Understanding and Using Plato

What is Plato?

Plato is an order handling system for a fictitious chrome plating shop. It serves as an example of a DataFlex application running against a SQL Server backend.

The employees of the shop operate Plato. They use it to enter orders and promote them through the different stages of the handling process. During the process, emails are sent to the customer with *proposals*, *order confirmations* and *invoices*.

Plato is not designed to be a fully functional, real life system. It is an example application that illustrates various designs and techniques, with special emphasis on the use of an SQL Server backend.

Contents

Plato User Access

The users of Plato are the employees of the chrome plating shop and they identify themselves by their email address. Plato relies on this since it will attempt to send emails to the user on various occasions. A user is assigned one of three access levels of which the highest enables administration of other users.

Users access level is stored in `WebAppUser.Rights` and may take on these values

- 1 (standard): Normal user. Can operate all data entry and reporting
- 2 (invoicing): Normal user with added access to invoice specific functions
- 10 (admin): The user can do everything including system setup

Initial Login

During the installation of the Plato sample application a couple of users are created. The main user to explore the application' capabilities is named 'John'. John's password the same as his user name (this applies to all the users in the system). Of course, these passwords aren't secure by themselves but we can do this because you can't read the password in the database. Passwords are encrypted using the new security module, and we didn't want to document the encrypted passwords. Of course, you can change the passwords and/or create new user accounts using any passwords you wish.

Self Registration

A user may click "Register" in the Plato login panel to specify the username and password of a new user. This newly created user will be able to login but will only be given access to a "welcome" panel.

An admin user assigns the access level to all new users before any work can be done. To notify the admin user a (clickable) red sign will appear in the dashboard page saying that new users are waiting to be validated.

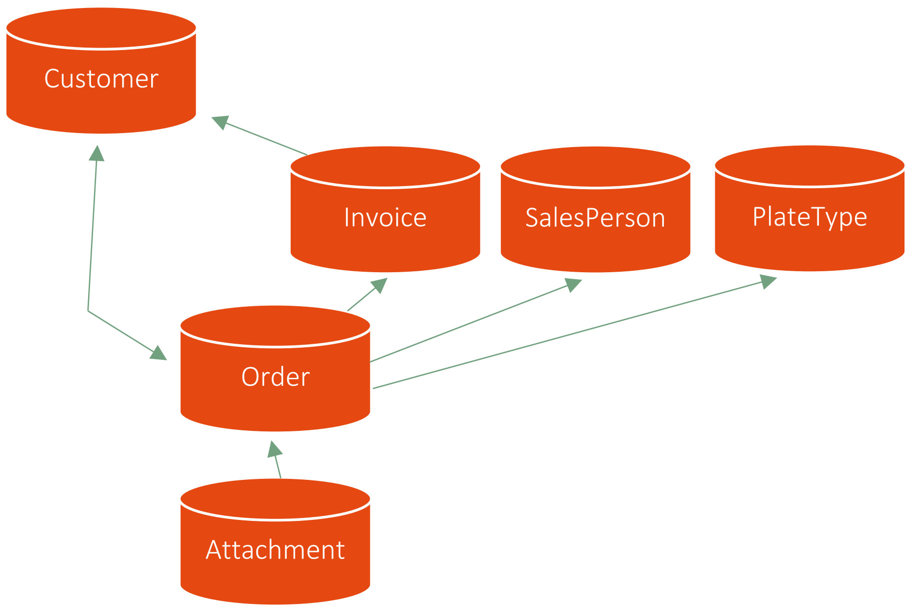
Restoring the Demo Data

Run PlatoReset.exe from the Programs folder to restore the original data including users John, Lisa and Karl. Each with their name as password. PlatoReset.exe uses Microsoft SQL Server's command line tool SQLCMD. (https://docs.microsoft.com/en-us/sql/ssms/scripting/sqlcmd-run-transact-sql-script-files?view=sql-server-ver15) and loads the data from PlatoData.sql in the SQL folder of the workspace.

The Plato Database

Description

The central table is the **order** table. An order record contains all information about the order (including quantity, unit price, and freight rate. There are no "order detail lines").



An order relates to a **customer**. A customer has an email address (to which we send emails) and the usual address information. Plus some setup for invoicing.

An order relates to an **invoice** as well (null-relations are allowed). In turn the invoice points to a customer.

