aptean

Gone for Good Limited

Gift Aid Capture and Interface

CALIDUS ePOD

1st October 2018 - 1.0
Reference: FS 353019

Contents

1 Functional Description

1.1 C-TMS

1.1.1 Sending Jobs to C-ePOD

A new System Parameter EPOD_OWNER_CONFIG will be created at Cost Centre level.

This will have description "C - Send Customer Name and ID or Lots Installed in Owner Name and ID. L or other value - Send Lots Installed in Owner Name only"

Database procedure DP_EPOD_WEBSERVICE.epod_out will be modified to check this parameter when in MODE2 (general EPOD mode) for the cost centre.

If set to value "C", the process will add 2 XML nodes, with data set from ORG_CUSTOMER:

- EPL_OWNER_ID, set to value LOTS_INSTALLED if present or CUSTOMER_ID.
- EPL_OWNER_NAME, set to value CUSTOMER_NAME

If set to "L" or any other value, the process will add 1 XML node, with data set from ORG_CUSTOMER, as it does now:

- EPL_OWNER_NAME, set to value LOTS_INSTALLED

Note: No branches of this package need to be modified.

1.1.2 Database Changes

The SCH_ORD_REFERENCE table will be modified with the following additional field:

- SUB_REF_VALUE2 - a CLOB (character large object)

1.1.3 Receiving Updates from C-ePOD

A new System Parameter EPOD_SIGNATURE_STORAGE will be created at Cost Centre level.

This will have description "Y - Store Signature, Signatory and T&Cs against the order when complete. N or other value - Do not store."

The Job Update message back from C-ePOD already includes the following:

- EPL_JOB_SIGNATURE - a base64-encoded image of the signature.
- EPL_JOB_SIGNATORY - the name of the customer signing for the goods.
- EPL_TNCS - The terms and Conditions, plus all other related information (in this case, the Gift Aid Registration form).

XML_REFERENCE import decodes will be added for the following:

- Source value "SIGNATURE", target value "Signature"
- Source value "TNCS", target value "Terms & Conditions"

Note also that source value "SIGNATORY" must be present.

Database procedure DP_EPOD_WEBSERVICE.epod_job_in will be modified to store values in these references on SCH_ORD_REFERENCE only if the new system parameter EPOD_SIGNATURE_STORAGE is set to "Y" and the mode is "MODE2".

In this case, the procedure will check the value is populated on the 3 fields below and store them if populated on SCH_ORD_REFERENCE:

- EPL_JOB_SIGNATURE - store in new field SUB_REF_VALUE2, for SUB_REF_NAME "SIGNATURE". SUB_REF_VALUE should be set to "Stored".
- EPL_JOB_SIGNATORY - store in new field SUB_REF_VALUE, for SUB_REF_NAME "SIGNATORY". SUB_REF_VALUE2 should be omitted.
- EPL_TNCS - store in new field SUB_REF_VALUE2, for SUB_REF_NAME "TNCS". SUB_REF_VALUE should be set to "Stored".

1.1.4 Exporting Trip/Order XML

The standard Trip/Order XML export will be modified to handle the large data being sent (potentially greater than 32Kb per line). The process will create Insert files for each CLOB required, with placeholders in the master file. After creation of the master file, the process will merge the contents of each insert file with the master file.

The existing procedure GEN_ORDER_SUB will be amended to process the CLOB data differently when new field SUB_REF_VALUE2 has been populated.

A count will be initiated to record each time the new field is populated. Currently information is written to the outbound file using the put_line command.

To store the CLOB data, a new process will be called to create a new file, referencing the count. The PUT_RAW command will be used to write the CLOB data to the new file.

```
UTL_FILE.OPEN_FILE ({FILE}||COUNT);
UTL_FILE.PUT_RAW (cp_file_handle2, sch_ord_referenves.sub_ref_value2);
UTL_FILE.FFLUSH (cp_file_handle2);
UTL_FILE.CLOSE_FILE (({FILE}||COUNT);
```

Within the GEN_ORDER_SUB process, the <SUB_REFERENCE> tag will point to the new file {FILE}||COUNT.

