Oracle Utilities Mobile Workforce Management

Plug-Ins Guide

Release 1.5.0.21

August 2013

ORACLE

Oracle Utilities Mobile Workforce Management, Release 1.5.0.21

Copyright © 1994, 2013 Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Chapter 1

Chapter 1

Plug-In Processing

This document outlines the Custom code (Plug-in) specifications for the Oracle Utilities Mobile Workforce Management applications. Most of the functionalities is specified and named as the base function (Pre/Post) in which will be provided as much as the base code/parameter(s) that is used within the application.

Topics include:

- **AWsCustomCodeDll**
- **ASvCustomCodeDll**
- **ARtrCustomCodeDll**
- **ACustomIcdsDll**
- **CECustomCodeDll**

AWsCustomCodeDll

The Workstation application utilizes the AWsCustomCodeDll.dll library for processing Plug-in functionalities. Plug-in functionalities will be the primary methodology used by the Implementation team. Any static object/pointer/call defined in Utility.dll library will be allowed to be used/called within any of the plug-in functionalities (i.e.: AUtil::ms_pDwDbConn, AUtil::ms_user, etc…). The different function calls are described in the following sections.

AWsPreProcessMIDCompleteFo::PreProcessMIDCompleteFo

```
BOOL PreProcessMIDCompleteFo(
CString& p_sOrderNum_i
)
```

Remarks:

This method is used to manipulate the field order fields (Flat File) as they were received from the Screen completion prior to insert/update into the XFoEx object before it can be sent to the Server application. It may entail truncating fields, combining fields, and/or changing the format of fields.

The parameter will be provided the order number, which is the name of the file in the *orders* directory. Any of the modification will be I/O to the file, any error occurs will be logged within the function, and the function will be returned as the TRUE/FALSE based on the requirements/ critical errors.

Return:

Parameter(s):

Field order number

Example:

- Overwrite the Tracking status based on the specific criteria that was made from the screen or a particular order/meter type.
- Split field into multiple tables, joint/move fields into a table, or load descriptions into order remarks.

AWsPostActionsAddDlg::PostActionsAddDlg

```
BOOL PostActionsAddDlg(
    CString& p_sOrderNum_i,
    BOOL& p_fLoadOrderInDetail_o
)
```

Remarks:

This method is used to manipulate the field order fields as they were received from the AddOrder dialog (DwAddOrder.def/MwAddorder.def) prior to insert/update into the XFoEx object before it can be sent to the Server application. It may entail truncating fields, combining fields, and/or changing the format of fields.

The parameter will be provided the order number which is the name of the file in the *orders* directory and the indication of loading the pickup order in detail/completion mode. Any of the modification will be I/O to the file, any error occurs will be logged within the function, and the function will be returned as the TRUE/FALSE based on the requirements/critical errors.

Return:

Parameter(s):

Field order number

Load order in completion mode flag

Example:

- Overwrite the field order/parent number based on the specific criteria that was made from the screen or a particular order/meter type.
- Rename the field order file based on the order name that was entered from the dialog.
- Split field into multiple tables, joint/move fields into a table, or load descriptions into order remarks
- Load indication into different field as the PICKUP_ORD_FLG

AWsPostCreateRelatedPickups::PostCreateRelatedPickups

```
BOOL PostCreateRelatedPickup(
    CString& p_sOrderNum_i
)
```

Remarks:

This method is used to manipulate the field order fields as they were received from the Related-Pickup screen prior to insert/update into the XFoEx object before it can be sent to the Server application. It may entail truncating fields, combining fields, and/or changing the format of fields.

The parameter will be provided the order number which is the name of the file in the *orders* directory and the indication of loading the pickup order in detail/completion mode. Any of the modification will be I/O to the file, any error occurs will be logged within the function, and the function will be returned as the TRUE/FALSE based on the requirements/critical errors.

Return:

Parameter(s):

- Field order number

Example:

- Overwrite the field order/parent number based on the specific criteria that was made from the screen or a particular order/meter type.
- Load indication into different field as the PICKUP_ORD_FLG
- Default specific custom field(s) from a table.
- Create a Meter Selections dialog if it's a Set meter order.
- Create a New Customer Info dialog if it's a new account.
- Prefix/suffix meter number or any field within the Set meter record.
- Populate spare columns in the DHTFOCMN record

AWsProcessCustomCrewFrameCmds::ProcessCustomCrewFrameCmds

```
BOOL ProcessCustomCrewFrameCmds(
    WPARAM wp,
    LPARAM lp
)
```

Remarks:

This method is used to process custom ICDs in the crew subsystem. Custom Icds can be sent to the crew subsystem by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD's ID
- Custom ICD object

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

AWsProcessCustomFoFrameCmds::ProcessCustomFoFrameCmds

```
BOOL ProcessCustomFoFrameCmds(
    WPARAM wp,
    LPARAM lp
)
```

Remarks:

This method is used to process custom ICDs in the field order subsystem. Custom Icds can be sent to the field order subsystem by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

Custom ICD's ID

Custom ICD object

Example:

Switch on the ICD ID and call methods to process the custom ICDs

AWsProcessCustomIcds::ProcessCustomIcds

```
BOOL ProcessCustomIcds(
    long p_iId_i,
    XIcd* p_pIcd_i,
    int& p_iSubSys_i
)
```

Remarks:

This method is used to process custom ICDs in the OnShipMail function. Custom Icds can be routed to the FO, CREW, MAIL, MAP, or SYS-MESSAGE subsystem by adding the appropriate code (AlistCtrl::eSubSystems) to the ProcessCustomIcds plug-in and the ICD will be not delete/ destroyed at the end of the function(OnShipMail). The custom ICD can also be processed in this plug-in instead of routing it to another thread, but the processing that can be done is limited. The base version of this method simply returns FALSE, since there are no custom ICDs in the base.

Any errors that occur may be logged within the function or logged by the calling function by setting the return message and returning FALSE.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD's ID
- Custom ICD object
- Subsystem's ID, which the ICD will be forward to.

Example:

- Switch on the ICD ID and route to the appropriate subsystem for processing

AWsProcessCustomMailFrameCmds:: ProcessCustomMailFrameCmds

```
BOOL ProcessCustomMailFrameCmds(
    WPARAM wp,
    LPARAM lp
)
```

Remarks:

This method is used to process custom ICDs in the mail subsystem. Custom Icds can be sent to the mail subsystem by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD's ID
- Custom ICD object

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

AwsProcessCustomMainfrmFrameCmds::ProcessCustomMainfrmFrameCmds

```
BOOL ProcessCustomMainfrmFrameCmds(
    WPARAM wp,
    LPARAM lp
)
```

Remarks:

This method is used to process custom ICDs in the main application(CMainFrame). Custom Icds can be sent to the main application by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD's ID
- Custom ICD object

Example:

Switch on the ICD ID and call methods to process the custom ICDs

AwsProcessCustomMapFrameCmds::ProcessCustomMapFrameCmds

```
BOOL ProcessCustomMapFrameCmds(
    WPARAM wp,
    LPARAM lp
)
```

Remarks:

This method is used to process custom ICDs in the map subsystem. Custom Icds can be sent to the map subsystem by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD's ID
- Custom ICD object

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

AwsPreCreateAssistOrder:: PreCreateAssistOrder

Remarks:

This plug-in will be added to the AWsCustomCodeDll project. It will be called after the screen has been validated and prior to inserting the new assist order in the database. This plug-in will give the implementation team the ability to further manipulate the XFoEx data prior to creating the order in the database.

Input

The XFoEx object should be passed into the plug-in, so that the field order data can be manipulated before insertion in to the database.

Output

No value will be returned from the plug-in. If any logging should occur, it should log the appropriate messages without the plug-in code.

Main Processing

The base plug-in will contain no code, but a return.

AwsGetGPSCoordinates:: GetGPSCoordinates

Remarks:

This plug-in will be used within a separate design at a later date to create a Custom GPS device at the time of GPSSupport object creation in the Station. This is needed in order to facilitate Custom GPS info for AVL timed transactions. Currently, Oracle Utilities Mobile Workforce Management already accommodates 3 custom GPS devices determined by the GPSSupport.ini parameter DEVICE_TYPE. Further customization can be added at a later time by setting this parameter value to CUSTOM and adding the plug-in point into the GPSSupportMain class to create and set its pointer to a custom device type.

AWsPostChangeOperator:: PostChangeOperator

Remarks:

This plug-in will be added to the AWsCustomCodeDll project. It will be called after the screen has been validated and prior to loading the new user options from the database. This plug-in will give the implementation team the ability to further manipulate the monitor dispatch area(s) prior to load the orders from the database.

Input

The string list object of the dispatch areas, indicator to reload the dispatch areas, and indicator to save the new dispatch areas to DHTLSTDA table locally, should be passed into the plug-in, so that the dispatch areas can be manipulated before reloading from the database.

Output

No value will be returned from the plug-in. If any logging should occur, it should log the appropriate messages without the plug-in code.

