immediately print on Default Printer
3 = Email Direct
4 = Email with Dialog
5 = Output to Screen
CondValuesCommaText: If the Report has parameters then this can either be left blank and the Parameter Entry screen will be presented or you can enter the Parameters here to

automatically prefill the Parameter Values. When used the following format must be adopted:
‘Condition1=Value1’ + #13 + ‘Condition2=Value2’
Note: The function recognises #13 as a ‘Carriage Return’
EmailAddress: Only required if being sent by Email or EXPORTREPORT ******CCAddress** : Optionally required if being sent by Email
BCCAddress: Optionally required if being sent by Email

**** Note: The **EmailAddress** can be replaced with the Procedure EXPORTREPORT which allows you to export the generated pdf document to any location on your network. I.e. The parameter following this Procedure is the location where you want the Report to be saved

In this example we will print the Location Listing Report

```
Begin
        OstendoReport('Item Summary Listing',0,
        'From Item Category=Electrical' + #13 +
        'To Item Category=Electrical' + #13 +
        'From Item Code=' + #13 +
        'To Item Code=' + #13 +
        'Exclude Conditions=No');
End.
```

In this example we will place the generated report directly under the Ostendo Folder

```
Begin
  OstendoReport('Standard Item Price List',3,'','EXPORTREPORT',OstendoPath + '
ItemPriceList.pdf','');
End.
```

4.142 ParseString

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: ParseString(TextToParse, Delimiter, IndexOfString);

Parsing takes an input file and looks for the delimiter in the Text. Each delimited section will become a separate field. You can then analyse the output and extract the relevant sections. The elements that make up this function are:
TextToParse: The Text to be Parsed
Delimiter: Define the Delimiter within the Text. For example it could be a comma, semi-colon, etc.
IndexOfString: The identity of the ‘Segment’ that was Parsed. The segments start at zero and are incremented by 1

In the following exercise the Input file will contain a comma=separated Text of A,B,C,D. The parsing will output the second (B) segment (I.e IndexOfString = 1)

```
// Define the Variable
Var
  TheValue: String;
```

```
// You can then get the specific Segment using
Begin
  TheValue := ParseString('A,B,C,D',',',1);
  showmessage('The selected value is ' + TheValue);
End.
```

4.143 ProcessBarcode

For use with

- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***
- ***Data Screen Script***
- ***Data Entry Script***

This takes the entered barcode and interrogates various Tables where barcodes are used and returns a ‘string’ comprising of the Ostendo Table where the record was found and the SysUniqueID of that record.

However, if the Barcode occurs more than once across Ostendo then a panel will be returned for you to select the correct entry

Format: ProcessBarcode(Value,True);

Value: The barcode to be processed
True: The entry options are True or False, ‘False’ is the default. .If set to True then the Inventory Table is excluded from this process. Note: An Item can be in many locations in Inventory and, if Inventory is included, then a record will be returned for each Location

To prepare for running the script go into Item **100-2000** and enter Barcode **12345**. Also enter barcode **12346** against Item **100-2001**.

Run the following script

```
// Define the Variable
Var
 TheBarcode: String;
 TheItemCode: String;
// You can then ask the question to answer OptionSelect using
 Begin
  TheBarcode := AskQuestion('Please enter a Barcode','TEXT','','');
  TheItemCode := ProcessBarcode(TheBarcode,true);
  Showmessage('Your selection is ' + TheItemCode);
End.
```

If you run the script and enter Barcode **12346** then the Table (**ITEMMASTER**) and **SysUniqueID** for the Item will be returned to the text for the Object. If you then repeat this for Barcode **12345** then a panel will be presented to show the two variants of the Item. If you select one then that selection will be shown in the Object.

4.144 QueryValue

For use with

Screen Data Script

Format: QueryValue(FieldName);

This is only used with Screen Data Scripts which interrogates the source record and returns the contents of a defined field.

FieldName: The Name of a field in the source record

In this exercise we will create a script that will get the Style of a Sales Order as the order is being created. You should note that this is only used with Screen Data Scripts as the source record is interrogated and the requested field value is returned. To perpare and run this exercise carry out the following steps

Go to ***File>Custom Scripts*** and add a new Custom Script called (say) **OrdStyle** and 'check' the '**This is a Screen Data script**' checkbox. Click on the 'Script' tab and add the following script

```
var
 OrdStatus: String;
begin
 OrdStatus := QueryValue('ORDERSTYLE');
 showmessage('The Order Style is ' + OrdStatus);
end.
```

Save and exit the Custom Script screen.

The next step is to tell Ostendo that the script is linked to a Sales Order Header. To do this go into ***File>System Configuration>Screen Data Scripts*** and create a new record containing the following

Screen: Select '**Sales Orders**' from the drop-down
Table Name: Select '**SALESHEADER**' from the drop-down list
SQL Type: Select '**Insert**'
Script Name: Select the above script Name

Save and exit

Now go into ***Sales>Sales Orders*** and create a Sales Order. You will see the showmessage appearing that indicates that the status has been stored in variable OrdStatus.

4.145 ReadComPort

This is primarily used for reading data streams from external media such as barcode scanners. It will continuously read the input stream for either the timeout duration or when an EndOfDataChar is encountered.

Format: ReadComPort(Port, TimeoutSeconds, EndOfDataChar, Baudrate);
Port: The Name of the Com Port. Defaults to COM1if not specified.
TimeoutSeconds: The active duration in seconds. Defaults to 5 secs.
EndOfDataChar: Defaults to 13
Baudrate: Defaults to 9600

4.146 ReAskAllQuestions

For use with
Custom Product Script

Format: ReAskAllQuestions;

This allows you to display a Button ‘Answer All Questions Again’ on the AskQuestion panel which, when pressed, will restart the Questions. The button itself is only displayed when you read from, or write to, this Variable in the script.

In this example we will ask three questions. At any time during the flow you can click on this button to restart the questions

```
var
  Ans1, Ans2, Ans3: String;
  QuestionIndex: Integer;

procedure QuestionAsk(QIndex: Integer);
begin
  case QIndex of
  1:
    Ans1 := AskQuestion('Question 1','TEXT','Enter some Text','',Ans1);
  2:
    Ans2 := AskQuestion('Question 2','TEXT','Enter some Text','',Ans2);
  3:
    Ans3 := AskQuestion('Question 3','TEXT','Enter some Text','',Ans3);
  end;
end;

procedure START;
var
  x: Integer;
begin
  for x := 1 to 3 do
  begin
    inc(QuestionIndex);
    QuestionAsk(QuestionIndex);
    if REASKALLQUESTIONS then
    begin
      REASKALLQUESTIONS := False;
      QuestionIndex := 0;
      FinishQuestions;
      START;
      exit;
    end;
  end;
end;

begin
  Start;
  showmessage(Ans1 + ' , ' + Ans2 + ' , ' + Ans3);
end.
```

4.147 ReAskLastQuestion

For use with
Custom Product Script

Format: ReAskLastQuestion;

This allows you to display a Button ‘Answer Last Question Again’ on the AskQuestion panel which, when pressed, will ask you to re-enter the answer to the previous question. The button itself is only displayed when you read from, or write to, this Variable in the script.

In this example we ask three questions. The screen will also include ReAskAllQuestions button along with this AskLastQuestion button. At any time during the flow you can click on the AskQuestionButton you can re-enter and answer to the previous question.

```
var
  Ans1, Ans2, Ans3: String;
  QuestionIndex: Integer;

procedure QuestionAsk(QIndex: Integer);
begin
  case QIndex of
  1:
    Ans1 := AskQuestion('Question 1','TEXT','Enter some Text','',Ans1);
  2:
    Ans2 := AskQuestion('Question 2','TEXT','Enter some Text','',Ans2);
  3:
    Ans3 := AskQuestion('Question 3','TEXT','Enter some Text','',Ans3);
  end;
end;

procedure START;
var
  x: Integer;
begin
  for x := 1 to 3 do
  begin
    inc(QuestionIndex);
    QuestionAsk(QuestionIndex);
    if REASKALLQUESTIONS then
    begin
      REASKALLQUESTIONS := False;
      QuestionIndex := 0;
      FinishQuestions;
      START;
      exit;
    end
    else
    if REASKLASTQUESTION then
    begin
      REASKLASTQUESTION := False;
      dec(QuestionIndex);
      QuestionAsk(QuestionIndex);
      START;
      exit;
    end;
  end;
end;

begin
  Start;
  showmessage(Ans1 + ' , ' + Ans2 + ' , ' + Ans3);
end.
```

4.148 ReceiveEmail

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: ReceiveEmail(Host, User, Password, DeleteMessages, AttachPath, Port, SSL);

This function allows you to receive emails from your email Service Provider. Having received the emails you can then extract information from selected emails. The email receipt process comprises of:

- **ReceiveEmail**; This receives the email(s) and gives each email a Message Number
- **GetEmailMessage**: This allows you to reference a specific Message Number and extract data from that message
- **GetEmailAttachmentCount**: This counts the number of attachment to a referenced Message Number

In this exercise we will simply receive the email and carry out a Message Count

The elements that make up this function are:

Host: The email server account name that services your emails. For Example: pop3.youremailserver.com
User: The pop3 User Name for access to Host.
Password: The pop3 password for access to Host.
Delete Messages: If you wish to delete the messages from the Host machine after receipt then enter TRUE, other leave blank
AttachPath: This is an optional entry to define path where attachments will be downloaded. (Defaults to the setting held against the 'Default email account' as set up on your PC)
Port: The standard pop3 port is 110 and this is the default if you leave this blank. An entry here allows you to change this Port setting
SSL: This should be either 'True' or 'False' (Defaults to 'False')

In the following exercise we will receive emails and carry out a message count.

It is suggested that you first send a couple of emails using the ***sendmailmessage*** function and then run this routine which includes the ***GetEmailMessage*** and ***GetEmailAttachment*** functions.

Create a new Custom Script with the following. Amend the '**Const**' values to suit your environment

```
Const
  Host = 'pop3.myemailhost.com';
  User = 'YourUserName';
  Password = 'yourpassword';
Var
  MessageCount,AttachmentCount,x,y: Integer;
  MessageBody, MessageFrom,MessageSubject,AttachmentFileName: String;
begin
  MessageCount := ReceiveEmail(Host,User,Password);
  Showmessage('Number of Messages is ' + inttostr(MessageCount));
  For x := 1 to MessageCount do
   Begin
    MessageSubject := GetEmailMessage('SUBJECT',x);
    MessageFrom := GetEmailMessage('FROMADDRESS',x);
    MessageBody := GetEmailMessage('BODY',x);
    Showmessage('The contents for email ' + inttostr(x) + ' is:' + #13 +
              'Subject: ' + MessageSubject + #13 +
              'From: ' + MessageFrom + #13 +
              'Body: ' + MessageBody);
    AttachmentCount := GetEmailAttachmentCount(x);
    For y := 1 to AttachmentCount do
     Begin
      AttachmentFileName := GetEmailMessage('ATTACHMENT',x,y);
      Showmessage(AttachmentFileName);
     end;
   end;
end.
```

4.149 RefreshActiveScreenHeader

For use with

Screen Data Scripts

Format: RefreshActiveScreenHeader := true;

This relates specifically to 'Related Menu' scripts whereby if a change is made to the linked Header then this function will refresh the data displayed in that linked Header screen.

To see this in action go into ***Inventory>Items*** and create a new Item

Next, go to Go into ***File>Custom Scripts*** and add a new Custom Script called (say) **ItemChange**.

In the '**Script**' tab enter the following:

```
begin
 executeSQL('update itemmaster set ITEMBARCODE = ITEMBARCODE || "Z" where
itemcode = "' + queryvalue('ITEMCODE')  + '"');
 refreshactivescreenheader := true ;
end.
```

Save and exit the Custom Script screen.

The next step is to tell Ostendo that the script is linked to the Item table. To do this go into ***File>System Configuration>Screen Data Scripts*** and create a new record containing the following

Screen: Select '**Items**' from the drop-down
Table Name: Select '**ITEMMASTER**' from the drop-down list
SQL Type: Select '**Update**'

Script Name: Select the above script Name
Save and exit
If you now go back to the Item Master screen and make a change to any field except the Barcode field. If you 'Save' the change you will find that the Barcode field is updated with a 'Z'

4.150 RefreshActiveScreenLine

For use with
Screen Data Scripts

Format: RefreshActiveScreenLine := true;

This relates specifically to 'Related Menu' scripts whereby if a change is made to the linked Order Line then this function will refresh the data displayed in that linked Order Line screen.

For further information see RefreshActiveScreenHeader

4.151 RefreshJobCalendar

If you have the Job Calendar open and you make changes to Job Activities (E.g. Resource status) via a script then those changes will not be reflected in the current display. This procedure will refresh the Job Calendar

4.152 RelatedMenuItemClicked

For use with
Edit View Scripts

Format: Procedure RelatedMenuItemClicked(MenuIndex: Integer);

This Procedure is used in combination with function AddRelatedMenuItem and allows you to add your own related screens to the 'Related' Button in the Edit View Panel

The elements that make up this procedure are:
MenuIndex: The MenuIndex as determined by the Procedure AddRelatedMenuItem

In the following example we will assume that you already have a related screen and we are going to add this to the 'Related' button in that view.

This Function determines the 'Index Number' in the List of Related screens displayed under the 'Related' button. To get the Index Number add the following to the end of the Edit View Script.

```
Begin
  AddRelatedMenuItem('Screen Name');
End.
```

Where you should replace 'Screen Name' with the name of your Screen.

If you go into the Edit View then you will see this option presented when you click on the 'Related' button. Now let us run this procedure.

If it is the only screen under the Related button then its Index will be 0 therefore add this to the Edit View Script under the RelatedMenuItemClicked Procedure.

```
procedure RelatedMenuItemClicked(MenuIndex: Integer);
```

```
begin
  If MenuIndex = 0 then
  begin
    RunSystemAction('Sales', 'Customers');
  end;
end;
```

If you go into the Edit View and select this under the 'Related' button you will find that the Customer Master screen will be presented.

4.153 RelatedScreenRefreshData

For use with

Screen Data Scripts

Format: RelatedScreenRefreshData(HeaderOrLine);

This relates specifically to 'Related Menu' scripts whereby if a change is made to the linked Order Header then this function will refresh the data displayed in that linked Order Header screen.

The element that makes up this function is:

HeaderOrLine: This refers to whether you are refreshing the Order Header or the Order Line. The options are 'HEADER' or 'LINE'

An example of where this is used can be seen in the 'Re-allocate Purchase Lines' script in the ' Useful Scripts' section of this Help

4.154 RenameFile

This procedure enables you to rename any file on your network. This can optionally be used in combination with Procedures ***DeleteFile, CopyFile, MoveFile*** and ***CreateDir***

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: RenameFile(Old File Name, New File Name);

Old File Name: The full path of the file to be renamed
New File Name: The full path of the file at its destination

In this example we will rename a file on the 'C' Drive.

```
begin
  RenameFile('C:\Temp\RenameScript.doc', 'C:\Temp\TheScript.doc');
  Showmessage('File Renamed');
end.
```

4.155 ReportMenuItemClicked

For use with
  Edit View Scripts

Format: Procedure ReportMenuItemClicked(MenuIndex: Integer);

This Procedure is used in combination with function AddReportMenuItem and allows you to add your Report or Analysis View to the 'Reports' Button in the Edit View Panel

The elements that make up this procedure are:
  MenuIndex: The MenuIndex as determined by the Procedure AddReportMenuItem

In the following example we will assume that you already have created an Edit View and we are going to add a report to the 'Reports' button in that view.