Unix commands will replace the file reference with the contents of the relevant file when building the XML to send out.

Once the Unix process has completed and the files are merged, the Insert files (containing the CLOB data) will be removed from the file system.

1.2 C-ePOD

1.2.1 Database/DAL

A new field will be added to EPOD_JOB:

- EPL_OWNER_ID, added as nvarchar(12)

This field will be added to all stored procedures in the database that require them.

This field is required on the device.

This field is required to be set on jobs imported through the standard XML import (flat file or webservice). The Import XSD will be modified to show these fields. This field is not required to be provided.

This field is required to be contained within the Export.

This field is not required to be used when filtering data for selection.

Standard XML Interface documentation will be modified as part of this change.

1.2.2 Server

The export to TTM from EPOD will be modified to use Owner ID before Owner Name.

Method EPOD_JOB.ToXElementTTM will be modified to achieve this.

Currently, the Owner Name is only used if configured to do so (with EPOD_SITE.EPL_USE_OWNER_NAME).

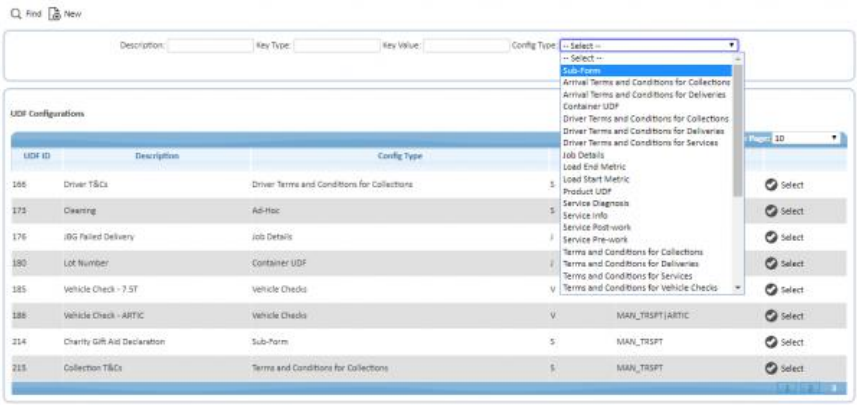
In this section, the value in EPL_OWNER_ID will be used if present and populated with a non-blank/null value, instead on EPL_OWNER_NAME.

If EPL_OWNER_ID is not present, the code will continue as it does now.

1.2.3 Admin

The existing UDF Configuration screen UDF_Config.aspx will be modified to:

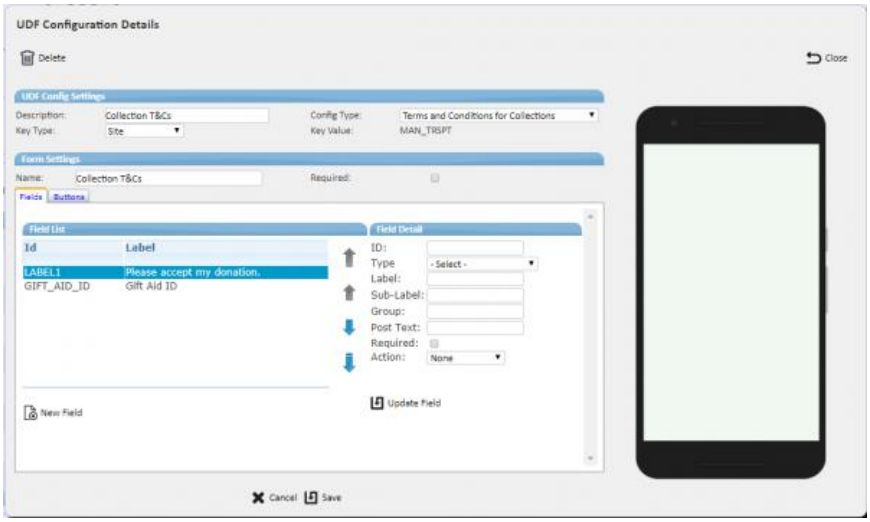
- Allow creation of sub-forms
- Allow default values
- Allow Button-type fields to be defined.