Invoice records are created by a batch routine that takes all orders for a particular customer that are ripe for invoicing and adds them to the same new invoice. A pdf file is generated and stored on disk (as well as mailed to the customer).

The **Attachment** table relates to Order and supports storing files relevant to the order. The files themselves are stored on disk, not in the attachment table. For this reason, there are two subfolders of the `data` folder; `InvoicesFolder` and `OrderFolders`. The contents of these are part of the Plato data.

An order has two attributes `PlateTypeID` and `SubjectSurface` that are both pointers to each their own table (**`PlateType`** and **`Substrate`**). For the sake of variation plate type has been implemented as a table relation while substrate has been implemented as a FileValidationTable (see cOrderDataDictionary.dd). This is the reason why **`PlateType`** appears in the database relations diagram above,but not **`Substrate`**.

In addition, Plato uses 3 lookup tables:

- `Substrate`
- `GeoCountry`

- `GeoSubdivision`

There are also tables maintained by logic outside of Plato (standard classes and classes from the SECMOD library):

```
WebAppSession, WebAppUser, WebAppServerProps,
WebAppUserPasswordHistory, WebAppUserReset, WebAppServerSettings and
WebAppAuthToken
```

The Order Table

An order is identified by an order number. An order has several payload columns such as `JobDescription`, `InvoiceAmount`, `EstimatedHours` etc.

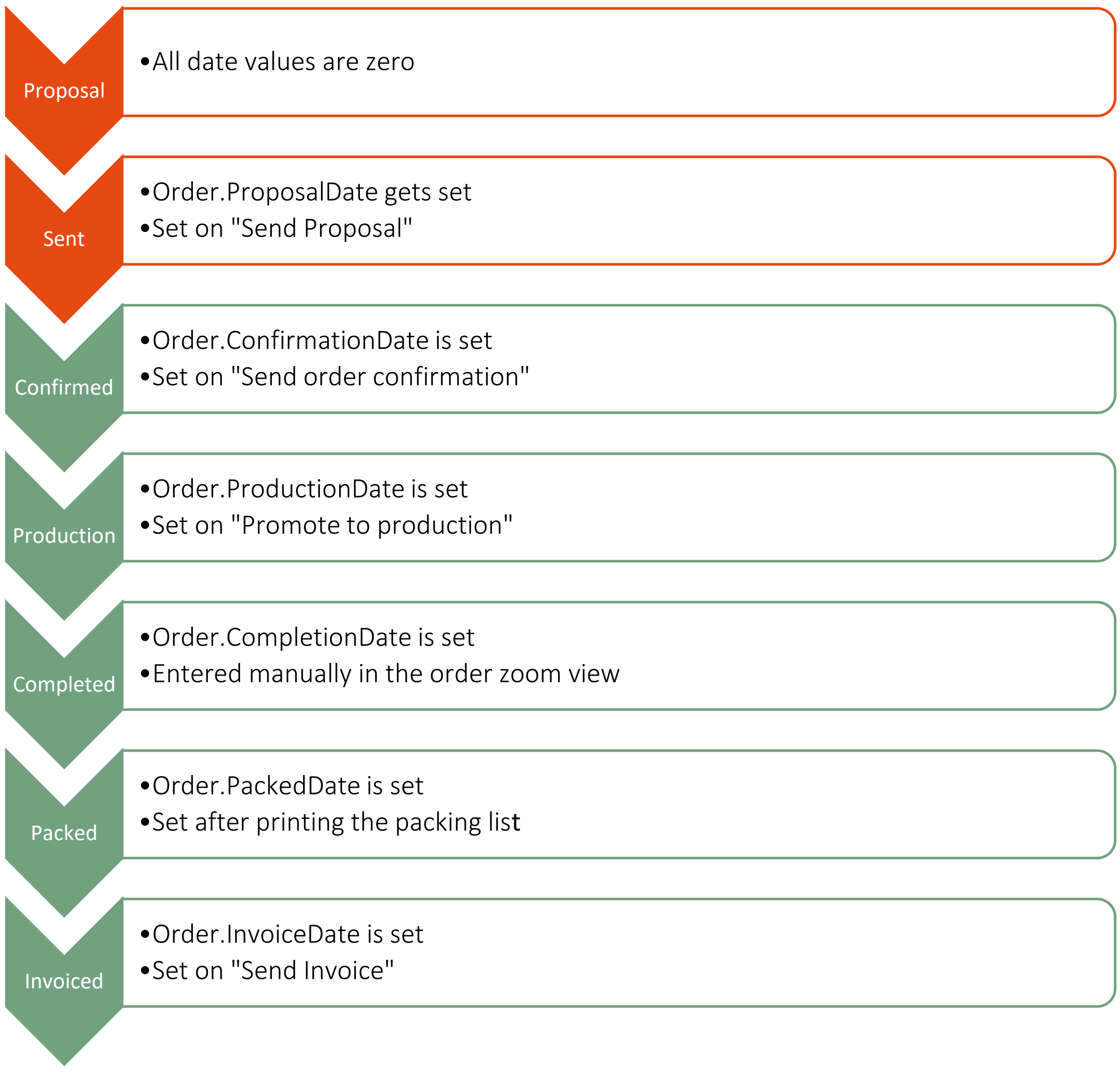
The state of an order (its place in the handling sequence) is determined in part by the value of `Order.OrderType`. 1 = `proposal`, 2 = `production`.