This Function determines the 'Index Number' in the List of Reports displayed under the 'Reports' button. To get the Index Number add the following to the end of the Edit View Script.

```
Begin
  AddReportMenuItem('Report Name');
End.
```

Where you should replace 'Report Name' with the name of your Report.

If you go into the Edit View then you will see this option presented when you click on the 'Reports' button. Now let us run this procedure.

If it is the only report under the Reports button then its Index will be 0 therefore add this to the Edit View Script under the ReportMenuItemClicked Procedure.

```
procedure ReportMenuItemClicked(MenuIndex: Integer);
begin
  If MenuIndex = 0 then
  begin
    OstendoReport('Report Name');
  end;
end;
```

If you go into the Edit View and select this under the 'Reports' button you will find that the report will be run.

4.156 ReplaceText

For use with
  General Custom Scripts
  Screen Data Script
  Order Script
  Custom Product Script

Format: ReplaceText(TextString, FromText, ToText);

This function allows you to identify a Text and then if a defined string is found within that Text then replace it with another string.

The elements that make up this function are:

TextString: The Text being analysed to see if it contains the string to be replaced
FromText: The string being searched for
ToText: The string that will replace the FromText if found

In this example we will search a Text field for 45 and replace it with 99. The resultant Text will be shown in a Showmessage

```
// Define the Variable
Var
 TheValue: String;
// You can then get the result using
Begin
 TheValue := ReplaceText('12ABC3456789', '45', '99');
 showmessage('The result of the replaced text is ' + TheValue);
End.
```

4.157 RoundToDecimalPrecision

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Variable := RoundToDecimalPrecision(Variable1,DecimalPlaces);

This function allows you to round any evaluated Variable to a defined number of decimal places. The elements that make up this function are:
Variable: The defined variable against which the result will be held.
Variable1: The variable to which this rounding is being applied
DecimalPlaces: Number of decimal places required (defaults to 2 if not specified)

In this example we will convert the entered variable and round to 4 decimal places

```
// Define the Variable
Var
 TheResult: Double;
// You can then get the result using
Begin
 TheResult := RoundToDecimalPrecision(123.456789, 4);
 showmessage('The result of the rounded Value is ' + floattostr(TheResult));
End.
```

4.158 Run

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: Run(FileName, QuoteFileParameterIfContainsSpaces{Opt Default=True});

This function allows you to run any file from within the script. The elements that makes up this function are:

FileName: The full path of the file

QuoteFileParameterIfContainsSpaces: This is optional. The default boolean value is True. This means if you are passing parameters to the exe file, then you should enclose the parameter string with double-quotes if there are blanks or spaces in the parameter string. Else specify False.

Note: If you are passing parameters, it is best to check what parameter strings are actually passed back to your program - especially if you are passing multiple parameters.

In this example we will run Notepad

```
Begin
  Run('Notepad.exe');
End.
```

or

```
Begin
  Run('Notepad.exe "C:\Temp\test.txt" ');
End.
```

or

```
Begin
  Run('Notepad.exe C:\Temp\test.txt ', false);
End.
```

4.159 RunInventoryReplenishment

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: RunInventoryReplenishment(Horizon, HorizonDays, ItemFrom, ItemTo, CategoryFrom, CategoryTo, ABCFrom, ABCTo, ExcludeForecast, MultiLevelExplosion, ConvertTransferToPurchase, IncludePricingForecast, SpecificSite, SiteName);

This function allows you to run the Inventory Replenishment Routine of Ostendo. The elements that make up this function are:

Horizon: The scheduling option relating to the Horizon Days. The options are LeadTime or Fixed
HorizonDays: The number of days relating to the previous option
ItemFrom: Enter the start Item Code for the run criteria. Not required if Multi-Level is False.
ItemTo: Enter the end Item Code for the run criteria. Not required if Multi-Level is False.
CategoryFrom: Enter the start Category Code for the run criteria. Not required if Multi-Level is False.
CategoryTo: Enter the end Category Code for the run criteria. Not required if Multi-Level is False.
ABCFrom: Enter the start ABC Code for the run criteria. Not required if Multi-Level is False.
ABCTo: Enter the end ABC Code for the run criteria. Not required if Multi-Level is False.
ExcludeForecast: This should be either True or False

MultiLevelExplosion: This should be either True or False to indicate exploding all BOM contents for this replenishment
ConvertTransferToPurchase: This should either be True or False
IncludePricingForecast: This should either be True or False
SpecificSite: This should either be True or False
SiteName: Enter the Specific Site for this Replenishment

In this example we will run a simple Replenishment

```
Begin
  RunInventoryReplenishment('LeadTime', 7, '100-2000', '100-2006', 'Composite', 'Composite', '', '',
False, True, False, False, True,'ABCSite');
End.
```

4.160 RunSystemAction

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: RunSystemAction(Category, SystemAction);

This function allows you to run any menu-listed screen in Ostendo. The elements that make up this function are:
Category: The Main Menu Item in Ostendo (Example: Inventory, Purchasing, etc)
SystemAction: The specific screen you wish to run that resides under the above Menu

For example: If you wish to run the Knowledge Base then the entry will be.

```
Begin
   runsystemaction('General','Knowledge Base');
End.
```

4.161 SaveSpreadSheet

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SaveSpreadSheet(FileName);

This function allows you to save the Spreadsheet you have previously generated using the SpreadSheet functions shown in this document. The elements that make up this function are:
FileName: The full path of the File to be saved including the File Name with an extension of .xls

In this exercise we will load a Spreadsheet using the LoadSpreadSheet function and then save it using this function

```
Begin
        LoadSpreadSheet('C:\Test.xls');
        SaveSpreadSheet('C:\Spreadsheet.xls');
        Showmessage('Spreadsheet Saved');
 End.
```

4.162 SaveValueToStore

For use with

- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

This function retrieves values that have been previously stored in memory via the SaveValueToStore function. The elements that make up this function are:

Name: The Name of the String that is currently in memory

This process will use the SaveValueToStore function to store the value in memory after which we will use this function to retrieve and display it

```
Begin
  SaveValueToStore('FirstName=Fred');
  Showmessage(GetValueFromStore('FirstName'));
End.
```

4.163 SendEmailMessage

For use with

- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: SendEmailMessage(Host, From, Recipients, Subject, Body, Attachments, CC, BCC, User, Password, Port, SSL);

This function allows you to send an email message from data generated within the script. The elements that make up this function are:

Host: The email server account name that services the emails. For Example: smtp.youremailserver.com
From: The sender of the email. Example info@development-x.com
Recipients: The email address of the recipient. If you are sending to more than one recipient then they can be separated with a semi-colon.
Subject: The entry that will appear in the email Subject line
Body: The entry that will appear in the body section of the email
Attachments: Full Path of the attachment (Example: C:\MyDocument ,doc). Separate by semi-colon for multiple attachments
CC: The email address of the copied recipient. If you are sending to more than one recipient then they can be separated with a semi-colon.
BCC: The email address of the 'Blind' copied recipient. If you are sending to more than one recipient then they can be separated with a semi-colon.
User: The User Name for access to Host. (Defaults to the setting held against the ' Default email account' as set up on your PC)

Password: The password for access to Host. (Defaults to the setting held against the ' Default email account' as set up on your PC)
Port: The Port defined by your email provider. (Defaults to the setting held against the ' Default email account' as set up on your PC)
SSL: This should be either 'True' or 'False' (Defaults to 'False')

In this example we will use Constants to define fixed values (You would obviously insert your own values linking to your own Host) and then construct the email itself

```
Const
        TheEmailHostName = 'smtp.development-x.com';
        TheEmailSenderAddress = 'Fred@development-x.com';
        TheUserEmail= 'Email Message Test';
        TheRecipientsEmail =  'Jim@recientsEmail.com';
        TheEmailSubject = 'Email Message Test';
Begin
        SendEmailMessage(TheEmailHostName,TheEmailSenderAddress,TheRecipients
        Email,'Test Subject Text','Test email Body Text');
End.
```

4.164 SendEmailMessageExternal

For use with
- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: SendEmailMessageExternal(Subject, Body,SenderAddress,SenderName,Recipients, Attachments,RequestReceipt, ShowDialog);

This function allows you to send an email message using an external email client instead of the built-in email client. the Email Client is specified in System Settings / User Options. The Email Client must be open/active.
The elements that make up this function are:
Subject: The entry that will appear in the email Subject line
Body: The entry that will appear in the body section of the email
SenderAddress: The email address of the sender
SenderName: The name of the Sender
Recipients: the email addresses of the recipients. If you are sending to more than one recipient then they can be separated with a semi-colon.
Attachments: Full Path of the attachment (Example: C:\MyDocument\abc.doc). Separate by carriage return (#13) for multiple attachments
RequestReceipt: True or False (default = False)
ShowDialog: True or False (default = True)

In this example we will use Constants to define fixed values (You would obviously insert your own values linking to your own Host) and then construct the email itself

```
Const
        TheEmailSenderAddress = 'Fred@development-x.com';
        TheSenderName = 'Fred' ;
        TheUserEmail= 'Email Message Test';
        TheRecipientsEmail =  'Jim@recientsEmail.com';
        TheEmailSubject = 'Email Message Test';
```

```
Begin
        SendEmailMessageExternal(TheEmailSubject,TheUserEmail,TheEmailSenderAd
        dress,TheSenderName,TheRecipientsEmail,'',False,True);
End.
```

4.165 SendESCPOSToPort

For use with

> ***General Custom Scripts***
> ***Screen Data Script***
> ***Order Script***
> ***Custom Product Script***

Format: SendESCPOSToPort(EscCommand, Port);

This function allows you to send an escape command to a Port on your network via the Parallel port. This can, for example' be used to open a Cash Drawer by touching a button on a touchscreen. The elements that make up this function are:

> **ESCCommand**: The escape command. For example: 27,112,0
> **Port**: The Port used by the external device being actioned by the Escape Command

4.166 SendFileFTP

For use with

> ***General Custom Scripts***
> ***Screen Data Script***
> ***Order Script***
> ***Custom Product Script***

Format: SendFileFTP(Host, User, Password, SourceFileName, DestFileName, PassiveMode, Port);

This function allows you to send a file to your ftp server so that it is available for download. The elements that make up this function are:

> **Host**: The ftp server, website hostname or IP Address to which the document or file can be uploaded.
> **User**: The Username for ftp account. For Example: ftpsite@yourservername.com
> **Password**: The Password for the ftp account.
> **SourceFileName**: The full path name of the file that you are uploading
> **DestFileName**: The name of the uploaded file as it appears on the ftp Server. For example 'mywebfile.txt' (This file will be placed on the server according to the ftp users root directory)
> **PassiveMode**: True or False setting for advanced users only. Defaults to False
> **Port**: Optional entry. Allows changing of ftp port if required. This defaults to 25 (which is the standard ftp port number)

This Example will show you how to upload a file or document to the ftp server. Note: You should replace the **Constants** Values below with your own data

```
Const
  Host = 'www.yourftpprovider.com';
  User = 'ftpuser@yourftpprovider.com';
  password = 'ftpPassword';
  SourceFileName = 'D:\Temp\ftpTestDocument.txt';
  DestFileName = 'MyTest.txt';

begin
  SendFileFTP(Host,User,Password,SourceFileName,DestFileName);
end.
```

To see that it has uploaded the file to the ftp Server, go to your ftp host site via your Internet Explorer and point to the file in its location. **Note**: as we have used the default location in the above script then a Folder will have been created that equals the Ostendo Company Name. Example:- http://www.development-x.com/demo/MyTest.txt if the upload was carried out from within the DEMO company

4.167 SendStringToPort

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SendStringToPort(Value, Port);

This function allows you to send a text string to a Port on your network via the Serial port. This is used to send Text such as the Operators Name to the port...and hence the Display Column. The elements that make up this function are:

Value: The text that is to be sent to the Port
Port: The Port used by the external device that will use the Text

4.168 SetBOMInstructions

For use with

Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when defining the Instructions against the configured BOM Header

Format: SetBOMInstructions('BOMInstructions');

This defines the detailed instructions to make the Custom Product. The elements that make up this function are:

BOMInstructions: Unlimited amount of text to describe the actions required to make the product

Example:

```
SetBOMInstructions('Cut the 2000mm Rimu Panel down to ' +
inttostr(DeskLength) + 'mm ');
```

4.169 SetBOMLeadTime

For use with
Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when defining the Lead Time against the configured BOM Header

Format: SetBOMLeadTime(LeadTime);

This defines the anticipated Leadtime to product the Custom Product. The elements that make up this function are:
LeadTime: The number of Days (integer only) required to produce the product

Example:
SetBOMLeadTime(2);

4.170 SetBOMRunDuration

For use with
Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when defining the Run Duration against the configured BOM Header

Format: SetBOMRunDuration(RunDuration,’RunDurationScale’);

This defines the total duration to make the Custom Product. The elements that make up this function are:
RunDuration: Any Number including decimals
RunDurationScale: Either ‘Hours’ or ‘Minutes’. If nothing entered then Minutes is assumed

Example:
SetBOMRunDuration(45,'Minutes');

4.171 SetBOMSetupDuration

For use with
Custom Product Script

This function can only be used in a Script created via ***Assembly>Custom Products***. It uses the linked Item Code as the prime key when defining the Setup Duration against the configured BOM Header

Format: SetBOMSetupDuration(SetupDuration,’SetupDurationScale’);

This defines the total duration to ‘set up’ in preparation for making the Custom Product. The elements that make up this function are:
SetupDuration: Any Number including decimals
SetupDurationScale: Either ‘Hours’ or ‘Minutes’. If nothing entered then Minutes is assumed

Example:

```
SetBOMSetupDuration(15,'Minutes');
```

4.172 SetScreenParameter

For use with
General Custom Scripts

This function is used in a Related Menu script and allows you to pass parameters from the current screen to the Related screen.

Format: SetScreenParameter(Value);

The element that makes up this function is:
Value: The field Value being passed

Example 1:

To see this in action create a new Script called '**Job Customer**'. In the '**Detail**' tab 'check' the '**Add to this Screen**' checkbox and select '**Job Orders**' from the drop-down list in the adjacent field. In the '**Script**' tab add the following script

```
begin
  RunSystemAction('Sales','Customers');
  SetScreenParameter('KEYFIELD=CUSTOMER');
  SetScreenParameter('KEYVALUE=' + GetSourceFieldValue('CUSTOMER'));
  SetScreenParameter('TABINDEX=1');  // This will open up in the Customer 'Detail'
screen rather than the 'List' screen
end.
```

Save and exit the Custom Script screen.

Go into ***Jobs>Job Orders*** and click on the '**Related**' button. You will find that the '**Job Customer**' Custom Script appears in the drop-down list. If you select this then the script will bring up the Customer Screen relating to this Job Order.