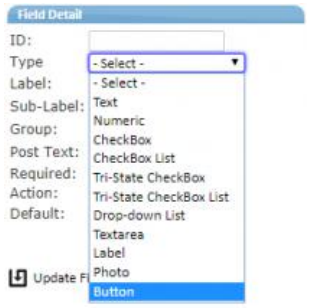


The UDF type "SUBFORM" description "Sub-Form" will be added to the list items for:

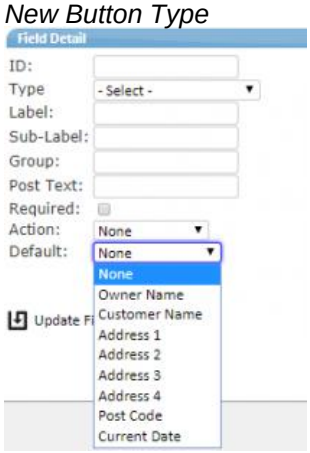
- Finding created UDF.
- Creating a new UDF.
- Modifying an existing UDF.

This is populated on database table EPOD_LIST_ITEMS for the List ID of EPOD_LISTS ListName "ConfigTypeDDLData".



- 

New Button Type

- 

New Default parameter

When creating or modifying UDF for types that are not "SUBFORM", a field type of "BT" (description "Button") may be selected.

When selecting Button types, the following parameters are valid - all others will be disabled:

- ID
- LABEL
- ACTION
- REQUIRED

Parameter "Action" values for Button types will be a drop-down list of Sub-Form UDFs in the system for that site (EPOD_UDF_CONFIG where EPL_KEY_VAL begins with the user's logged-in Site ID).

The drop-down list will show "None" as a default option. All other options will be populated with the following from EPOD_UDF_CONFIG:

- Value as EPL_CONFIG_ID.
- Descriptive text as EPL_DESCRIPTION

Button types will saved in the standard UDF Field XML notation, as per the following example:

```
<FIELD ID="{UDF Field ID}">
    <TEXT>{UDF Text}</TEXT>
    <FORMAT>BT</FORMAT>
    <REQUIRED>{UDF Required}</REQUIRED>
    <ACTION>{UDF Form ID}</ACTION>
</FIELD>
```

The curly brackets identify the fields from the configuration screen.

Parameter "Default" values will be set from a predefined list of values stored in EPOD_LIST_ITEMS, for a new List on EPOD_LISTS. For this change, the following fields are required:

- Owner Name (value PDAJOB.EPL_OWNER_NAME)

- Customer Name (value PDAJOB.EPL_CUSTOMER_NAME)
- Address 1 (value PDAJOB.EPL_ADDRESS_1)
- Address 2 (value PDAJOB.EPL_ADDRESS_1)
- Address 3 (value PDAJOB.EPL_ADDRESS_1)
- Address 4 (value PDAJOB.EPL_ADDRESS_1)
- Post Code (value PDAJOB.EPL_POSTCODE)
- Current Date (value PDAJOB.EPL_END_ACTUAL_DATE)

Note: This will be the same list as the list of Data-bound field values, developed recently. Any new values above should be added to this list.

Default parameters will saved in the standard UDF Field XML notation with a tag VALUE, as per the following example:

```
<FIELD ID="">
    <TEXT></TEXT>
    <SUBTEXT></SUBTEXT>
    <GROUP></GROUP>
    <POST></POST>
    <FORMAT></FORMAT>
    <REQUIRED></REQUIRED>
    <VALUE></VALUE>
</FIELD>
```

The UDF Preview panel of the Create/Edit pop-up will be modified to support the new Button types. Button fields (as opposed to Buttons on the form) will be displayed on a new line like normal fields. The button text will be the value in the Label. The button will be formatted to the same look and feel as Form buttons.

As part of product changes to this screen, the following changes will also be made to this preview form at this time, to more closely match the layout on the actual device:

- No colon after each label.
- Text fields 50% width float right.
- Text/Numeric fields on the same line as the label.
- Form title in a header at the top of the form, formatted as 95% width, white text on blue background.
- Check Boxes having ticks to the left.
- Leaving space at the bottom of the screen window and showing the form buttons at all times (if present).
- General font: smaller in the screen window.
- Sub-label present, half-size, under the appropriate field, float left 70% width.
- Post-text present, half-size, under the appropriate field, float right 30% width.