An order has several dates (listed below) that together determine the status of an order

- `ProposalDate`, when proposal was mailed to customer
- `ConfirmationDate`, when the proposal was accepted by the customer
- `ProductionDate`, when set in production
- `CompletionDate`, when work was completed
- `PackedDate`, when packed
- `InvoiceDate`, when invoiced

Order Workflow

The workflow of an order is supposed to progress like this:



Proposal
- All date values are zero

Sent
- Order.ProposalDate gets set
- Set on "Send Proposal"

Confirmed
- Order.ConfirmationDate is set
- Set on "Send order confirmation"

Production
- Order.ProductionDate is set
- Set on "Promote to production"

Completed
- Order.CompletionDate is set
- Entered manually in the order zoom view

Packed
- Order.PackedDate is set
- Set after printing the packing list

Invoiced
- Order.InvoiceDate is set
- Set on "Send Invoice"

The red chevron indicates `ordertype = 1` and the green ones `ordertype = 2`.
An order may start its life as a proposal or directly as a production order. Hence the two tiles in the dashboard “New proposal” and “New work order”.

Examples on logic based on this:

Orders that are ready to be invoiced will have:

```
OrderType equals to 2, PackedDate not empty and no InvoiceDate
```

Unfinished production orders will have:

```
OrderType equals to 2 and no CompletionDate
```

An order has two more dates, but these do not influence the flow of the order:

- `CreateDate`, when the order record was created
- `DeliveryDate`, when the order should be delivered

The Salesperson “Role”

When an order is created, the user assigns a salesperson selected from the SalesPerson table, UNLESS the user is also a SalesPerson. In that case, the user is automatically assigned as the salesperson for that order.

Login as Karl to see how the automatic salesperson assignment works. Karl is the default salesperson in Plato.

A salesperson may be associated with a user in the “Salesperson” view from the “Settings” menu.

This allows us to add special behaviors to specific users without creating extra columns in the WebAppUser table. At the same time, the SalesPerson record can serve as a placeholder for any additional context we may have when defining this “role”.

Features

Embedded SQL

The user interface classes of a DataFlex web application access the table data through the data dictionaries, which automatically generate SQL statements and execute them on the server in an optimized manner. This is handled for us and no action is required.

But on occasion we want to compose an SQL statement in a string, execute it on the server and get the result back to the program. To automate the handling of connections and statements we have defined the cSQLExec class.

Plato creates a singleton object of this cSQLExec class object and program code can access the object via the global handle variable named ghoSql.

The object uses the connection information assigned to the WebAppSession table. When you want to use the class in your own application, and you have multiple connections (to one or more SQL instance or database) you need to check whether this singleton object can be used.

```
Function ExecuteSql String sSql Returns String[][]
```

The ExecuteSql function executes the statement in sSql and returns the result set.

```
String[][] aResult
Get ExecuteSql of ghoSql "SELECT * FROM [Order]" ;
    to aResult
If (SizeOfArray(aResult)>0) Begin
    ….
End
```

Settings

Plato contains two sets of settings. One is about security and e-mailing and one set for the Plato specific settings. Only admin users can view and maintain the settings.

Plato Specific Settings

The settings are categorized in Defaults, Address, Reports Info, Export options, Table usage and Miscellaneous.