Example 2:

This script allows the user to open relevant **Sales Deliveries** for a Sales Order they are positioned on from the **Sales Orders** screen

```
var
TheSQL, TheDeliveryNo,TheOrderNumber :string;

begin
   TheOrderNumber := GetSourceFieldValue('ORDERNUMBER');
   TheSQL := 'select sysuniqueid, OrderNumber, DeliveryNo, DeliveryStatus, PackedBy
from SalesDeliveryHeader where OrderNumber = ''' + theOrderNumber + '''';
   TheDeliveryNo := DisplayData(TheSQL,'Select a Delivery for Drilldown','DeliveryNo');

   If (TheDeliveryNo <> '') then
   begin
      setscreenparameter('keyfield=DeliveryNo'); {This sets the key field in called screen}
      setscreenparameter('keyvalue=' + TheDeliveryNo); {This is the selected field value
from the Displayed List}
            setscreenparameter('tabindex=1'); {This sets which Tab the called screen will display
in 0=List 1=Detail}
            SetScreenParameter('INCLUDECLOSEDUPDATED=true'); //can be (True / 1) or
(False / 0) any case
            runsystemaction('Sales','Sales Deliveries'); {This actually calls the screen based on
Module & Screen name}
          end;
        end.
```

4.173 SetWorkflowObjectColour

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SetWorkflowObjectColour(ObjectID, ObjectColour);

This function allows you to amend the colour of an Object within a Workflow. The Object must not be currently set to 'Gradient Fill'. This is useful if you wish to have a visual presentation of the status of an object, etc. The elements that make up this function are:

Object ID: Right-Mouse on the Object in the Workflow to get the ID
Colour: See the defined colours in the Workflow Editor

Having created an object in a workflow find out the Object ID by 'right-mousing' on the Object or go into OstDesigner.

The Colour selection is the standard colour preceded with the letters cl. Alternatively you can create your own colour against (say) a Tracking Code and then see - in the database - what the number code is that defines that unique colour. You can then insert the number in place of the colour. For example you can enter either clAqua or 13959039

```
Begin
    SetWorkflowObjectColour(4,claqua);
    Showmessage('Colour Updated');
End.
```

4.174 SetWorkflowObjectGradientColour

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SetWorkflowObjectGradientColour(ObjectID, BeginColour, EndColour);

This function allows you to amend the Gradient colour of an Object within a Workflow. The Object must be currently set to 'Gradient Fill'. This is useful if you wish to have a visual presentation of the status of an object, etc. The elements that make up this function are:

Object ID: Right-Mouse on the Object in the Workflow to get the ID
BeginColour: Set the Begin Colour of the Gradient Fill
EndColour: Set the End Colour of the Gradient Fill

Having created an object in a workflow find out the Object ID by ‘right-mousing’ on the Object or go into OstDesigner.

The Colour selection is the standard colour preceded with the letters cl. Alternatively you can create your own colour against (say) a Tracking Code and then see - in the database - what the number code is that defines that unique colour.   You can then insert the number in place of the colour.   For example you can enter either clAqua or 13959039

```
Begin
        SetWorkflowObjectGradientColour(4,claqua,clnavy);
        Showmessage('Colour Updated');
End.
```

4.175 SetWorkflowObjectHint

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SetWorkflowObjectHint(ObjectID, Hint);

This function allows you to add or amend the Hint held against an Object within a Workflow. Note: A Hint is the line of text that appears when you move the cursor over the Object. The elements that make up this function are:

Object ID: Right-Mouse on the Object in the Workflow to get the ID
Hint: The Text shown when the cursor moves over the Object

Having created an object in a workflow find out the Object ID by ‘right-mousing’ on the Object or go into OstDesigner.

This example will create a Hint of ‘This is a Hint’ against the Object

```
Begin
        SetWorkflowObjectHint(4,'This is a Hint');
        Showmessage('Hint Updated');
End.
```

4.176 SetWorkflowObjectTag

For use with

General Custom Scripts linked to Workflows

Format: SetWorkflowObjectTag(ObjectID,ObjectTag)

ObjectID: Right-Mouse on the Object in the Workflow to get the ObjectID.
ObjectTag: The Reference Number to populate the Object’s Tag field

This Procedure updates the Tag Number contained in the Object

This example will update the Object with a requested Tag Number. Add an Object to a Workflow. Create the following script in Ostendo and link the Object to it. If you run the script you can enter a reference Number. If you then go back to the Workflow Editor and click on this object you will see that the Tag Number – in the Inspector Panel - now contains the entered reference Number.

4.177 SetWorkflowObjectText

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SetWorkflowObjectText(ObjectID, ObjectText);

This function allows you to add or amend Text that appears within an Object. The elements that make up this function are:

Object ID: Right-Mouse on the Object in the Workflow to get the ID
ObjectText: The Text shown within the Object

Having created an object in a workflow find out the Object ID by ‘right-mousing’ on the Object or go into OstDesigner.

This example will create or overwrite the current Text within an Object. You should note that the Object itself must be a ‘Text’ object

```
Begin
        SetWorkflowObjectText(4,'Workflow');
        Showmessage('Text Updated');
End.
```

4.178 SetWorkflowObjectTransparency

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SetWorkflowObjectTransparency(ObjectID, Transparency);

This function allows you to add or amend the Object so that its transparency can be adjusted if you wish to view other Objects lying behind this one. The elements that make up this function are:

Object ID: Right-Mouse on the Object in the Workflow to get the ID
Transparency: The Transparency value expressed as a percentage

Having created an object in a workflow find out the Object ID by ‘right-mousing’ on the Object or go into OstDesigner.

To see Transparency in action you need to create two Objects where one partially overlays the other. We will apply the transparency to the front Object

```
Begin
        SetWorkflowObjectTransparency(4,50);
        Showmessage('Transparency Updated');
End.
```

4.179 SetWorkflowObjectVisible

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SetWorkflowObjectVisible(ObjectID, Visible);

This function allows you to make an Object Visible or Invisible. The elements that make up this function are:

Object ID: Right-Mouse on the Object in the Workflow to get the ID
Visible: This can be True or False

Having created an object in a workflow find out the Object ID by 'right-mousing' on the Object or go into OstDesigner.

To hide the Object use the following

```
Begin
        SetWorkflowObjectVisible(4,False);
        Showmessage('Object now Invisible');
End.
```

4.180 ShowMessage

This is not a specific Ostendo Function but has been included here as it is used extensively in scripting. It is an alternative to function MessageDlg

For use across all scripting in Ostendo

Format: Showmessage(Message)

Message: The Message that will be displayed

This example will return the Message. Create the following script in Ostendo and run the script

```
begin
  Showmessage('This is the message');
end.
```

4.181 ShowProgress

For use with

General Custom Scripts
Screen Data Script

Format: ShowProgress(Caption, Max, AllowCancel);

This function allows you to display a progress bar. The elements that make up this function are:

Caption: The caption that will be displayed at the top of the panel
Max: The Maximum Value that you are monitoring
AllowCancel: Shows a 'Cancel' button for aborting the progress

This function is used in combination with functions ***EndProgress*** and ***UpdateProgress*** (and optionally ***UpdateProgressCaption***). The following example uses the four available functions related to the Progress Bar

```
Const
  ProgressCount = 2000;
Var
  x: Integer;
begin
  ShowProgress('My Progress Bar',ProgressCount);
   For x := 1 to progressCount do
   Begin
    if x >= (ProgressCount / 2) then
     Begin
      UpdateProgressCaption('Getting There');
     end;
    UpdateProgress(x);
   end;
  EndProgress;
  Showmessage ('Progress Display Completed');
End.
```

4.182 SSGetCellText

For use with
General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SSGetCellText(Col, Row, SheetIndex);

This function allows you to get information from a selected Cell in a Spreadsheet. When using this function you should first place the Spreadsheet into memory using the LoadSpreadsheet function. The elements that make up this function are:

Col: The specific Column in the Spreadsheet
Row: The specific Row in the Spreadsheet
SheetIndex: The specific Sheet Number in the Spreadsheet

Select a Spreadsheet on your computer and run the following to load the Spreadsheet into Memory and then select data from Cell located at column1 Row 1 of the first sheet. You should note that SheetIndex, column 1 Row 1 in the spreadsheet is known as 0,0 in the function

```
begin
  LoadSpreadSheet('c:\scripttest.xls');
  showmessage(uppercase(SSGetCellText(0,0,0)));
end.
```

Of course you would not normally get information from a single cell. Here is a script that uses SSGetRowCount to determine the number of rows in a spreadsheet and then imports the content of each row using the SSGetCellText function.

In the following example will import this data into the Categories Table found under ***Inventory>Settings>Categories.***   Firstly create a spreadsheet containing multiple lines, each containing Item Category and Description.

The next step is to create the script containing the following.

```
Var
// define variables x=accumulator, RowCount = Number of Rows with data
x, RowCount :Integer;
begin
//Point to your spreadsheet
 LoadSpreadSheet('c:\ScriptImportTest.xls');
// The zero in brackets refers to the sheet number within the spreadsheet
 RowCount := SSGetRowCount(0);
// Spreadsheets start at column zero and finishes at Count minus 1
  for x := 0 to RowCount -1 do
   begin
    InsertRecord('Categories',
    'Category=' + SSGetCellText(0,x) + #13 +
    'CategoryDescription='+ SSGetCellText(1,x));
   end;
  showmessage('Categories Added');
end.
```

4.183 SSGetColumnCount

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SSGetColumnCount(SheetIndex);

This function allows you to get information about how many columns are being used in a defined Spreadsheet. When using this function you should first place the Spreadsheet into memory using the LoadSpreadsheet function. The elements that make up this function are:

SheetIndex: The specific Sheet Number in the selected Spreadsheet

Select a Spreadsheet on your computer and run the following to load the Spreadsheet into Memory and then this routine will count the number of columns between the first and last column used. You should note that ScreenIndex commences at 0 for the opening sheet

```
begin
  LoadSpreadSheet('c:\scripttest.xls');
  showmessage('Number of Columns is ' + intToStr(SSGetColumnCount(0)));
end.
```

4.184 SSGetContentColumnCount

For use with

> ***General Custom Scripts***
> ***Screen Data Script***
> ***Order Script***
> ***Custom Product Script***

Format: SSGetContentColumnCount(SheetIndex);

This function allows you to get information about how many columns that contain data in a defined Spreadsheet. I.e. ignore blank columns embedded in the data columns. When using this function you should first place the Spreadsheet into memory using the LoadSpreadsheet function.

The elements that make up this function are:

> **SheetIndex**: The specific Sheet Number in the selected Spreadsheet

Select a Spreadsheet on your computer and run the following to load the Spreadsheet into Memory after which this function will count the number of columns that are actually used. You should note that ScreenIndex commences at 0 for the opening sheet

```
begin
 LoadSpreadSheet('c:\scripttest.xls');
 showmessage('Number of Columns is ' + intToStr(SSGetContentColumnCount(0)));
end.
```

4.185 SSGetContentRowCount

For use with

> ***General Custom Scripts***
> ***Screen Data Script***
> ***Order Script***
> ***Custom Product Script***

Format: SSGetContentRowCount(SheetIndex);

This function allows you to get information about how many rows that contain data in a defined Spreadsheet. I.e. ignore blank rows embedded in the data rowss. When using this function you should first place the Spreadsheet into memory using the LoadSpreadsheet function. The elements that make up this function are:

> **SheetIndex**: The specific Sheet Number in the selected Spreadsheet

Select a Spreadsheet on your computer and run the following to load the Spreadsheet into Memory after which this function will count the number of rows that are actually used. You should note that ScreenIndex commences at 0 for the opening sheet

```
begin
 LoadSpreadSheet('c:\scripttest.xls');
 showmessage('Number of Rows is ' + intToStr(SSGetContentRowCount(0)));
end.
```

4.186 SSGetRowCount

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SSGetRowCount(SheetIndex);

This function allows you to get information about how many rows are being used in a defined Spreadsheet. When using this function you should first place the Spreadsheet into memory using the LoadSpreadsheet function. The elements that make up this function are:

SheetIndex: The specific Sheet Number in the selected Spreadsheet

Select a Spreadsheet on your computer and run the following to load the Spreadsheet into Memory and then this routine will count the number of rows between the first and last row used. You should note that ScreenIndex commences at 0 for the opening sheet

```
begin
  LoadSpreadSheet('c:\scripttest.xls');
  showmessage('Number of Rows is ' + intToStr(SSGetRowCount(0)));
end.
```

4.187 SSSetCellText

For use with

General Custom Scripts
Screen Data Script
Order Script
Custom Product Script

Format: SSSetCellText(Col, Row, CellText, SheetIndex);

This function allows you to populate individual Cells in a Spreadsheet with Text. When using this function you should first place the Spreadsheet into memory using the LoadSpreadsheet function. The elements that make up this function are:

Col: The Column within the spreadsheet containing the Cell. Note Columns always begin at column 0
Row: The Row within the spreadsheet containing the Cell. Note Rows always begin at row 0
CellText: The Text that you pasting into the Cell
SheetIndex: The specific Sheet Number in the selected Spreadsheet. Note Sheet Indexes always begin at sheet 0

Select a Spreadsheet on your computer and run the following to load the Spreadsheet into Memory and then this routine will add the specified Text into the defined Cell. After confirming the update the final line saves the spreadsheet currently held in memory

```
begin
  LoadSpreadSheet('c:\scripttest.xls');
  SSSetCellText(0,0,'My Added Text',0);
  SaveSpreadSheet('c:\Spreadsheet.xls');
  showmessage('Text Added and Spreadsheet saved');
```

```
end.
```

4.188 UpdateProgress

For use with
General Custom Scripts
Screen Data Script

Format: UpdateProgress(Value);

This function allows you to update the Progress Bar with the current Value. The elements that make up this function are:

Value: The variable used to increment the Progress Bar

This is used in combination with functions ***ShowProgress*** and ***EndProgress*** (and optionally ***UpdateProgressCaption***). The following example uses the four available functions related to the Progress Bar

```
            Const
              ProgressCount = 2000;
            Var
              x: Integer;
            begin
              ShowProgress('My Progress Bar',ProgressCount);
               For x := 1 to progressCount do
               Begin
                if x >= (ProgressCount / 2) then
                 Begin
                  UpdateProgressCaption('Getting There');
                 end;
                UpdateProgress(x);
               end;
              EndProgress;
              Showmessage ('Progress Display Completed');
End.
```

4.189 UpdateProgressCaption

For use with
General Custom Scripts
Screen Data Script

Format: UpdateProgressCaption(Caption);

This function allows you to update the caption on the Progress Bar to reflect the progress or activity taking place. The elements that make up this function are:

Caption: The caption that will replace the current caption at the top of the Progress panel

This is function is optionally used in combination with functions ***ShowProgress, UpdateProgress*** and ***EndProgress***. The following example uses the four available functions related to the Progress Bar

```
Const
```

```
  ProgressCount = 2000;
Var
  x: Integer;
begin
  ShowProgress('My Progress Bar',ProgressCount);
   For x := 1 to progressCount do
   Begin
    if x >= (ProgressCount / 2) then
     Begin
      UpdateProgressCaption('Getting There');
     end;
    UpdateProgress(x);
   end;
  EndProgress;
  Showmessage ('Progress Display Completed');
End.
```

4.190 WorkFlowActiveScheme

For use with

General Custom Scripts linked to Workflows

Format: **WorkflowActiveScheme:**

This function gets the current Active Scheme

This example will go to another Scheme and then display the name of that Scheme. Create a Workflow with (say) two schemes and note the scheme names. On the first Scheme add an Object.

Create the following Standard Script

```
Var
 ActiveScheme: String;

begin
  WorkflowChangeScheme('Scheme2');
  ActiveScheme:= WorkflowActiveScheme;
  Showmessage('Current Active scheme is ' + ActiveScheme);
end.
```

** Where Scheme2 is the name of the second scheme

Link the object to this script then click on the object. The second Scheme will now be displayed along with a message telling you the name of the current Scheme.