The user control UDFRender will be modified to achieve these preview changes

- 

New Preview style

The Configurable POD/POC report (ConfigPOD.aspx) will be modified to ignore Button field types - they will not be part of the completion document produced, which includes the data entered on the sub-form

1.2.4 Device

The following field will be added to the Job table EPOD_JOB:

- EPL_OWNER_ID - nvarchar(12)

The Data Access Layer object PDA_JOB will be modified to receive this field in the import from the server when requesting loads or receiving an update response. If not present, this should be defaulted to blank.

The Signature screen (Signature.js) will be modified to set the End Date and Time (EPL_END_ACTUAL_DATE and Time of EPOD_JOB) to the current date and time at the point of entering the screen. This will not be saved at this time. The setting of the End Date and Time after confirmation of the signature should remain.

UDF objects will be modified to support the following:

- Data-bound Default Values in tag "VALUE".
- Button fields.

The UDF Object (in method createUDFields2 of styling.js) will be changed to support these requirements.

Currently, the UDF object supports putting in the value from already-entered UDF forms, but not from the format itself. This will be modified to add the value from the format XML, so support defaulting from within the UDF format design. The VALUE tag within the completed or saved UDF form takes precedence over the value in the UDF format.

Additionally, the processing of this VALUE tag will support data-bound defaults.

The value of the VALUE tag will be checked to see if it exactly matches the following:

- Starts with "PDA"
- Then has any number of alphanumeric and underscore characters only
- Then has one "."
- Then has alphanumeric text and underscore characters only.

This will be checked with a regexp pattern as follows:

```
(PDA[a-zA-Z0-9_]+\.)?[a-zA-Z0-9_]+
```

Should this wholly match with no other characters, then the field value is considered data-bound.

In this case, the process will get the value from the data field (the text after the ".") from the data object (the text before the "."). If the data object or data field does not exist, then this will be caught and the value will be set to "".

Note: This is very similar to the existing data-bound field functionality.

The UDF Object supports many field types. This will be extended to support the Button field types "BT".

For Button field types, the process will get the UDF configuration indicated by the value in the ACTION tag (through the instantiation of the object PDA_UDF_CONFIG). If this configuration is not found, then the button will not be added. If found, then the Button field will be added like other fields, in the next row, utilising the standard button styling on the device style.

The child tags of the FIELD tag will be stored as now in the fieldObj properties of the object created. The UDF Form format (PDA_UDF_CONFIG.EPL_UDF_FIELDS) will be stored in this object, under the ACTION property.

A button click event handler will be added to, on clicking the button, check the VALUE and ACTION property values of fieldObj. These will be passed to a new generic function to display a pop-up UDF entry form.

The new function will be similar to existing forms already in place in the product - this change will centralise this into a single function that can be used for those forms that require it (popup.js). The existing areas that can be copied are:

- Pre/Post Service UDF (in SM_Service.js)
- Pre-Job UDF (in PreJobUDF.js)

- Container and Product UDF (in ColDel.js)

The function should receive parameters of the data objects in use (in this case, PDALOAD and PDAJOB) as well as the UDF Format and the completed UDF Form (if already completed).

The function will create a new full-screen window for the form entry. **Note:** It is important to ignore any button field specified in a sub-form - these can only be displayed in a main form.

This form must also include form buttons. It is expected in this form that the buttons will be:

- **Cancel** - Changes must be made to the Cancel button type to remove the returned UDF, in case the customer changes their mind after full or partial entry of the form. The popup window will then be closed.
- **Confirm** - this will validate the form as normal, checking that the required fields have been entered. If this fails, the form will highlight the information not entered and inform the customer. If this passes validation, the process will save the UDF and close the form, returning the entered UDF to the calling form.

When confirmed or cancelled, the form will return a status ("C" or "X", depending on whether the Confirm or Cancel button was clicked in the sub-form. The button object will store the returned XML in the VALUE fieldObj property, and extract the status into another property for checking at validation.