Defaults

The default country, plate type and substrate for a new proposal or order can be selected here.

Address

The address values here are used in the reports and passed as parameter values via the report integration code.

Reports Info

Shows the information about the used DataFlex Reports OCX, connection string, version etc.

Export Options

Exporting of an invoice to CSV was made for the Exact Online bookkeeping system. The IDs for ledger numbers and row indicators can be modified and maintained here. The application requires changes when a different output is desired.

Table Usage

These values are not really settings, the boxes show the number of rows in each of the tables of the application. The values are retrieved via querying one of the DataFlex database API attributes for a table.

Secmod Settings

The first three tabs for the security settings are about passwords; how many characters, repetition of patterns etc.

The last tab is about configuring an SMTP server to send e-mails to. Note that when this is configured you should disable the option to capture e-mails and from the demo data remove the double '@' signs in the e-mail addresses. Please not the demo data may have e-mail addresses that might exist, we did not proof a non-existence.

Email

Blocked Emails

As mentioned, mail is sent to customers during handling of proposals, order confirmations and invoices. And mail is sent to users when passwords are forgotten and the like.

But Plato is a demo system and therefore steps have been taken to enable blocking of email, so they are not actually sent but instead simulated in a break out browser tab.

A user with administrative rights (that's John/John) can toggle this behaviour under "Plato settings" from the settings menu. Checking "Capture emails" will block any mails from being sent from Plato. The default state is that emails will be captured.

Live Email

When email is not captured, it is sent via SMTP. The parameters used are specified under "Security settings" in the "Mail" tab.

Note that the customer data supplied with the sample all have illegal email addresses (with two @ signs in them). Sending to an illegal address will trigger the capturing mechanism as if capturing had been switched on.

Quick Start

Handling an Order

Start from the dashboard

- Click "New proposal"
- Select customer (or create and select customer)
- Fill in the “Job description”
- Enter a unit price 37
- Click "Print draft"
- Close draft
- Click "Send proposal" (this has the side effect of setting order.proposaldate)
- Click save icon to save the proposal (you will return to the dashboard with your proposal at the top of "pending proposals")
- Click the proposal (you're back in the order-zoom)
- Click "Promote to production"

At this point the "Send order confirmation" page appears. The reason is that we sent a proposal further up. It wants to know whether we should also send a confirmation.

Had we chosen not to "Send proposal" we would have skipped this dialog and returned to the dashboard with the order at the top of the top-10 list.

- Click "Send confirmation" (and we're back in the dashboard)
- Click "Print packing slip"
- Click the printer icon top right

A print dialog will appear. The side effect of showing that dialog is that the Orders.PackingDate was set.

- Cancel the print dialog
- Go back one step on the crumb path to the order-zoom
- Fill in the "Order completed" date and save the order.

You are back in the dashboard and the order is nowhere to be found.

- From the hamburger menu select "Invoicing" -> "Create invoice”

The customer select appears showing the customers with non invoiced orders. The number of not invoiced orders is shown.

When oCustomerSelect acts in this role (selecting customers for invoicing) it will only list customers with orders ready to be invoiced.

- Select the customer
- Click "Pro forma invoice"
- Click "Send Invoice" from the hamburger menu select "Invoicing" -> "Export invoice”

And we’re full circle.

The cDRReportPlato Class

In order to facilitate the consistent look and feel of the reports in the application, and to allow for more portability of the application, we have developed a nested subclass of the cDRReport class. cDRReportPlato is in turn a subclass of cDRReportMSSQL. Each subclass performs a separate set of functions. cDRReportMSSQL will set the correct database connection at runtime, and cDRReportPlato takes care of setting default parameters across all reports in a consistent, and easy manner.

By setting the report parameters in the subclass, the code only needs to be implemented once, and if in the future the company makes a change, all the reports can be updated quickly. A similar approach could be used in a multi-tenant environment, where the reports may need to be sent out with different header information for each tenant company in the system.

We also set a 'locale' parameter to show how the date formatting can be adjusted at runtime, by using the Locale parameter.

In each case we must first test that the parameter is defined in the report before trying to set it. Setting a parameter from integration where it has not been defined in the report file itself, will cause an error at runtime.