4.191 WorkFlowChangeScheme

For use with

General Custom Scripts linked to Workflows

Format: **WorkflowChangeScheme**(SchemeName);

This function allows you to change the current Workflow Screen to another that is in the current Workflow. The elements that make up this function are:

SchemeName: The name of the Scheme which you want to display

This example will show how to go to another Scheme. Create a Workflow with (say) two schemes and note the scheme names. On the first Scheme add an Object.

Create the following Standard Script

```
begin
  WorkflowChangeScheme('Scheme2');
end.
```

** Where Scheme2 is the name of the second scheme

Link the object to this script then click on the object. The second Scheme will now be displayed.

4.192 WorkFlowGetLayerVisible

For use with

General Custom Scripts linked to Workflows

Format: Variable:= WorkflowGetLayerVisible(LayerName);

Variable: Function returns 'True' or 'False'
LayerName: The name of the Layer being enquired on

This Function allows you to enquire on the current visibility status of a Layer

```
Var
OnOrOff: Boolean;
begin
  OnOrOff := WorkflowGetLayerVisible('Layer1');
  Showmessage(OnOrOff);
end.
```

4.193 WorkflowObjectLoadPicture

For use with
- ***General Custom Scripts***
- ***Screen Data Script***
- ***Order Script***
- ***Custom Product Script***

Format: WorkflowObjectLoadPicture(ObjectID, Filename);

This function allows you to add or amend the current picture that exists in a Picture Object. The elements that make up this function are:

Object ID: Right-Mouse on the Object in the Workflow to get the ID. Note: The Object must be a Picture Object
Filename: The full path pointing to the Picture

Having created an object in a workflow find out the Object ID by ‘right-mousing’ on the Object or go into OstDesigner.

This example will overwrite the current Picture Object. You should note that the Object itself must be a ‘Picture’ object

```
Begin
        WorkflowObjectLoadPicture(6,'c:\House.jpg');
        Showmessage('Picture Updated');
End.
```

4.194 WorkFlowSetLayerVisible

For use with

General Custom Scripts linked to Workflows

Format: **WorkflowSetLayerVisible**(LayerName; Visible);

LayerName: The name of the Layer to be made Visible/Invisible
Visible: Set to True or False

This Procedure turns a layer on or off across all Schemes within the Workflow.

5 'Lookup' Numbers

The Functions AskMandatoryQuestionWithLookup and AskQuestionWithLookup use a number to define the table being referenced. These number are:

Ref#	**Description**
1000	PriceLevel
1001	Supplier
1002	Standard Units
1003	Locations
1004	Items
1005	Adjustment Types
1006	Credit Terms
1007	Categories
1008	Inventory Templates
1009	Price Group
1010	Descriptors
1011	Item Batch
1012	Item Colour
1013	Item Grade
1014	Item Size
1015	Customers
1016	Employee Buyer
1017	Tax Groups
1018	Company Address
1019	Purchase Types
1020	Shipping Method
1021	Workflow Status
1022	Tax Codes
1023	Location Groups
1025	Pricing Groups
1027	Analysis Groups
1028	Customer Types
1029	Customer Regions
1030	Customer Codes
1031	Customer Lead Sources
1032	Warranty Codes
1033	Article Types
1034	Article Categories
1035	Supplier Types
1036	Supplier Regions
1037	Supplier Codes
1038	Resources
1039	Departments
1040	Employees
1041	Asset Types
1042	Kitsets

1043	Sales Types
1044	Employee Sales Person
1045	Customer Shipping Method
1046	Sales Order Delivery
1047	Descriptors (General Purpose)
1048	Deliveries Ready To Invoice
1049	Job Type
1052	Resources (Assets)
1053	Company (Assets)
1054	Projects
1055	Equipment-Customer Asset
1056	Job Tasks
1057	Task Bills
1058	Job Templates
1059	Warehouse
1061	Job Category
1062	Descriptor (TaskBill)
1063	Labour Codes
1064	Item Properties
1065	Descriptor Properties
1066	Labour Properties
1067	Contract
1069	Rate Level
1070	Warranty List
1071	List Master
1072	Articles
1073	Template Tasks
1074	Notes Categories
1075	Job Orders
1076	Contract Number
1077	Cost Centre (Excl Map Restricted)
1078	Rate Scales
1079	Project Types
1080	Service Plan Code
1081	Assembly Tracking Code
1082	Assembly Orders
1083	Routing Template
1084	Job Categories
1085	Purchase Orders
1086	Cost Centre (All)
1087	Std Properties
1088	Payment Method
1089	Payment Account
1090	Customer Deposits
1091	Credits
1092	Sales Orders
1093	Sales Orders - Counter

1094	Invoice Group
1095	Invoices
1096	Invoices- All
1097	Statement Cycles
1098	Call Classifications
1099	Call Sub Classifications
1100	Sales Orders - Deliveries
1101	Locations ReturnSysId
1103	Standard Units - Time Unit
1104	Call Resolution Codes
1105	Items - Custom
1106	Customer Asset Type
1107	Cost Groups
1108	Sales Workflow Status
1109	Invoices-excl credits and planned status
1110	Items - Non Custom
1111	Items - Assembly
1112	Item Revision
1113	Quote Lost Reasons
1114	Job Orders - open and in progress
1115	Assembly Orders - open and in progress
1117	Job Orders - Quotes
1118	Tasks
1119	Steps
1120	Kitsets From Kitset Master
1121	Service Confirm
1122	Invoices - Printed
1123	Users
1124	Manufacturers
1125	Financial batches
1127	Items- from BOMMaster - Assembly Order
1128	Purchase Orders - not closed
1129	Cost Centre - Job no debtor
1130	Call Centre Ticket ID
1131	Catalogue
1132	Scrap Codes
1133	Common text
1135	PO Shipments Workflow Status
1136	Supplier Shipping Type
1137	Currency Codes
1138	Purchase Shipments - not closed
1139	POS Station ID
1140	Payment Method - POS

6 Using Scripts within a Script

1. You can create your own library of common Functions and Procedures and hold them independently within - or outside of – Ostendo, then call these from within your Main Script. The advantage of using this feature is that you do not need to key in full script details every time you create a Script but can simply quote the name of the 'Sub' script

To set this up in the main script you are required to add two lines

```
Uses
'FunctionsLibrary';
```

Where Uses tells Ostendo that you are using
'FunctionsLibrary' is the name of the 'Sub' Script that you are calling

The first line ('Uses') MUST be the first line in the Main Script. The next line is the script name. The following example has a script called 'MainScript' within which the sub script 'FunctionsLibrary' will be called

Firstly create the Main script called (say) 'MainScript' with the following

```
uses 'FunctionsLibrary';
var
 TheCustomer,ThePhoneNumber: string;
begin
 TheCustomer := AskQuestionWithLookup('Customer', '',1015);
 {This calls a simple procedure within the FunctionsLibrary to display the Customer name
passed}
 DisplayCustomer(TheCustomer);
 {This calls a simple function within the FunctionsLibrary and returns the phone number for
the Customer passed}
 ThePhoneNumber := ReturnPhoneNumber(TheCustomer);
 showmessage('The phone number for ' + TheCustomer + ' is ' + ThePhoneNumber + '');
end.
```

Next create a Script called (say) 'FunctionsLibrary' with the following

```
procedure DisplayCustomer(PassedCustomerName:string);
begin
  showmessage(PassedCustomerName);
end;

function ReturnPhoneNumber(PassedCustomerName:string):string;
begin
  result := GetSQLResult('select CUSTOMERPHONE from CustomerMaster where
Customer = ''' + PassedCustomerName + '''');
end;

begin
end.
```

If you now run 'MainScript' you be asked to select a Customer from the Customer Master file. This is stored in Variable 'TheCustomer'.

Procedure 'DisplayCustomer' is now called from ''FunctionsLibrary' and the variable 'TheCustomer' passed to it. This procedure will then display the passed Customer Name.

Function 'ReturnPhoneNumber' is now called from ''FunctionsLibrary' and the Customer's phone number is extracted from the CustomerMaster record of the passed Customer. This is then displayed in a Showmessage.

2. Other Options

2.1. Stroing the 'called' Script

You have three options where to store the 'called' script and this reflected in the opening line of the main Script. I.e. If you specify :

uses 'FunctionsLibrary';
    Ostendo will look for the 'FunctionsLibrary' script in Ostendo's Table CUSTOMSCRIPTS

uses 'FunctionsLibrary.txt';
    Ostendo will look for file 'FunctionsLibrary.txt' which should be in a folder (which you have previously created) called 'Functions' located directly under the Ostendo folder (example C:\Program Files\Ostendo\Functions\FunctionsLibrary.txt)

uses 'D:\Dev-X\Scripts\FunctionsLibrary.txt';
    Ostendo will look for file 'FunctionsLibrary.txt' at the location you state

2.2. Encrypted Scripts

You can use either encrypted or unencrypted scripts

7 Standard Functions and Procedures

```
function IntToStr(i: Integer): String
function FloatToStr(e: Extended): String
function DateToStr(e: Extended): String
function TimeToStr(e: Extended): String
function DateTimeToStr(e: Extended): String
function VarToStr(v: Variant): String
function StrToInt(s: String): Integer
function StrToFloat(s: String): Extended
function StrToDate(s: String): Extended
function StrToTime(s: String): Extended
function StrToDateTime(s: String): Extended
function Format(Fmt: String; Args: array): String
function FormatFloat(Fmt: String; Value: Extended): String
function FormatDateTime(Fmt: String; DateTime: TDateTime): String
function FormatMaskText(EditMask: string; Value: string): string
function EncodeDate(Year, Month, Day: Word): TDateTime
procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word)
function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime
procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word)
function Date: TDateTime
function Time: TDateTime
function Now: TDateTime
function DayOfWeek(aDate: DateTime): Integer
function IsLeapYear(Year: Word): Boolean
function DaysInMonth(nYear, nMonth: Integer): Integer
function Length(s: String): Integer
function Copy(s: String; from, count: Integer): String
function Pos(substr, s: String): Integer
function Uppercase(s: String): String
function Lowercase(s: String): String
function Trim(s: String): String
function NameCase(s: String): String
function CompareText(s, s1: String): Integer
function Chr(i: Integer): Char
function Ord(ch: Char): Integer
procedure SetLength(var S: String; L: Integer)
function Round(e: Extended): Integer
function Trunc(e: Extended): Integer
function Int(e: Extended): Integer
function Frac(X: Extended): Extended
function Sqrt(e: Extended): Extended
function Abs(e: Extended): Extended
function Sin(e: Extended): Extended
function Cos(e: Extended): Extended
function ArcTan(X: Extended): Extended
function Tan(X: Extended): Extended
function Exp(X: Extended): Extended
function Ln(X: Extended): Extended
function Pi: Extended
procedure Inc(var i: Integer; incr: Integer = 1)
procedure Dec(var i: Integer; decr: Integer = 1)
procedure ShowMessage(Msg: Variant)
function ValidInt(cInt: String): Boolean
function ValidFloat(cFlt: String): Boolean
function ValidDate(cDate: String): Boolean
```

8 Useful Scripts

This section contains useful scripts that you may wish to use. You can simply copy and paste these into Ostendo ***File>Custom Scripts***