If the Button field has a REQUIRED tag set to "Y", then the status must not be "X". When validating the T&Cs form entry, this must be checked.

Note: This is generic coding - in this implementation, the **Gift Aid Registration** button will *not* be required.

When validating the T&Cs form and generating the UDF to return to the server (and on to C-TMS through the job update), the VALUE tag should be populated like any other field object i.e. from the value stored against the object.

For Button fields, the value will actually be resulting XML from the sub-form.

So, a form ("MYFORM") with a single normal TEXT field called "MYFIELD" that has "ABC123" entered into it would return the following XML:

```
<FORM NAME="MYFORM">
    <FIELD ID="MYFIELD">
        <TEXT>My Field</TEXT>
        <SUBTEXT></SUBTEXT>
        <GROUP></GROUP>
        <POST></POST>
        <FORMAT>T</FORMAT>
        <REQUIRED>N</REQUIRED>
        <VALUE>ABC123</VALUE>
    </FIELD>
</FORM>
```

A form ("MYFORM") with a single Button field "MYBUTTON" with a sub-form "MYSUBFORM", ID 321 (entering MYFIELD as above) that has been completed will return the following:

```
<FORM NAME="MYFORM">
    <FIELD ID="MYBUTTON">
        <TEXT>My Button</TEXT>
        <SUBTEXT></SUBTEXT>
        <GROUP></GROUP>
        <POST></POST>
        <FORMAT>BT</FORMAT>
        <REQUIRED>N</REQUIRED>
        <ACTION>321</ACTION>
        <VALUE>
            <FORM NAME="MYSUBFORM" CONFIRM="C">
                <FIELD ID="MYFIELD">
                    <TEXT>My Field</TEXT>
                    <SUBTEXT></SUBTEXT>
                    <GROUP></GROUP>
                    <POST></POST>
                    <FORMAT>T</FORMAT>
                    <REQUIRED>N</REQUIRED>
                    <VALUE>ABC123</VALUE>
                </FIELD>
            </FORM>
        </VALUE>
```


```
        </FIELD>
    </FORM>
```

Although more complex in this implementation, this will be the basis of the configuration of the Collection T&Cs and of the Gift Aid Registration form. Sample configurations have been created and are present in the project folder.

This UDF will be returned in the EPL_TNCS field to C-TMS.

2 Appendix A: Quote & Document References

Cost Details				
Activity	**Estimate No. of Days**	**No. of Days**	**Rate per Day (?)**	**Cost (? Exc. VAT)**
Requirements	0.00	0.00	0	?0.00
Change Request Evaluation	1.00	1.00	0	?0.00
Functional Specification	2.50	2.50	0	?0.00
Technical Specification	0.00	0.00	0	?0.00
Development	18.25	18.25	0	?0.00
Testing and Release	2.50	2.50	0	?0.00
Implementation	1.00	1.00	0	?0.00
Project Management	1.50	1.50	0	?0.00
TOTAL	26.75	26.75		?0.00

Estimate excludes training, release to live and go live support.

A.1 References

Ref No	Document Title & ID	Version	Date
1	Charity Gift Aid Declaration.docx		11/09/2018
2	Complete TNC Example.xml		26/09/2018

A.2 Glossary

Term	Definition
EPOD	Electronic Proof of Delivery. The OBS EPOD system is *CALIDUS* ePOD.
CALIDUS eSERV	The OBS mobile system to complete Service functionality in the field. This is part of the *CALIDUS* ePOD system.
PDA	The mobile device on which the C-ePOD system will run in the field. This can be a Phone, EDA or industrial PDA, running Android.
DAL	Data Access Layer. A mechanism for accessing data by the system that is removed from the application, allowing for simplified access and providing protection to the data, as only approved DAL methods can be used to modify it.
GPS	Global Positioning System. A mechanism of retrieving accurate positioning information in the form of Latitude and Longitude (Lat-Long) co-ordinates from a device.
GPRS, 3G, HSDPA, Data Service	All terms referring to mobile device network connectivity, and the speed at which the device connects to the internet.

A.3 Authorised By

Murray Middleton OBS Logistics Project Manager _____

Mike Raynor Gone for Good Limited Representative _____