Main Processing

The base plug-in will contain no code, but a return.

AWsPostLogonProcess:: PostLogonProcess

Remarks:

This plug-in will be added to the AWsCustomCodeDll project. It will be called after the screen has been validated and prior to log on process to the Station. This plug-in will give the implementation team the ability to further manipulate any additional process once the user has successfully logged on to the Station.

Input

None.

Output

No value will be returned from the plug-in. If any logging should occur, it should log the appropriate messages without the plug-in code.

Main Processing

The base plug-in will contain no code, but a return.

AwsCustomOrderDeleteReassignReturnMsg::ProcessCustomDeleteReassignReturnMsg

Remarks:

This method is used to process the XIcdDeleteOrder, XIcdReassignFo, XIcdReturnFoAck ICDs in the MobileStation. This plug-in will give the implementation team the ability to further manipulate the process and the user notification message(s) based the order on the MobileStation. The base version of this method would have the existing process and simple user notification message(s).

Input

The XIcd object should be passed into the plug-in, so that the process and the message can be manipulated before deletion of the order file.

Output

List of Field order numbers and the user notification messages will be returned from the plug-in and the messages will be populated when it returns TRUE, otherwise no message will be shown.

Main Processing

The base plug-in will contain the existing code to process the ICDs and the simple user notification messages.

Example:

- · Add external field order/parent/meter number based on particular order/meter type.
- · Custom user message format.

ASvCustomCodeDll

The Server application utilizes the ASvCustomCodeDll.dll library for processing Plug-in functionalities. Plug-in functionalities will be the primary methodology used by the Implementation team. These are the different functions calls:

ASvBuildGeocodeAddresses::BuildGeocodeAddresses

```
BOOL BuildGeocodeAddresses (
    XFoEx& p_oFoEx_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i,
    CString& p_strStreetAddress_o,
    CString& p_strCityStateZip_o,
    CString& p_strCountry_o,
    CString& p_strMatchMode_o
)
```

Remarks:

This method is used to build the address fields that will be used by the GEOCODER to geocode the order. The GEOCODER requires 2 address fields: Street Address and City/State/Zip, plus the Country code. The base version of this plug-in will load DHTFOEXT.CUST_ADDR_1 into the street address field and DHTFOEXT.CUST_ADDR_3 concatenated with DHTFOCMN.ZIP_CODE into the city/state/zip field, default the country to "US", and default the match mode to "DEFAULT". If this works for a particular implementation, then no changes are needed; if not, this plug-in should be modified to load the geocode address fields with the appropriate data.

This plug-in is called from the ProcessMfFieldOrderIcd and ProcessMobilityCreatedFoIcd methods in the XThreadSvFoEx class. It is called before the field order is inserted in the database.

Returns:

TRUE: Auccess

FALSE: Error occurred in plug-in code – no geocoding will be performed and an error will be generated.

Parameter(s):

- Field Order object
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data
- Reference to the street address field (output)
- Reference to the city/state/zip field (output)
- Reference to the country field (output)
- Reference to the match mode field (output)

Example:

- Load CUST_ADDR_1 into Street address field
- Concatenate ZIP_CODE to CUST_ADDR_3 to build city/state/zip field
- Change the default country of "US" to something else
- Change the default match mode of "DEFAULT" to something

ASvCustomDispatchNotification::CustomDispatchNotification

```
BOOL CustomDispatchNotification(
    CString& p_strMessage_o,
    CString& p_strFoNumber_i,
    CString& p_strCrewId_i,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to perform customized code when an order is successfully dispatched to a mobile device. This plug-in is called from the ProcessDealFoTxnQueue method in the XThreadSvDispatch class after each order is updated to dispatched. It may contain custom dispatch related code like paging for emergencies.

Any errors that occur may be logged within the function or logged by the calling function by setting the return message and returning FALSE.

Returns:

TRUE: No plug-in code or success

FALSE: Error occurred in plug-in code

Parameter(s):

- Return error message
- Field order number of dispatched order
- Id of crew assigned to dispatched order
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Paging for emergency order

ASvCustomFieldOrderEODProcess::CustomFieldOrderEODProcess

```
BOOL CustomFieldOrderEODProcess(
    CString& p_strMessage_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to perform customized code during the End of Day process. This plug-in is called from the EodProcess method in the XThreadSvEod class before the main or normal field order archiving is carried out.

Any errors that occur may be logged within the function or logged by the calling function by setting the return message and returning FALSE.