8.1 Copy Job Templates

```
{This script enables you create a Job Template in Ostendo by copying an existing Job template}

var
 TheOldTemplate: String;
 TheNewTemplate: String;

procedure InsertTemplateMaster(TheOldTemplateCode,TheNewTemplateCode:String);
var
  TempQuery,TempQuery2: TpFIBQuery;
  TempTrans,TempTrans2: TpFIBTransaction;
  x: Integer;
  insertSQL: String;
begin
  TempTrans := TpFIBTransaction.Create(nil);
  TempTrans2 := TpFIBTransaction.Create(nil);
  TempQuery := TpFIBQuery.Create(nil);
  TempQuery2 := TpFIBQuery.Create(nil);
  try
  begin
   TempTrans.DefaultDatabase := OstendoDB;
   TempTrans2.DefaultDatabase := OstendoDB;
   TempQuery.Database := OstendoDB;
   TempQuery.Transaction := TempTrans;
   TempQuery.Options := qoStartTransaction;
   TempQuery.SQL.Clear;
   TempQuery2.Database := OstendoDB;
   TempQuery2.Transaction := TempTrans2;
   TempQuery2.Options := qoStartTransaction + qoAutoCommit;
   TempQuery2.SQL.Clear;
   TempQuery.SQL.ADD('select
TEMPLATECODE,TEMPLATEDESCRIPTION,TEMPLATESTATUS,TEMPLATENOTES,');
   TempQuery.SQL.ADD('CONFIGUREDBYTASK,TASKSEQUENCING from TEMPLATEMASTER
where TEMPLATECODE = ''' + TheOldTemplateCode + '''');
   TempQuery.ExecQuery;
   While not(TempQuery.eof) do
   begin
    insertSQL := 'insert into TEMPLATEMASTER
(TEMPLATECODE,TEMPLATEDESCRIPTION,TEMPLATESTATUS,TEMPLATENOTES,' +
                 'CONFIGUREDBYTASK,TASKSEQUENCING) values (' +

':TEMPLATECODE,:TEMPLATEDESCRIPTION,:TEMPLATESTATUS,:TEMPLATENOTES,' +
                 ':CONFIGUREDBYTASK,:TASKSEQUENCING)';
    TempQuery2.SQL.ADD(insertSQL);
    TempQuery2.PN('TEMPLATECODE').asstring := TheNewTemplateCode;
    TempQuery2.PN('TEMPLATEDESCRIPTION').asstring := TempQuery.FN(
'TEMPLATEDESCRIPTION').asstring;
    TempQuery2.PN('TEMPLATESTATUS').asstring := TempQuery.FN('TEMPLATESTATUS'
).asstring;
```

```
    TempQuery2.PN('TEMPLATENOTES').asstring := TempQuery.FN('TEMPLATENOTES'
).asstring;
    TempQuery2.PN('CONFIGUREDBYTASK').asstring := TempQuery.FN(
'CONFIGUREDBYTASK').asstring;
    TempQuery2.PN('TASKSEQUENCING').asstring := TempQuery.FN('TASKSEQUENCING'
).asstring;
    TempQuery2.ExecQuery;
    TempQuery.next;
  end;
 end;
 finally
   TempTrans.Free;
   TempTrans2.Free;
   TempQuery.Close;
   TempQuery2.Close;
   TempQuery.Free;
   TempQuery2.Free;
 end;
end;

procedure InsertTemplateTasks(TheOldTemplateCode,TheNewTemplateCode:String);
var
 TempQuery,TempQuery2: TpFIBQuery;
 TempTrans,TempTrans2: TpFIBTransaction;
 x: Integer;
 insertSQL: String;
begin
 TempTrans := TpFIBTransaction.Create(nil);
 TempTrans2 := TpFIBTransaction.Create(nil);
 TempQuery := TpFIBQuery.Create(nil);
 TempQuery2 := TpFIBQuery.Create(nil);
 try
 begin
  TempTrans.DefaultDatabase := OstendoDB;
  TempTrans2.DefaultDatabase := OstendoDB;
  TempQuery.Database := OstendoDB;
  TempQuery.Transaction := TempTrans;
  TempQuery.Options := qoStartTransaction;
  TempQuery.SQL.Clear;
  TempQuery2.Database := OstendoDB;
  TempQuery2.Transaction := TempTrans2;
  TempQuery2.Options := qoStartTransaction + qoAutoCommit;
  TempQuery.SQL.ADD('select
TASKSEQUENCE,TASKNAME,TASKDESCRIPTION,TASKTOJOBLINES,');
  TempQuery.SQL.ADD(
'TASKBILLCODE,TASKBILLDESCRIPTION,TASKBILLUNIT,TASKBILLQTY,');
  TempQuery.SQL.ADD(
'TASKINSTRUCTIONS,TASKESTIMATEDDURATION,DURATIONSCALE,DEPARTMENTCODE'
);
  TempQuery.SQL.ADD(' from TEMPLATETASKS where TEMPLATECODE = ''' +
TheOldTemplateCode + '''');
  TempQuery.ExecQuery;
  While not(TempQuery.eof) do
  begin
    InsertSQL := 'insert into TemplateTasks
(TEMPLATECODE,TASKSEQUENCE,TASKNAME,TASKDESCRIPTION,TASKTOJOBLINES,' +
                  'TASKBILLCODE,TASKBILLDESCRIPTION,TASKBILLUNIT,TASKBILLQTY,' +
'TASKINSTRUCTIONS,TASKESTIMATEDDURATION,DURATIONSCALE,DEPARTMENTCODE)
values (' +

':TEMPLATECODE,:TASKSEQUENCE,:TASKNAME,:TASKDESCRIPTION,:TASKTOJOBLINES,'
+
              ':TASKBILLCODE,:TASKBILLDESCRIPTION,:TASKBILLUNIT,:TASKBILLQTY,' +

':TASKINSTRUCTIONS,:TASKESTIMATEDDURATION,:DURATIONSCALE,:DEPARTMENTCOD
E)';
    TempQuery2.SQL.Clear;
    TempQuery2.SQL.ADD(insertSQL);
    TempQuery2.PN('TEMPLATECODE').asstring := TheNewTemplateCode;
    TempQuery2.PN('TASKSEQUENCE').asinteger := TempQuery.FN('TASKSEQUENCE'
).asinteger;
    TempQuery2.PN('TASKNAME').asstring := TempQuery.FN('TASKNAME').asstring;
    TempQuery2.PN('TASKDESCRIPTION').asstring := TempQuery.FN('TASKDESCRIPTION'
).asstring;
    TempQuery2.PN('TASKTOJOBLINES').asstring := TempQuery.FN('TASKTOJOBLINES'
).asstring;
    TempQuery2.PN('TASKBILLCODE').asstring := TempQuery.FN('TASKBILLCODE').asstring;
    TempQuery2.PN('TASKBILLDESCRIPTION').asstring := TempQuery.FN(
'TASKBILLDESCRIPTION').asstring;
    TempQuery2.PN('TASKBILLUNIT').asstring := TempQuery.FN('TASKBILLUNIT').asstring;
    TempQuery2.PN('TASKBILLQTY').value := TempQuery.FN('TASKBILLQTY').value;
    TempQuery2.PN('TASKINSTRUCTIONS').asstring := TempQuery.FN('TASKINSTRUCTIONS'
).asstring;
    TempQuery2.PN('TASKESTIMATEDDURATION').value := TempQuery.FN(
'TASKESTIMATEDDURATION').value;
    TempQuery2.PN('DURATIONSCALE').asstring := TempQuery.FN('DURATIONSCALE'
).asstring;
    TempQuery2.PN('DEPARTMENTCODE').asstring := TempQuery.FN('DEPARTMENTCODE'
).asstring;
    TempQuery2.ExecQuery;
    TempQuery.next;
  end;
 end;
 finally
   TempTrans.Free;
   TempTrans2.Free;
   TempQuery.Close;
   TempQuery2.Close;
   TempQuery.Free;
   TempQuery2.Free;
 end;
end;

procedure InsertTemplateResources(TheOldTemplateCode,TheNewTemplateCode:String);
var
 TempQuery,TempQuery2: TpFIBQuery;
 TempTrans,TempTrans2: TpFIBTransaction;
 x: Integer;
 insertSQL: String;
begin
 TempTrans := TpFIBTransaction.Create(nil);
 TempTrans2 := TpFIBTransaction.Create(nil);
 TempQuery := TpFIBQuery.Create(nil);
 TempQuery2 := TpFIBQuery.Create(nil);
 try
 begin
  TempTrans.DefaultDatabase := OstendoDB;
  TempTrans2.DefaultDatabase := OstendoDB;
  TempQuery.Database := OstendoDB;
  TempQuery.Transaction := TempTrans;
  TempQuery.Options := qoStartTransaction;
  TempQuery.SQL.Clear;
  TempQuery2.Database := OstendoDB;
  TempQuery2.Transaction := TempTrans2;
  TempQuery2.Options := qoStartTransaction + qoAutoCommit;
  TempQuery.SQL.ADD('select TASKNAME,RESOURCETYPE, RESOURCENAME');
  TempQuery.SQL.ADD(' from TEMPLATERESOURCES where TEMPLATECODE = ''' +
TheOldTemplateCode + '''');
  TempQuery.ExecQuery;
  While not(TempQuery.eof) do
  begin
    InsertSQL := 'insert into TEMPLATERESOURCES
(TEMPLATECODE,TASKNAME,RESOURCETYPE, RESOURCENAME) values (' +
                 ':TEMPLATECODE,:TASKNAME,:RESOURCETYPE, :RESOURCENAME)';
    TempQuery2.SQL.Clear;
    TempQuery2.SQL.ADD(insertSQL);
    TempQuery2.PN('TEMPLATECODE').asstring := TheNewTemplateCode;
    TempQuery2.PN('TASKNAME').asstring := TempQuery.FN('TASKNAME').asstring;
    TempQuery2.PN('RESOURCETYPE').asstring := TempQuery.FN('RESOURCETYPE')
.asstring;
    TempQuery2.PN('RESOURCENAME').asstring := TempQuery.FN('RESOURCENAME'
).asstring;
    TempQuery2.ExecQuery;
    TempQuery.next;
  end;
 end;
 finally
   TempTrans.Free;
   TempTrans2.Free;
   TempQuery.Close;
   TempQuery2.Close;
   TempQuery.Free;
   TempQuery2.Free;
 end;
end;

procedure InsertTemplateLines(TheOldTemplateCode,TheNewTemplateCode:String);
var
 TempQuery,TempQuery2: TpFIBQuery;
 TempTrans,TempTrans2: TpFIBTransaction;
 x: Integer;
 insertSQL: String;
begin
 TempTrans := TpFIBTransaction.Create(nil);
 TempTrans2 := TpFIBTransaction.Create(nil);
 TempQuery := TpFIBQuery.Create(nil);
 TempQuery2 := TpFIBQuery.Create(nil);
 try
 begin
  TempTrans.DefaultDatabase := OstendoDB;
```

```
   TempTrans2.DefaultDatabase := OstendoDB;
   TempQuery.Database := OstendoDB;
   TempQuery.Transaction := TempTrans;
   TempQuery.Options := qoStartTransaction;
   TempQuery.SQL.Clear;
   TempQuery2.Database := OstendoDB;
   TempQuery2.Transaction := TempTrans2;
   TempQuery2.Options := qoStartTransaction + qoAutoCommit;
   TempQuery.SQL.ADD('select
TASKNAME,LINENUMBER,CODETYPE,LINECODE,LINEDESCRIPTION, LINEUNIT,');
   TempQuery.SQL.ADD('TEMPLATEQTY,LINENOTES ');
   TempQuery.SQL.ADD(' from TEMPLATELINES where TEMPLATECODE = ''' +
TheOldTemplateCode + '''');
   TempQuery.ExecQuery;
   While not(TempQuery.eof) do
   begin
     InsertSQL := 'insert into TemplateLines
(TEMPLATECODE,TASKNAME,LINENUMBER,CODETYPE,LINECODE,LINEDESCRIPTION,
LINEUNIT,' +
                 'TEMPLATEQTY,LINENOTES) values (' +

':TEMPLATECODE,:TASKNAME,:LINENUMBER,:CODETYPE,:LINECODE,:LINEDESCRIPTION,
:LINEUNIT,' +
                 ':TEMPLATEQTY,:LINENOTES)';
     TempQuery2.SQL.Clear;
     TempQuery2.SQL.ADD(insertSQL);
     TempQuery2.PN('TEMPLATECODE').asstring := TheNewTemplateCode;
     TempQuery2.PN('TASKNAME').asstring := TempQuery.FN('TASKNAME').asstring;
     TempQuery2.PN('LINENUMBER').asinteger := TempQuery.FN('LINENUMBER').asinteger;
     TempQuery2.PN('CODETYPE').asstring := TempQuery.FN('CODETYPE').asstring;
     TempQuery2.PN('LINECODE').asstring := TempQuery.FN('LINECODE').asstring;
     TempQuery2.PN('LINEDESCRIPTION').asstring := TempQuery.FN('LINEDESCRIPTION
').asstring;
     TempQuery2.PN('LINEUNIT').asstring := TempQuery.FN('LINEUNIT').asstring;
     TempQuery2.PN('TEMPLATEQTY').value := TempQuery.FN('TEMPLATEQTY').value;
     TempQuery2.PN('LINENOTES').asstring := TempQuery.FN('LINENOTES').asstring;
     TempQuery2.ExecQuery;
     TempQuery.next;
   end;
  end;
  finally
    TempTrans.Free;
    TempTrans2.Free;
    TempQuery.Close;
    TempQuery2.Close;
    TempQuery.Free;
    TempQuery2.Free;
  end;
end;

begin
 TheOldTemplate := AskMandatoryQuestionWithLookup('Old Template','Please select an Existing
Template to Copy',1058);
 TheNewTemplate := AskMandatoryQuestion('New Template','TEXT','Please Enter the New
Template Code','');

 InsertTemplateMaster(TheOldTemplate,TheNewTemplate);
 InsertTemplateTasks(TheOldTemplate,TheNewTemplate);
 InsertTemplateResources(TheOldTemplate,TheNewTemplate);
 InsertTemplateLines(TheOldTemplate,TheNewTemplate);
end.
```

8.2 Distinct Value Check

```
{This script allows you to interrogate any column in a Spreadsheet and returns the distinct values
within that column.  This is very useful when importing data into Ostendo where certain fields in
Ostendo should be pre-populated with values from the spreadsheet.}

var
  OpenDlg: TOpenDialog;
  x,ColumnNumber: Integer;
  ImportFile: String;
  OutputList: TStringlist;
  TempList: TStringList;

function GetImportFile: string;
var
  OpenDlg: TOpenDialog;
begin
  OpenDlg := TOpenDialog.Create(Nil);
  try
    OpenDlg.InitialDir := 'C:\';
    OpenDlg.Filter := 'Excel Files (*.csv)|*.csv|All Files (*.*)|*.*';
    if OpenDlg.Execute then
      Result := OpenDlg.filename
    else
      Result := '';
  finally
   OpenDlg.Free;
  end;
end;


{Main Code Section}
begin
  OutputList := TStringList.Create;
  TempList := TStringList.create;
  ImportFile := GetImportFile;
  if trim(ImportFile) = '' then exit;
  try
    ColumnNumber := AskQuestion('Column Number','INTEGER','Please enter the Column
Number','');
    TempList.LoadFromFile(ImportFile);
    for x := 0 to TempList.Count - 1 do
     begin
      if OutputList.IndexOf(trim(ParseString(TempList.Strings[x],',',(ColumnNumber - 1)))) < 0 then
       begin
        OutputList.Add(trim(ParseString(TempList.Strings[x],',',(ColumnNumber - 1))));
       end;
     end;
    OutputList.SaveToFile('c:\DistinctValues.txt');
    run('c:\DistinctValues.txt');
  finally
    TempList.free;
    OutputList.free;
  end;
end.
```