The "default" parameters that we attempt to set for every report are:

- psLogoPath
- CompanyAddress
- CompanyCountry
- CompanyFax
- CompanyPhone
- CompanyZipCity
- Locale

The cDRReportMSSQL Class

The purpose of cDRReportMSSQL is to augment Function OpenReport and the automatically connect to the currently used database regardless of which connection string is stored in the .dr file.

This is done by generating a "DSN-less" ODBC connection string. We have made an executive decision that this example subclass will only be intended for use with reports that connect to the MSSQL database. The class expects the developer to establish a 'reference' table, which will be used to fetch the database connection information, which in turn is used to create the DSN-less connection.

There is only one Public property specific to this class: Handle phFollowTableConnection. This is the file number of a connected table in the database, from which connection details are "borrowed".

If you wish to use this technique with other databases, feel free to model your new subclass on this one.

How DR Reports Interact with Plato

1. CustomersList.dr

This report produces a plain list of customers and contains a built-in predefined filter function on Customer.City but is intentionally commented out. That is the most basic way to filter a report, and is often used in report development, but for most cases is too limited for runtime use. The application integration can and does filter the report based on user selections in CutomerListReportSelect.wo using the new cWebtagsForm class. The sample shows how several tag groups can be joined to produce a very complex filter. Generating the filters is a two-step process, where first we filter the data to show all the available options, and then when the report is run, the selections are passed to the report display view, CustomerReportResults.wo as named Values in the NavigateData Structure. The SetFilters method in the Results Web Object collects all the named Value pairs, and creates the appropriate filter function, using string manipulation.

For debugging purposes, it is a good idea to put a breakpoint on the "Set psFilterFunction" line at the end of the SetFilters method in the report results page.

Used tables: Customer

2. Invoice.dr

Prints an invoice containing all orders that are packed but not yet invoiced for the customer passed in the report filter. An explicit Filter Value is set in integration, which overwrites the default filter of CustomerNumber = 10

This report uses the predefined parameters from the cDRReportPlato class, and an explicit parameter 'Draft' to toggle the "proforma" text. The "suppress" function for text boxes are set using

```
Return {?Draft}=0
```

Used tables: Order and Customer

3. Order.dr

Uses direct / Embedded SQL to look up/decode the "Subject Surface" but uses a linked table to display Plate Type name. Shows how a function can be used to convert a simple ASCII value into a Unicode symbol for a more modern look, the checkboxes for sanding, buffing and repair are populated by code that looks like this in a function

```
if {Order.PrepareBuffing}="1" then
      return "✓"
else
      return ""
end
```

This report also demonstrates the use of a sub-report for related Images, and barcodes for the Work Order number.

Used tables: Order, Customer, PlateType and SalesPerson

4. OrderConfirmation.dr

Printed order confirmation is sent to customers email address
In this report we demonstrate how a report can be created on the server into a PDF file, and then emailed as an attachment. The attached files are stored on the server in a folder structure organized by year, with the date sent being added to the order number as the file name. Do note that this can create a 'dead file' storage issue if used in production, so a proper data retention policy should be applied to these temporary files & folders.

Used tables: Order and Customer

5. OrdersList.dr

The report prints a list of order where all orders are either not completed (WorkOrderYear=0) or all orders completed in a particular year (WorkOrderYear<>0). Dynamic section suppression allows the same report to appear differently, based on whether open or closed work orders are being listed. We also demonstrate dynamically changing the sort order of the report through integration. Embedded SQL is used to populate the 'year' drop down combo box, guiding the user to select only years for which there do exist completed orders. This report also has a "hyperlink ID" defined on the order number column, and the Customer name column. At runtime, the fields are displayed as hyperlinks, and clicking on the linked data will take the user to the appropriate Order, or Customer page.

Used tables: Order and Customer

6. PackingList.dr

Print packing slip for order. Shows use of boxes and lines. Barcode. This report shows the use of the `ReportTableColumnName` in integration. This method translates the passed table and column name reference into the table and column name as defined in the report. The alternative is hard coded table and column names in a string but it requires a change when table or column names are changed in the project.

Used tables: Order and Customer

7. Proposal.dr

Prints a quotation and optionally mails it to the customer, using the same techniques as in the Order Confirmation.

Shows how a field can have graduated shading, uses barcodes, the logo is dynamically loaded based on the parameter passed by the subclass.

Used tables: Order, Customer and GeoSubDivision