Return:

TRUE: No plug-in code or success

FALSE: Error occurred in plug-in code

Parameter(s):

- Return error message
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Remove duplicate historical field orders

ASvFillFoSpares::FillFoSpares

```
void FillFoSpares(
    XFoEx& p_oFoEx_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to fill the Spare columns in the field order object with the appropriate data. This plug-in is called from the ProcessMfFieldOrderIcd, ProcessMobilityCreatedFoIcd, and PickupOrderNew methods in the XThreadSvFoEx class. It is called before the field order is inserted/updated in the database

Return:

Parameter(s):

Field Order object

Pointer to a database connection

Pointer to the thread

Pointer to the global server thread data

Example:

Set Customer address number as the spare column

Set default remarks

Duplicate the contact number

Store the order type description in a spare column

ASvModifyCrewMaintInfo::ModifyCrewMaintInfo

```
void ModifyCrewMaintInfo(
      XCrewMaintInfo& p_oCrewInfo_io,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to manipulate the crew maintenance information before it is sent to the scheduling module via the Router and XIM. This plug-in is called from the ProcessCrewUpdated method in the XThreadSvMisc class when a crew record is created/updated. It is called before the XIcdMobilityUpdatedCrew ICD is built with the XCrewMaintInfo object.

Return:

Parameter(s):

Crew maintenance information object

Pointer to a database connection

Pointer to the thread

Pointer to the global server thread data

Example:

- Populate the custom property fields in the crew maintenance information object
- Parse existing field (e.g. Pull state from City field and store in State field).

ASvModifySchedulingInfo::ModifySchedulingInfo

```
void ModifySchedulingInfo(
      XSchedulingInfo& p_oSchedInfo_io,
    XFoEx& p_oFoEx_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to manipulate the scheduling information before it is sent to the scheduling module via the Router and XIM. This plug-in is called from the ProcessSchedulingInfo method in the XThreadSvFo class when a field order is created/updated. It is called before the XIcdMobilitySchedFo ICD is built with the XSchedulingInfo object.

Return:

Parameter(s):

- Scheduling information object
- Field Order object
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Populate the custom property fields in the scheduling information object

ASvPostCreateOrderProcess::PostCreateOrderProcess

```
void PostCreateOrderProcess(
    XFoEx& p_oFoEx_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This plug-in is called from the ProcessMfFieldOrderIcd method in the XThreadSvFoEx class. It is called after the order has been successfully inserted into the database.

Returns:

Parameter(s):

- Field Order object
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Send notification to the Dispatcher(s) for a specific order type
- Send internal email to the Dispatcher(s)

ASvPostUpdateOrderProcess::PostUpdateOrderProcess

```
void PostUpdateOrderProcess(
    XFoEx& p_oFoEx_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This plug-in is called from the ProcessMfFieldOrderIcd method in the XThreadSvFoEx class. It is called after the order has been successfully updated in the database.

Returns:

Parameter(s):

- Field Order object
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Send notification to the Dispatcher(s) for a specific order type
- Send internal email to the Dispatcher(s)

ASvPreArchiveFieldOrdersEOD::PreArchiveFieldOrdersEOD

```
BOOL PreArchiveFieldOrderEOD(
    CString& p_strMessage_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to perform customized code during the End of Day process. This plug-in is called from the ArchiveFieldOrders method in the XThreadSvEod class before the main or normal field order archiving is carried out.

Any errors that occur may be logged within the function or logged by the calling function by setting the return message and returning FALSE.

Return:

TRUE: No plug-in code or success

FALSE: Error occurred in plug-in code

Parameter(s):

Return error message

Pointer to a database connection

Pointer to the thread

Pointer to the global server thread data

Example:

- Archiving host system orders only
- Expiration of field orders

ASvPreCompletionToRouter::PreCompletionToRouter

```
void PreCompletionToRouter(
    XFoEx& p_oFoEx_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This plug-in is called from the ProcessMobileCompletionIcd, ProcessFsmsFieldOrderIcd, and PickupOrderNew methods in the XThreadSvFoEx class. It is called immediately before "sending" the completion ICD to the Router.

Returns:

Parameter(s):

- Field Order object
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Set default value with a specific order type
- Trim/replace value within the order

ASvPreOrderCreateProcess::PreOrderCreateProcess

```
BOOL PreOrderCreateProcess(
    XFoEx& p_oFoEx_io,
    CString& p_strErr_o,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This plug-in is called from the ProcessMfFieldOrderIcd method in the XThreadSvFoEx class before the bulk of the create/update order processing has been done.