8.3 Key Daily Statistics

```
{This Example generates Key daily statistics from your database and either displays it on your
screen or emails it to nominated recipients

The KPI's contained in this script are split into 6 sections
        1 = Job Order Statistics
        2 = Sales Orders Statistics
        3 = POS Statistics
        4 = Invoice Statistics
        5 = Debtor Statistics
        6 = Inventory and WIP Statistics

You can define an unlimited number of Email recipients and, against each recipient, nominate
which of the above sections they are allowed to see.

Other points regarding this script.

1. You can run the script and have the results returned to your screen.  This useful if you wish to
view the results when amending the KPIs.  This is the method currently set in the script below.

2.  As scripts can be run from the CMD line on your PC this script can be scheduled to
automatically run at any time day or night.  In this example it is assumed that it will be run after
midnight and shows statistics from the previous day.  }

var {Variable section}
 EmailList: TStringlist;
 TheKPIOptions,KPINumber,DailySalesOrderCount,DailyJobOrderCount: String;
 x,y,z,TheOptionsLength: Integer;
 TheUserEmail,TheEmailBody,DailySalesOrderValue,CurrentDebtorBalance : String;
 CurrentStockValue, TheSQL, TheAgingID, AgingMethod: string;
Period1Caption,Period2Caption,Period3Caption,Period4Caption,Period5Caption: string;
PeriodACaption,PeriodBCaption,OverdueSalesValue,DailyPaymentsIn: string;
 Period1DebtorBalance, Period2DebtorBalance, Period3DebtorBalance: string;
 Period4DebtorBalance, Period5DebtorBalance, TotalDebtorBalance: string;
 DailyDepositsTaken,DailyAvgSalesOrderValue: string;
 DailyJobOrderValue,DailyAvgJobOrderValue: string;
  CurrentAssemblyWIP,CurrentJobWIP: string;
DailySalesInvoiceValue,DailyAvgSalesInvoiceValue,DailySalesInvoiceCount: string;
DailyPOSValue,DailyAvgPOSValue,DailyPOSCount: string;
 TheCompanyName: string;
 AgingRun : boolean;
Const {Constant section}
 {You will need to change the Email Host name and the sender address}

 TheEmailHostName = 'mail.development-x.com';
 TheEmailSenderAddress = 'KPI@development-x.com';
 TheEmailSubject = 'Ostendo Key Daily Statistics for ' + datetostr(date - 1);

 {The constant below allows testing of this script without emailing
 If set to False a popup message will display for each email recipient
 rather than actually emailing, otherwise if it is True emails will be
 sent to each recipient}

 EmailTheResult = false;

procedure CreateKPIEmailsAndSend;
begin
{Email Section with KPI routines per recepient}
 try
  EmailList := TStringList.Create;
  {All email recepients are defined based on the follwoing structure.
  The first field is the email address with a colon (:) seperating it
  from the Daily statistic groups required for that recepient. Each
  statistic group number is seperated by a comma (,). An example of
  the syntax is below. Many recepients with varying statistic group
  options can be defined}
  EmailList.add('info@development-x.com:1,2,3,4,5,6'); {All Groups}
  EmailList.add('sales@development-x.com:2,4,5,6'); {Typical distribution Groups}
  {The group numbers are as follows:
  1 = Job Order Statistics
  2 = Sales Orders Statistics
  3 = POS Statistics
  4 = Invoice Statistics
  5 = Debtor Statistics
  6 = Inventory and WIP Statistics}


  AgingRun := false;

  for x := 0 to EmailList.count - 1 do
  begin
    TheUserEmail := ParseString(EmailList.strings[x], ':', 0);
    TheKPIOptions :=  ParseString(EmailList.strings[x], ':', 1);

    for z := 0 to NoOfKPIS do
     begin
      KPINumber := ParseString(TheKPIOptions, ',', z);

      case KPINumber of
      '1': {Job Order Statistics}
        begin
         TheEmailBody := TheEmailBody + #13 + #13 + 'Job Order Statistics';
         TheEmailBody := TheEmailBody + #13 + '===================================';

         if GetSQLResult('select count(sysuniqueid) from JobHeader where (orderdate + 1)= ''now''')
= null then
          begin
           DailyJobOrderCount := '0';
          end
         else
          begin
           DailyJobOrderCount := GetSQLResult('select count(sysuniqueid) from JobHeader where
(orderdate + 1)= ''now''');
```

```
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily Job Order Count: ' + DailyJobOrderCount;

        if GetSQLResult('select avg(ORIGINALORDERAMOUNT) from JobHeader where (orderdate
+ 1)= "now"') = null then
         begin
          DailyAvgJobOrderValue := '0';
         end
        else
         begin
          DailyAvgJobOrderValue := GetSQLResult('select avg(ORIGINALORDERAMOUNT) from
JobHeader where (orderdate + 1)= "now"');
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily Average Order Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyAvgJobOrderValue));

        if GetSQLResult('select sum(ORIGINALORDERAMOUNT) from JobHeader where
(orderdate + 1)= "now"') = null then
         begin
          DailyJobOrderValue := '0';
         end
        else
         begin
          DailyJobOrderValue := GetSQLResult('select sum(ORIGINALORDERAMOUNT) from
JobHeader where (orderdate + 1)= "now"');
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily Job Order Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyJobOrderValue));
       end;
     '2': {Sales Order Statistics}
       begin
        TheEmailBody := TheEmailBody + #13 + #13 + 'Sales Order Statistics';
        TheEmailBody := TheEmailBody + #13 + '====================================';

        if GetSQLResult('select count(sysuniqueid) from SalesHeader where (orderdate + 1)= "now"'
) = null then
         begin
          DailySalesOrderCount := '0';
         end
        else
         begin
          DailySalesOrderCount := GetSQLResult('select count(sysuniqueid) from SalesHeader
where (orderdate + 1)= "now"');
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily Sales Order Count: ' +
DailySalesOrderCount;

        if GetSQLResult('select avg(ORIGINALORDERAMOUNT) from SalesHeader where
(orderdate + 1)= "now"') = null then
         begin
          DailyAvgSalesOrderValue := '0';
         end
        else
         begin
          DailyAvgSalesOrderValue := GetSQLResult('select avg(ORIGINALORDERAMOUNT) from
SalesHeader where (orderdate + 1)= "now"');
```

```
          end;
         TheEmailBody := TheEmailBody + #13 + ' Daily Average Order Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyAvgSalesOrderValue));

         if GetSQLResult('select sum(ORIGINALORDERAMOUNT) from SalesHeader where
(orderdate + 1)= "now"') = null then
          begin
           DailySalesOrderValue := '0';
          end
         else
          begin
           DailySalesOrderValue := GetSQLResult('select sum(ORIGINALORDERAMOUNT) from
SalesHeader where (orderdate + 1)= "now"');
          end;
         TheEmailBody := TheEmailBody + #13 + ' Daily Sales Order Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailySalesOrderValue));

         if GetSQLResult('select sum(REMAININGQTY * ORDERUNITPRICE) from
SalesLines,SalesHeader where (salesLines.Requireddate + 1) <= "now" and
SalesLines.Linestatus = "Open" and SalesLines.OrderNumber = SalesHeader.Ordernumber and
SalesHeader.orderstatus <> "Closed" and SalesHeader.orderstatus <> "Lost" and
SalesHeader.orderstatus <> "Quote" and SalesHeader.orderstatus <> "Planned"') = null then
          begin
           OverdueSalesValue := '0';
          end
         else
          begin
           OverdueSalesValue := GetSQLResult('select sum(REMAININGQTY * ORDERUNITPRICE)
from SalesLines,SalesHeader where (salesLines.Requireddate + 1) <= "now" and
SalesLines.Linestatus = "Open" and SalesLines.OrderNumber = SalesHeader.Ordernumber and
SalesHeader.orderstatus <> "Closed" and SalesHeader.orderstatus <> "Lost" and
SalesHeader.orderstatus <> "Quote" and SalesHeader.orderstatus <> "Planned"');
          end;
         TheEmailBody := TheEmailBody + #13 + ' Overdue Sales Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(OverdueSalesValue));
        end;
      '3': {POS Statistics}
        begin
         TheEmailBody := TheEmailBody + #13 + #13 + 'POS Statistics';
         TheEmailBody := TheEmailBody + #13 + '==================================';

         if GetSQLResult('select count(sysuniqueid) from POSHeader where (saledate + 1)= "now"
and (SALESTATUS = "Invoiced" or SALESTATUS = "OnHold")') = null then
          begin
           DailyPOSCount := '0';
          end
         else
          begin
           DailyPOSCount := GetSQLResult('select count(sysuniqueid) from POSHeader where
(saledate + 1)= "now" and (SALESTATUS = "Invoiced" or SALESTATUS = "OnHold")');
          end;
         TheEmailBody := TheEmailBody + #13 + ' Daily POS Count: ' + DailyPOSCount;

         if GetSQLResult('select avg(SALENETTAMOUNT) from POSHeader where (saledate + 1)=
"now" and (SALESTATUS = "Invoiced" or SALESTATUS = "OnHold")') = null then
          begin
          DailyAvgPOSValue := '0';
         end
        else
         begin
          DailyAvgPOSValue := GetSQLResult('select avg(SALENETTAMOUNT) from POSHeader
where (saledate + 1)= "now" and (SALESTATUS = "Invoiced" or SALESTATUS = "OnHold")');
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily Average POS Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyAvgPOSValue));

        if GetSQLResult('select sum(SALENETTAMOUNT) from POSHeader where (saledate + 1)=
"now" and (SALESTATUS = "Invoiced" or SALESTATUS = "OnHold")') = null then
         begin
          DailyPOSValue := '0';
         end
        else
         begin
          DailyPOSValue := GetSQLResult('select sum(SALENETTAMOUNT) from POSHeader
where (saledate + 1)= "now" and (SALESTATUS = "Invoiced" or SALESTATUS = "OnHold")');
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily POS Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyPOSValue));
       end;
      '4': {Invoice Statistics}
       begin
        TheEmailBody := TheEmailBody + #13 + #13 + 'Sales Invoice Statistics';
        TheEmailBody := TheEmailBody + #13 + '==================================';

        if GetSQLResult('select count(sysuniqueid) from SalesInvoiceHeader where (invoicedate +
1)= "now"') = null then
         begin
          DailySalesInvoiceCount := '0';
         end
        else
         begin
          DailySalesInvoiceCount := GetSQLResult('select count(sysuniqueid) from
SalesInvoiceHeader where (invoicedate + 1)= "now"');
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily Sales Invoice Count: ' +
DailySalesInvoiceCount;

        if GetSQLResult('select avg(INVOICENETTAMOUNT) from SalesInvoiceHeader where
(invoicedate + 1)= "now"') = null then
         begin
          DailyAvgSalesInvoiceValue := '0';
         end
        else
         begin
          DailyAvgSalesInvoiceValue := GetSQLResult('select avg(INVOICENETTAMOUNT) from
SalesInvoiceHeader where (invoicedate + 1)= "now"');
         end;
        TheEmailBody := TheEmailBody + #13 + ' Daily Average Invoice Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyAvgSalesInvoiceValue));

        if GetSQLResult('select sum(INVOICENETTAMOUNT) from SalesInvoiceHeader where
(invoicedate + 1)= "now"') = null then
```

```
          begin
           DailySalesInvoiceValue := '0';
          end
         else
          begin
           DailySalesInvoiceValue := GetSQLResult('select sum(INVOICENETTAMOUNT) from
SalesInvoiceHeader where (invoicedate + 1)= ''now''');
          end;
         TheEmailBody := TheEmailBody + #13 + ' Daily Sales Invoice Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailySalesInvoiceValue));
        end;
      '5': {Debtor Statistics}
        begin
         TheEmailBody := TheEmailBody + #13 +  #13 + 'Debtor and Aging Statistics';
         TheEmailBody := TheEmailBody + #13 + '==================================';

         if GetSQLResult('select sum(PAYMENTAMOUNT) from CustomerPayments where
(paymentdate + 1)= ''now'' and PAYMENTSTYLE = ''Received Payment'' ') = null then
          begin
           DailyPaymentsIn := '0';
          end
         else
          begin
           DailyPaymentsIn :=  GetSQLResult('select sum(PAYMENTAMOUNT) from
CustomerPayments where (paymentdate + 1)= ''now'' and PAYMENTSTYLE = ''Received
Payment'' ');
          end;
         TheEmailBody := TheEmailBody + #13 + ' Daily Payments In: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyPaymentsIn));

         if GetSQLResult('select sum(DEPOSITAMOUNT) from CustomerDeposits where
(depositdate + 1)= ''now''') = null then
          begin
           DailyDepositsTaken := '0';
          end
         else
          begin
           DailyDepositsTaken :=  GetSQLResult('select sum(DEPOSITAMOUNT) from
CustomerDeposits where (depositdate + 1)= ''now''');
          end;
         TheEmailBody := TheEmailBody + #13 + ' Daily Deposits Taken: ' +
FormatFloat('$###,###,##0.00',strtofloat(DailyDepositsTaken));

         if (AgingRun = False) then
          begin
           executeSQL('delete from CustomerAging');
           executeSQL('insert into CustomerAging (AGINGASATDATE) values ((cast(''NOW'' as date)
-1))');
           TheAgingID := GetSQLResult('select max(sysuniqueid) from CustomerAging');
           executeSQL('execute procedure CREATE_CUSTOMERAGING (cast(''NOW'' as date) -1) ,'
+ TheAgingID + ',0,'''','''','''','''','''''');
           {Get the Aging Headings}
           AgingMethod := GetSQLResult('select CALCULATEDFROM from AGINGPERIODS');
           if (AgingMethod = 'Monthly') then
            begin
             Period1Caption := GetSQLResult('select MONTHLYCAPTIONONE from CustomerAging');
             Period2Caption := GetSQLResult('select MONTHLYCAPTIONTWO from CustomerAging'
);
             Period3Caption := GetSQLResult('select MONTHLYCAPTIONTHREE from
CustomerAging');
             Period4Caption := GetSQLResult('select MONTHLYCAPTIONFOUR from CustomerAging'
);
             Period5Caption := GetSQLResult('select MONTHLYCAPTIONFIVE from CustomerAging');
            end
           else
            begin
             Period1Caption := 'Current';
             Period2Caption := GetSQLResult('select PERIOD2 from AGINGPERIODS');
             Period2Caption := '1 to ' + Period2Caption;
             PeriodACaption := GetSQLResult('select PERIOD2 from AGINGPERIODS');
             PeriodBCaption := GetSQLResult('select PERIOD3 from AGINGPERIODS');
             Period3Caption := PeriodACaption + ' to ' + PeriodBCaption;
             PeriodACaption := GetSQLResult('select PERIOD3 from AGINGPERIODS');
             PeriodBCaption := GetSQLResult('select PERIOD4 from AGINGPERIODS');
             Period4Caption := PeriodACaption + ' to ' + PeriodBCaption;
             PeriodACaption := GetSQLResult('select PERIOD4 from AGINGPERIODS');
             Period5Caption := PeriodACaption + '+';
            end;
          end;
         AgingRun := True;

         if GetSQLResult('select sum(TOTALBALANCE) from CUSTOMERAGINGHEADER') = null
then
          begin
           TotalDebtorBalance := '0';
          end
         else
          begin
           TotalDebtorBalance := GetSQLResult('select sum(TOTALBALANCE) from
CUSTOMERAGINGHEADER');
          end;
         TheEmailBody := TheEmailBody + #13 + ' Total Debtor Balance: ' +
FormatFloat('$###,###,##0.00',strtofloat(TotalDebtorBalance));

         if GetSQLResult('select sum(CURRENTBALANCE) from CUSTOMERAGINGHEADER') =
null then
          begin
           Period1DebtorBalance := '0';
          end
         else
          begin
           Period1DebtorBalance := GetSQLResult('select sum(CURRENTBALANCE) from
CUSTOMERAGINGHEADER');
          end;
         TheEmailBody := TheEmailBody + #13 + ' * ' + Period1Caption + ' Debtor Balance: ' +
FormatFloat('$###,###,##0.00',strtofloat(Period1DebtorBalance));

         if GetSQLResult('select sum(PERIOD1BALANCE) from CUSTOMERAGINGHEADER') =
null then
          begin
           Period2DebtorBalance := '0';
          end
```

```
        else
         begin
          Period2DebtorBalance := GetSQLResult('select sum(PERIOD1BALANCE) from
CUSTOMERAGINGHEADER');
         end;
        TheEmailBody := TheEmailBody + #13 + ' * ' + Period2Caption + ' Debtor Balance: ' +
FormatFloat('$###,###,##0.00',strtofloat(Period2DebtorBalance));

        if GetSQLResult('select sum(PERIOD2BALANCE) from CUSTOMERAGINGHEADER') =
null then
         begin
          Period3DebtorBalance := '0';
         end
        else
         begin
          Period3DebtorBalance := GetSQLResult('select sum(PERIOD2BALANCE) from
CUSTOMERAGINGHEADER');
         end;
        TheEmailBody := TheEmailBody + #13 + ' * ' + Period3Caption + ' Debtor Balance: ' +
FormatFloat('$###,###,##0.00',strtofloat(Period3DebtorBalance));

        if GetSQLResult('select sum(PERIOD3BALANCE) from CUSTOMERAGINGHEADER') =
null then
         begin
          Period4DebtorBalance := '0';
         end
        else
         begin
          Period4DebtorBalance := GetSQLResult('select sum(PERIOD3BALANCE) from
CUSTOMERAGINGHEADER');
         end;
        TheEmailBody := TheEmailBody + #13 + ' * ' + Period4Caption + ' Debtor Balance: ' +
FormatFloat('$###,###,##0.00',strtofloat(Period4DebtorBalance));

        if GetSQLResult('select sum(PERIOD4BALANCE) from CUSTOMERAGINGHEADER') =
null then
         begin
          Period5DebtorBalance := '0';
         end
        else
         begin
          Period5DebtorBalance := GetSQLResult('select sum(PERIOD4BALANCE) from
CUSTOMERAGINGHEADER');
         end;
        TheEmailBody := TheEmailBody + #13 + ' * ' + Period5Caption + ' Debtor Balance: ' +
FormatFloat('$###,###,##0.00',strtofloat(Period5DebtorBalance));
       end;
     '6': {Inventory and WIP Statistics}
       begin
        TheEmailBody := TheEmailBody + #13 +  #13 + 'Inventory and WIP Statistics';
        TheEmailBody := TheEmailBody + #13 + '===================================';

        TheSQL := 'select sum(inventory.inventoryqty * itemmaster.averagecost *
itemunits.conversionfactor)' +
                  ' from WAREHOUSEMASTER, INVENTORY,ITEMMASTER,ITEMUNITS' +
                  ' where INVENTORY.warehousecode = warehousemaster.warehousecode' +
                 ' and Itemmaster.itemcode = inventory.itemcode and ' +
                 ' itemunits.itemcode = itemmaster.itemcode and ' +
                 ' itemunits.tounit = inventory.inventoryunit ';
        if GetSQLResult(TheSQL) = null then
         begin
          CurrentStockValue := '0';
         end
        else
         begin
          CurrentStockValue := GetSQLResult(TheSQL);
         end;
        TheEmailBody := TheEmailBody + #13 + ' Current Stock Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(CurrentStockValue));

        TheSQL := 'select sum(currentwipvalue)' +
                 ' from JOBHEADER' +
                 ' where orderstatus <> ''Closed''';
        if GetSQLResult(TheSQL) = null then
         begin
          CurrentJobWIP := '0';
         end
        else
         begin
          CurrentJobWIP := GetSQLResult(TheSQL);
         end;
        TheEmailBody := TheEmailBody + #13 + ' Current Job WIP Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(CurrentJobWIP));

        TheSQL := 'select sum(wipvalue)' +
                 ' from ASSEMBLYHEADER' +
                 ' where orderstatus <> ''Closed''';
        if GetSQLResult(TheSQL) = null then
         begin
          CurrentAssemblyWIP := '0';
         end
        else
         begin
          CurrentAssemblyWIP := GetSQLResult(TheSQL);
         end;
        TheEmailBody := TheEmailBody + #13 + ' Current Assembly WIP Value: ' +
FormatFloat('$###,###,##0.00',strtofloat(CurrentAssemblyWIP));

       end;

     end;
    end;

    {Footer Note}
    TheEmailBody := TheEmailBody + #13 + #13 + 'Statistics were generated at: ' +
datetimetostr(now) + ' for ' + datetostr(date-1);

    if EmailTheResult = true then
     begin

SendEmailMessage(TheEmailHostName,TheEmailSenderAddress,TheUserEmail,TheEmailSubje
ct, TheEmailBody);
```

```
      TheEmailBody := '';
     end
    else
     begin
      showmessage(TheUserEmail + ':' + TheEmailSubject + ':' + TheEmailBody);
      TheEmailBody := '';
     end;

  end;

 finally
  EmailList.free;
 end;

end;

function NoOfKPIS:integer;
var NumberOfKPIs: integer;
begin
    TheOptionsLength := length(TheKPIOptions);
    NumberOfKPIs := 0;
    for y := 1 to TheOptionsLength do
     begin
      if (copy(TheKPIOptions,y,1)) = ',' then
       begin
         NumberOfKPIs := NumberOfKPIs + 1;
       end;
     end;
    Result := NumberOfKPIs;
end;

begin
  CreateKPIEmailsAndSend;
end.
```

8.4 Re-Allocate Purchase Invoice Lines

```
{This is a Related Menu script allows you to call up an existing Purchase Invoice and amend all
line Allocations.  This will remove any existing Allocations form the current Invoice Lines and
replace it with your new selection.   At the same time it will go to the source of that allocation and
amend the source.}

var
 TheJobNumber,JobTaskCount,TheJobTask: string;
 TheProductCode, TheCodeType, TheProductUnit: string;
 TheProductDescription, TheCatalogueName: string;
 TheAllocID,TheSourceID: double;
 TheInvoiceBatchNo,InvoiceCount: string;

procedure UpdatePurchaseInvoiceLines;
var
  TempQuery: TpFIBQuery;
  TempTrans: TpFIBTransaction;
begin
  TempTrans := TpFIBTransaction.Create(nil);
  TempQuery := TpFIBQuery.Create(nil);
  try
  begin
   TempTrans.DefaultDatabase := OstendoDB;
   TempQuery.Database := OstendoDB;
   TempQuery.Transaction := TempTrans;
   TempQuery.Options := qoStartTransaction;
   TempQuery.SQL.Clear;
   TempQuery.SQL.ADD('select PurchaseInvoiceAllocations.sysuniqueid,
PurchaseInvoiceLines.codetype, PurchaseInvoiceLines.linecode, PurchaseInvoiceLines.lineunit, ');
   TempQuery.SQL.ADD('linedescription, linkedcataloguename ');
   TempQuery.SQL.add(' from PurchaseInvoiceAllocations, PurchaseInvoiceLines ');
   TempQuery.SQL.add(' where PurchaseInvoiceAllocations.headersysuniqueid =
PurchaseInvoiceLines.sysuniqueid ');
   TempQuery.SQL.add(' and PurchaseInvoiceLines.invoicebatchno = ' + TheInvoiceBatchNo);
   TempQuery.ExecQuery;
   While not(TempQuery.eof) do
   begin
     TheProductCode := TempQuery.FN('linecode').asstring;
     TheCodeType := TempQuery.FN('codetype').asstring;
     TheProductUnit := TempQuery.FN('lineunit').asstring;
     TheProductDescription := TempQuery.FN('linedescription').asstring;
     TheCatalogueName := TempQuery.FN('linkedcataloguename').asstring;
     TheAllocID := TempQuery.FN('sysuniqueid').value;

     TheSourceID := AllocateToJobOrder(TheJobNumber, TheJobTask, TheProductCode,
TheCodeType, TheProductUnit, TheProductDescription, TheCatalogueName, TheAllocID);
     executeSQL('update PurchaseInvoiceAllocations set AllocationType = ''Job Order'',
AllocationReference = ''' + TheJobNumber + ''',sourcesysuniqueid = ' + floattostr(TheSourceID) + '
where sysuniqueid = ' + floattostr(TheAllocID));

    TempQuery.next;
   end;
  end;
  finally
    TempTrans.Free;
    TempQuery.Close;
    TempQuery.Free;
  end;
end;

begin
 TheInvoiceBatchNo := GetSourceFieldValue('INVOICEBATCHNO','HEADER');
 InvoiceCount := GetSQLResult('select count(sysuniqueid) from PurchaseInvoices where
INVOICEBATCHNO = ' + TheInvoiceBatchNo + ' and PurchaseInvStatus = ''InProgress''');
 if (InvoiceCount = '1') then
  begin
  {Change Allocation}
   TheJobNumber := AskMandatoryQuestionWithLookup('Job Number', 'Please select the Job
Number', 1075);
   JobTaskCount := GetSQLResult('select count(sysuniqueid) from JobTasks where OrderNumber
= ''' + TheJobNumber + '''');
   if JobTaskCount <> '1' then
    begin
     TheJobTask := AskQuestionWithUserDefinedLookup('select Taskname as "Task" from
Jobtasks where ordernumber = ''' + TheJobNumber + '''', 'Job Task','Please select the Job Task','',
'Job Tasks for ' + TheJobNumber, 'task');
```

```
    end
   else
    begin
     TheJobTask := GetSQLResult('select Taskname from Jobtasks where ordernumber = ''' +
TheJobNumber + '''');
    end;
   {Procedure to update the Allocations}
   UpdatePurchaseInvoiceLines;
   ExecuteSQL('update PurchaseInvoices set DEFAULTALLOCATIONTYPE = ''Job Order'',
DEFAULTALLOCATIONREF = ''' + TheJobNumber + ''' where INVOICEBATCHNO = ' +
TheInvoiceBatchNo);
   RelatedScreenRefreshData('HEADER');
   RelatedScreenRefreshData('LINE');
  end;
 end.
```

8.5 Upload a Report to a Website

```
{This script shows exporting an Ostendo report to a pdf file}
{then uploading to a website via ftp then viewing the pdf file}
{Wide range of uses including displaying up to date pricing}

begin
  OstendoReport('Standard Item Price List',3,'','EXPORTREPORT',OstendoPath +
'ItemPriceList.pdf','');
  SendFileFTP('www.yourftpprovider.com','ftpuser@yourftpprovider.com','ftpPassword'
,OstendoPath + 'ItemPriceList.pdf','ItemPriceList.pdf',True);
  if MessageDlg('File Uploaded, View file on Website?',mtinformation,mbYes + mbNo,0) = mrYes
then
    Run('http://www.development-x.com/demo/ItemPriceList.pdf');
end.
```

You should amend the Run('http://www.development-x.com/demo/ItemPriceList.pdf') source to suit your ftp service provider

9 Advanced Scripts

Advanced Scripts show some examples of scripting that can be used. Some knowledge of Delphi language is required for those scripts that include a display entry screen

9.1 Data Entry Script

{This script uses Delphi to create the Entry Form itself including fields such as:

- *Optional text Entry*
- *Mandatory Text Entry*
- *Date field with calendar date selection from drop-down calendar*
- *Field with selection from pre-defined options*

Any entered data can - using scripting - be held against variables and used as required.}

```
var
 frmMain: TForm;
 pnlTop: TPanel;
 pnlBottom: TPanel;
 btnClose: TButton;
 btnOK: TButton;
 lblField1: TLabel;
 lblField2: TLabel;
 lblField3: TLabel;
 lblField4: TLabel;
 edtField1: TEdit;
 edtField2: TEdit;
 dateField1: TDateTimePicker;
 ComboField1: TComboBox;

procedure OKClick;
begin
  if (edtfield1.text = '') then
   begin
    showmessage('Please enter your First Name');
    edtfield1.setfocus;
    exit;
   end;

  if (edtfield2.text = '') then
   begin
    showmessage('Please enter your Last Name');
    edtfield2.setfocus;
    exit;
   end;

  if (ComboField1.text = '') then
   begin
    showmessage('Please select either Male or Female');
    ComboField1.setfocus;
    exit;
   end;

  showmessage('Hi ' + edtfield1.text + ', This is a simple Entry Form Example');
```

```
end;

procedure Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if key = #13 then {The Enter Key}
   edtfield2.setfocus;
end;

procedure Edit2KeyPress(Sender: TObject; var Key: Char);
begin
  if key = #13 then {The Enter Key}
   dateField1.setfocus;
end;

procedure Date1KeyPress(Sender: TObject; var Key: Char);
begin
  if key = #13 then {The Enter Key}
   ComboField1.setfocus;
end;

procedure CloseClick;
begin
  frmMain.close;
end;

begin
 frmMain := TForm.Create(nil);
 try
  frmMain.width := 350;
  frmMain.height := 250;
  frmMain.caption := 'Ostendo - Data Entry Form';
  frmMain.position := poScreenCenter;
  frmMain.borderstyle := bsSizeable;
  frmMain.name := 'MainForm';

  {Top Panel}
  pnlTop := TPanel.Create(frmMain);
  pnlTop.Parent := frmMain;
  pnlTop.Align := alclient;
  pnlTop.BevelOuter :=  bvLowered;

  lblfield1 := TLabel.Create(pnlTop);
  lblfield1.Parent := pnlTop;
  lblfield1.left := 20;
  lblfield1.top := 20;
  lblfield1.caption := 'First Name';

  edtfield1 := TEdit.Create(pnlTop);
  edtfield1.Parent := pnlTop;
  edtfield1.left := 20;
  edtfield1.top := 40;
  edtfield1.width := 140;
  edtfield1.taborder := 0;
  edtfield1.onKeyPress := 'Edit1KeyPress';

  lblfield2 := TLabel.Create(pnlTop);
  lblfield2.Parent := pnlTop;
```

```
lblfield2.left := 180;
lblfield2.top := 20;
lblfield2.caption := 'Last Name';

edtfield2 := TEdit.Create(pnlTop);
edtfield2.Parent := pnlTop;
edtfield2.left := 180;
edtfield2.top := 40;
edtfield2.width := 140;
edtfield2.taborder := 1;
edtfield2.onKeyPress := 'Edit2KeyPress';

lblfield3 := TLabel.Create(pnlTop);
lblfield3.Parent := pnlTop;
lblfield3.left := 20;
lblfield3.top := 80;
lblfield3.caption := 'Birth Date';

dateField1 := TDateTimePicker.Create(pnlTop);
dateField1.Parent := pnlTop;
dateField1.left := 20;
dateField1.top := 100;
dateField1.width := 140;
dateField1.taborder := 2;
dateField1.onKeyPress := 'Date1KeyPress';

lblfield4 := TLabel.Create(pnlTop);
lblfield4.Parent := pnlTop;
lblfield4.left := 180;
lblfield4.top := 80;
lblfield4.caption := 'Male/Female';

ComboField1 := TComboBox.Create(pnlTop);
ComboField1.Parent := pnlTop;
ComboField1.left := 180;
ComboField1.top := 100;
ComboField1.width := 140;
ComboField1.taborder := 3;
ComboField1.Style := csDropDownList;
ComboField1.Items.Add('Female');
ComboField1.Items.Add('Male');

{Create the Bottom Panel with Buttons}

{Bottom Panel}
pnlBottom := TPanel.Create(frmMain);
pnlBottom.Parent := frmMain;
pnlBottom.Height := 50;
pnlBottom.width := 350;
pnlBottom.Align := albottom;
pnlBottom.BevelOuter :=  bvNone;

{OK Button}
btnOK := TButton.Create(pnlBottom);
btnOK.Parent := pnlBottom;
btnOK.left := 145;
btnOK.Top := 15;
```

```
  btnOK.Width := 90;
  btnOK.Height := 20;
  btnOK.Caption := '&OK';
  btnOK.OnClick := 'OKClick';
  btnOK.Hint := 'Accept Entries';
  btnOK.ShowHint := True;
  btnOK.taborder := 98;
  btnOK.anchors := akright;


  {Close Button}
  btnClose := TButton.Create(pnlBottom);
  btnClose.Parent := pnlBottom;
  btnClose.left := 245;
  btnClose.Top := 15;
  btnClose.Width := 90;
  btnClose.Height := 20;
  btnClose.Caption := '&Close';
  btnClose.OnClick := 'CloseClick';
  btnClose.Hint := 'Close Data Entry Screen';
  btnClose.ShowHint := True;
  btnClose.taborder := 99;
  btnClose.anchors := akright;

  frmMain.showmodal;


 finally
  frmMain.free;
 end;

end.
```

9.2 Data Grid Update Script

{This script uses Delphi to create a Grid showing multiple records into which you can update selected fields.

The practical example below is a grid that displays Item Sell and Buy Prices and allows you, in a single screen, to update selected Prices}

```
var
 frmMain: TForm;
 pnlTop: TPanel;
 pnlBottom: TPanel;
 btnClose: TButton;
 btnOK: TButton;
 lblField1: TLabel;
 lblField2: TLabel;
 lblField3: TLabel;
 lblField4: TLabel;
 edtField1: TEdit;
 edtField2: TEdit;
 dateField1: TDateTimePicker;
 ComboField1: TComboBox;
 TempQuery: TpFIBDataSet;
 TempTrans: TpFIBTransaction;
 dbgData: TDBGrid;
 TempDS: TDatasource;
 ItemDisplay,ItemPriceUpdate: String;

procedure CloseClick;
begin
  TempQuery.edit;
  TempQuery.post;
  frmMain.close;
end;

begin
 frmMain := TForm.Create(nil);
 try
  begin
  frmMain.width := 760;
  frmMain.height := 600;
  frmMain.caption := 'Ostendo - Update Item Prices Grid (Immediate Update)';
  frmMain.position := poScreenCenter;
  frmMain.borderstyle := bsSizeable;
  frmMain.name := 'MainForm';

  TempTrans := TpFIBTransaction.Create(frmMain);
  TempQuery := TpFIBDataSet.Create(frmMain);

  TempTrans.DefaultDatabase := OstendoDB;
  TempQuery.Database := OstendoDB;
  TempQuery.Transaction := TempTrans;
  TempQuery.Autocommit := true;
  ItemDisplay := 'select ItemCode as "Item Code",' +
              'itemdescription as "Description",' +
              'itemunit as "Unit",' +
              'lastcost as "Last Cost",' +
              'standardcost as "Std Cost",' +
              'averagecost as "Avg Cost",' +
              'stdbuyprice as "Buy Price",' +
              'stdsellprice as "Sell Price",' +
              'sysuniqueid from Itemmaster order by 1';
  ItemPriceUpdate := 'update Itemmaster set stdbuyprice = :"Buy Price",' +
                 'stdsellprice = :"Sell Price"' +
                 ' where sysuniqueid = :old_sysuniqueid';
  TempQuery.SelectSQL.add(ItemDisplay);
  TempQuery.UpdateSQL.add(ItemPriceUpdate);
  TempQuery.open;

  TFloatField(TempQuery.FN('Last Cost')).currency := true;
  TFloatField(TempQuery.FN('Avg Cost')).currency := true;
  TFloatField(TempQuery.FN('Std Cost')).currency := true;
  TFloatField(TempQuery.FN('Buy Price')).currency := true;
  TFloatField(TempQuery.FN('Sell Price')).currency := true;

  TempDS := TDatasource.create(frmMain);
  TempDS.Dataset := TempQuery;
  {Top Panel}
  pnlTop := TPanel.Create(frmMain);
  pnlTop.Parent := frmMain;
  pnlTop.Align := alclient;
  pnlTop.BevelOuter :=  bvLowered;

  {Create the Data aware grid and set the column options}
  dbgData := TDBGrid.create(frmMain);
  dbgData.parent := pnlTop;
  dbgData.align := alClient;
  dbgData.datasource := TempDS;
  dbgData.columns[0].width := 140;
  dbgData.columns[0].readonly := true;
  dbgData.columns[1].width := 200;
  dbgData.columns[1].readonly := true;
  dbgData.columns[2].width := 40;
  dbgData.columns[2].readonly := true;
  dbgData.columns[3].readonly := true;
  dbgData.columns[4].readonly := true;
  dbgData.columns[5].readonly := true;
  dbgData.columns[8].visible := false; {Hide the Sysuniqueid field}
  {Create the Bottom Panel with Buttons}

  {Bottom Panel}
  pnlBottom := TPanel.Create(frmMain);
  pnlBottom.Parent := frmMain;
  pnlBottom.Height := 50;
  pnlBottom.width := 700;
  pnlBottom.Align := albottom;
  pnlBottom.BevelOuter :=  bvNone;

  {Close Button}
  btnClose := TButton.Create(pnlBottom);
  btnClose.Parent := pnlBottom;
  btnClose.left := 600;
  btnClose.Top := 15;
  btnClose.Width := 90;
  btnClose.Height := 20;
  btnClose.Caption := '&Close';
  btnClose.OnClick := 'CloseClick';
  btnClose.Hint := 'Close Price Update Screen';
  btnClose.ShowHint := True;
  btnClose.taborder := 99;
  btnClose.anchors := akright;

  frmMain.showmodal;
  end;

 finally
  frmMain.free;

 end;

end.
```

9.3 Call Centre - Email Import

```
{XXXXX CALL CENTRE XXXXX}

{This script shows importing from and email send via a web form}
{Shows using the InsertRecord function with field mapping using a StringList}
{Wide range of uses, the example show entering call centre Tickets from a web page}
{You can use page below to test example}
{http://www.development-x.com/callticket.html}

{Script Constants, change to suit your email settings - or demo using the ones below}
const
  POP3Server = 'mail.development-x.com';
  POP3UserName = 'demo@development-x.com';
  POP3Password = 'demo';

var
  CallCount: Integer;

function GetNextMappingIndex(CurrIndex: Integer; EmailBody: String): Integer;
var
  TempList: TStringList;
  x: Integer;
begin
  {Function to find the end of a multi line value - (memo notes)}
  Result := CurrIndex;
  TempList := TStringList.create;
  try
    TempList.Text := EmailBody;
    for x := (CurrIndex+1) to TempList.Count - 1 do
    begin
      if trim(TempList.Names[x]) <> '' then
      begin
        Result := x;
        exit;
        break;
      end;
      inc(Result);
    end;
  finally
    TempList.free;
  end;
end;

function RetreiveValueFromBody(EmailBody: String; FieldName: String): String;
var
  TempList: TStringList;
  x, i, endindex: Integer;
begin
  {Function to get a value from the email body in format Name=Value}
  Result := '';
  TempList := TStringList.create;
  try
    TempList.Text := EmailBody;
    x := TempList.IndexOfName(FieldName);
    if x = -1 then exit;
    {Multi Line Values}
    endindex := GetNextMappingIndex(x,EmailBody);
    Result := TempList.Values[TempList.Names[x]];
    if endindex > (x+1) then
    for i := (x+1) to endindex do
    if pos('MID #',TempList.Strings[i]) = 0 then {remove any unwanted text from the email body}
      Result := Result + #13 + TempList.Strings[i];
  finally
    TempList.free;
  end;
end;

procedure GetEmailRequest;
var
  MsgCount: Integer;
  x: Integer;
  EmailSubject, EmailBody: String;
  Mappings: TStringList;
begin
  Mappings := TStringList.Create;
  try
    {receive email using settings specified at top of script in "const" section}
    MsgCount := ReceiveEmail(POP3Server,POP3UserName,POP3Password,True);
    for x := 1 to MsgCount do
    begin
      EmailSubject :=GetEmailMessage('Subject',x);
      EmailBody := GetEmailMessage('Body',x);
      {look for a specific subject}
      if ( (trim(EmailBody) <> '') and (uppercase(EmailSubject) = 'CALLTICKET') ) then
      begin
        Mappings.Clear;
        {Fixed Values}
        Mappings.Add('CALLMETHOD=Web');
        Mappings.Add('COMPANYTYPE=Prospect');
        Mappings.Add('CALLDATE=' + DateToStr(Date));
        Mappings.Add('CALLTIME=' + timetostr(Now));
        {Values read from email}
        Mappings.Add('COMPANYNAME=' +
RetreiveValueFromBody(EmailBody,'COMPANYNAME'));
        Mappings.Add('CALLNOTE=' + RetreiveValueFromBody(EmailBody,'CALLNOTE'));
        {Insert record into CALLNOTES table}
        InsertRecord('CALLNOTES',Mappings.text,True);
        inc(CallCount);
      end;
    end;
  finally
    Mappings.free;
  end;
end;

begin {Main Script Section}
  CallCount := 0;
  {Call function to receive email messages sent from website, mobile phone, PDA, text to email
etc}
  GetEmailRequest;
  MessageDlg(inttostr(CallCount) + ' call requests were processed',mtInformation,mbok,0);
end.
```

9.4 Import Timesheets from Excel

```
{XXXXX Import Timesheets from excel file XXXXX}

{This script shows opening an excel file and creating a TimeSheet batch}
{from the cells contents. Note: the excel columns and rows are 0 based}
{so to read the first cell would be SSGetCellText(0,0) (Column,Row)}
{The download email attachment example could be included with this one}
{to first download the excel file from an email}

function GetExcelFile: string;
var
  OpenDlg: TOpenDialog;
begin
  OpenDlg := TOpenDialog.Create(Nil);
  try
    OpenDlg.Filter := 'Excel Files (*.xls)|*.xls|All Files (*.*)|*.*';
    if OpenDlg.Execute then
      Result := OpenDlg.filename
    else
      Result := '';
  finally
   OpenDlg.Free;
  end;
end;

function InsertHeader(ExcelFileName,EmployeeName,WeekEnding: String): Integer;
begin
  Result := InsertTimesheetHeader('InProgress','For Week Ending: ' + WeekEnding,'Employee'
   ,EmployeeName,True,Date,'Waiting Approval',date,'','Imported from Excel (' + ExcelFileName +
')',Date);
end;

procedure CreateTimeSheetEntries;
var
  ExcelFileName,EmployeeName: String;
  x, RowCount, BatchNumber: Integer;
begin
  ExcelFileName := GetExcelFile;
  if trim(ExcelFileName) = '' then exit;
  LoadSpreadSheet(ExcelFileName);
  RowCount := SSGetContentRowCount;
  BatchNumber := InsertHeader(ExcelFileName,SSGetCellText(1,0),SSGetCellText(1,1));
  EmployeeName := SSGetCellText(1,0);
  for x := 3 to RowCount -1 do
  {Check we have a date}
  if trim(SSGetCellText(0,x)) <> '' then
  begin
    InsertTimesheetLine(
    BatchNumber
    , strtodatetime(SSGetCellText(0,x))
    ,SSGetCellText(1,x)
    ,SSGetCellText(2,x)
    ,EmployeeName
    ,'STD'
    ,SSGetCellText(4,x)
    ,'LAB-SERVICE'
    ,0
    ,0
    ,strtofloat(SSGetCellText(3,x))
    ,'Chargeable'
    ,''
    ,SSGetCellText(5,x)
    ,False
    ,False
    ,False
    ,'');
  end;
  {Optionally update the spreadsheet so you know it has been imported}
  SSSetCellText(2,0,'Batch No:');
  SSSetCellText(2,1,'Imported On:');
  SSSetCellText(3,0,inttostr(BatchNumber));
  SSSetCellText(3,1,datetimetostr(Date));
  SaveSpreadsheet(ExcelFileName);
end;

begin
  CreateTimeSheetEntries;
  MessageDlg('Timesheet Imported',mtinformation,mbok,0);
end.
```

9.5 Microsoft Access Database Example

```
{XXXXX Simple microsoft access database example XXXXX}

{This script shows opening a MS Access file and performing}
{delete, insert and select query's on the database table "Customers"}
{Could be used to either write data from Ostendo to Access for a}
{custom application or import data from a PDA or other device using Access}

var
  ADO: TADOConnection;
  ADOQry: TADOQuery;
begin
  ADO := TADOConnection.Create(nil);
  ADOQry := TADOQuery.Create(nil);
  try
    ADO.ConnectionString := 'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' + OstendoPath +
'accessdemo.mdb';
    ADO.LoginPrompt := False;
    ADO.Connected := true;
    ADOQry.Connection := ADO;
    {delete sql}
    ADOQry.SQL.Add('delete from Customers');
    ADOQry.ExecSQL;
    ADOQry.Close;
    ADOQry.SQL.Clear;
    {Insert sql}
    ADOQry.SQL.Add('insert into Customers (CompanyName,EmailAddress) values (''Test
Company'', ''test@tc.test'')');
    ADOQry.ExecSQL;
    ADOQry.Close;
    ADOQry.SQL.Clear;
    {Select sql}
    ADOQry.SQL.Add('Select * from Customers');
    ADOQry.Open;
    showmessage(ADOQry.FieldByName('CompanyName').AsString
     + ' - ' + ADOQry.FieldByName('EmailAddress').AsString);
    ADOQry.close;
    ADO.Connected := false;
  finally
    ADOQry.Free;
    ADO.Free;
  end;
end.

{Read table from ostendo loop and save to text file}
var
  OSTQry: TpFIBQuery;
  CustList: TStringList;
begin
  OSTQry := TpFIBQuery.Create(nil);
  CustList := TStringList.Create;
  try
    OSTQry.Database := OstendoDB;
    OSTQry.SQL.Add('select CUSTOMER from CUSTOMERMASTER');
    OSTQry.ExecQuery;
    while not OSTQry.Eof do
    begin
      CustList.Add(OSTQry.FN('CUSTOMER').AsString);
      OSTQry.Next;
    end;
    CustList.SaveToFile(OstendoPath + 'customers.txt');
    run(OstendoPath + 'customers.txt');
  finally
    CustList.Free;
    OSTQry.Free;
  end;
end.
```

9.6 MYOB Customer File CSV Import

```
{XXXXX MYOB CSV CUSTOMERS IMPORT XXXXX}


{This script shows importing from an MYOB comma separated values file}
{Shows parsing values from the line using its index}
{How to use a progress indicator is also shown}

function GetCSVFile: string;
var
  OpenDlg: TOpenDialog;
begin
  OpenDlg := TOpenDialog.Create(Nil);
  try
    OpenDlg.Filter := 'Text Files (*.txt)|*.txt|Comma Separated Values (*.csv)|*.csv|All Files (*.*)|*.*';
    if OpenDlg.Execute then
      Result := OpenDlg.filename
    else
    Result := '';
  finally
   OpenDlg.Free;
  end;
end;

function ImportFromCSVFile: boolean;
var
  ImportFile: String;
  ImportValuesList, FieldMappings: TStringList;
  x: Integer;
  CustomerType, TaxGroup: String;
begin
  Result := True;
  ImportFile := GetCSVFile;
  if trim(ImportFile) = '' then
  begin
    Result := False;
    exit;
  end;
  {Ask for a customer type, Tax Group to be used}
  CustomerType := AskQuestionWithLookup('Select a Customer Type', 'This will be used for all
Customers you are importing', 1028, '');
  TaxGroup := AskQuestionWithLookup('Select a Tax Group', 'This will be used for all Customers
you are importing', 1017, '');
  if trim(CustomerType) = '' then
  begin
    Result := False;
    exit;
  end;
  ImportValuesList := TStringList.Create;
  FieldMappings := TStringList.Create;
  try
    {Load ImportFile into StringList}
    ImportValuesList.LoadFromFile(ImportFile);
    FieldMappings.clear;
    {Loop the list and add fieldnames/values to mappings StringList}
    {Start at 1 rather than 0 to skip header record if you exported it from MYOB}
    ShowProgress('Importing Customers...', (ImportValuesList.Count-1), False);
    for x := 1 to ImportValuesList.Count -1 do {check customer does not already exist}
    begin
      if DBValueExists('CUSTOMERMASTER', 'CUSTOMER',
ParseString(ImportValuesList.Strings[x],',',0), False) <> True then
      begin
        FieldMappings.clear;
        FieldMappings.Add('CUSTOMER=' + ParseString(ImportValuesList.Strings[x],',',0));
        FieldMappings.Add('CUSTOMERTYPE=' + CustomerType);
        FieldMappings.Add('CUSTOMERSTATUS=Active');
        FieldMappings.Add('CUSTOMERINVOICENAME=' +
ParseString(ImportValuesList.Strings[x],',',0));
        FieldMappings.Add('CUSTOMERADDRESS1=' +
ParseString(ImportValuesList.Strings[x],',',5));
        FieldMappings.Add('CUSTOMERADDRESS2=' +
ParseString(ImportValuesList.Strings[x],',',6));
        FieldMappings.Add('CUSTOMERADDRESS3=' +
ParseString(ImportValuesList.Strings[x],',',7));
        FieldMappings.Add('CUSTOMERPOSTALCODE=' +
ParseString(ImportValuesList.Strings[x],',',11));
        FieldMappings.Add('CUSTOMERSTATE=' + ParseString(ImportValuesList.Strings[x],',',10));
        FieldMappings.Add('CUSTOMERCITY=' + ParseString(ImportValuesList.Strings[x],',',9));
        FieldMappings.Add('CUSTOMERCOUNTRY=' +
ParseString(ImportValuesList.Strings[x],',',12));
        FieldMappings.Add('CUSTOMERPHONE=' + ParseString(ImportValuesList.Strings[x],',',13));
        FieldMappings.Add('CUSTOMERFAX=' + ParseString(ImportValuesList.Strings[x],',',16));
        FieldMappings.Add('CUSTOMEREMAIL=' + ParseString(ImportValuesList.Strings[x],',',17));
        FieldMappings.Add('CUSTOMERWEB=' + ParseString(ImportValuesList.Strings[x],',',18));
        FieldMappings.Add('TAXGROUP=' + TaxGroup);
        InsertRecord('CUSTOMERMASTER', FieldMappings.text, False);
      end;
      UpdateProgress(x);
    end;
  finally
    EndProgress;
    ImportValuesList.Free;
    FieldMappings.Free;
  end;
end;

begin
  if ImportFromCSVFile then
    MessageDlg('Import Complete',mtinformation,mbok,0)
  else
    MessageDlg('Import Cancelled',mtinformation,mbok,0);
end.
```