Any errors that occur may be logged within the function or logged by the calling function by setting the return message and returning FALSE.

Return:

TRUE: No plug-in code or success

FALSE: Error occurred in plug-in code

Parameter(s):

- Field order object
- Return error message
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Set default value with a specific order type
- Add column/table value(s) to match with Oracle Utilities Mobile Workforce Management database
- Trim/replace value within the order

ASvPreValidateMeterReqToRouter::PreValidateMeterReqToRouter

```
BOOL PreValidateMeterReqToRouter (
    XIcdMfValidateMeterReq& p_oIcd_i,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to process the Validate Meter Request from a mobile device. The base version of this method simply routes the ICD to the Router application for processing. This plug-in is called from the ProcessValidateMeterRequest method in the XThreadSvMiscEx class.

Any errors that occur may be logged within the function or logged by the calling function by setting the return message and returning FALSE.

Returns:

TRUE: No plug-in code or success

FALSE: Error occurred in plug-in code

Parameter(s):

- Validate Meter Request ICD object
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Directly call a Meter Management database to validate the request

ASvProcessAvlCustomIcds::ProcessAvlCustomIcds

```
BOOL ProcessAvlCustomIcds(
    XIcd& p_oIcd_i,
    CDatabase* p_pDb_i,
    BOOL& p_fAck_o,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to process custom ICDs in the AVL thread. Custom Icds can be sent to the AVL thread by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base. This plug-in is called from the Thread method in the XThreadSvAvl class.

If the custom ICD is guaranteed, the logic to generate an RfAck can be put in the plug-in or, using the passed in fAck flag and the DHTICDPR table, the RfAck can be generated automatically by the base code.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid ICD found

Parameter(s):

- Custom ICD object
- Pointer to a database connection
- Acknowledgement flag
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

ASvProcessCustomIcds::ProcessCustomIcds

```
BOOL ProcessCustomIcds(
    XIcd& p_oIcd_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to process custom ICDs in the ProcessIcds thread. Custom Icds can be routed to the AVL, FO, Logoff, Logon, or Miscellaneous thread by adding the appropriate code to the ProcessCustomIcds plug-in. The custom ICD can also be processed in this plug-in instead of routing it to another thread, but the processing that can be done is limited. This plug-in does not have access to a database connection. The base version of this method simply returns FALSE,

since there are no custom ICDs in the base. This plug-in is called from the ProcessExtendedIcds method in the XThreadSvProcessEx class.

Any errors that occur may be logged within the function or logged by the calling function by setting the return message and returning FALSE.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD object
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Switch on the ICD ID and route to the appropriate thread for processing

ASvProcessCustomWarnings::ProcessCustomWarnings

```
BOOL ProcessCustomWarnings(
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to generate custom warnings in the Warnings thread. The base version of this method simply returns TRUE, since there are no custom warnings in the base. This plug-in is called from the Thread method in the XThreadSvWarning class.

Any errors that occur may be logged within the function or logged by the calling function by returning FALSE.

Return:

TRUE: No plug-in or successful generation of custom warning

FALSE: Error occurred generating custom warnings

Parameter(s):

- Pointer to database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Crew has been working too many hours

ASvProcessFoCustomIcds::ProcessFoCustomIcds

```
BOOL ProcessFoCustomIcds(
    XIcd& p_oIcd_i,
    CDatabase* p_pDb_i,
    BOOL& p_fAck_o,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to process custom ICDs in the FO thread. Custom Icds can be sent to the FO thread by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base. This plug-in is called from the ProcessExtendedIcds method in the XThreadSvFoEx class.

If the custom ICD is guaranteed, the logic to generate an RfAck can be put in the plug-in or, using the passed in fAck flag and the DHTICDPR table, the RfAck can be generated automatically by the base code.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD object
- Pointer to a database connection
- Acknowledgement flag
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

ASvProcessLogoffCustomIcds::ProcessLogoffCustomIcds

```
BOOL ProcessLogoffCustomIcds(
    XIcd& p_oIcd_i,
    CDatabase* p_pDb_i,
    BOOL& p_fAck_o,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to process custom ICDs in the Logoff thread. Custom Icds can be sent to the Logoff thread by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base. This plug-in is called from the ProcessExtendedIcds method in the XThreadSvLogoffEx class.

If the custom ICD is guaranteed, the logic to generate an RfAck can be put in the plug-in or, using the passed in fAck flag and the DHTICDPR table, the RfAck can be generated automatically by the base code.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD object
- Pointer to a database connection
- Acknowledgement flag
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

ASvProcessLogonCustomIcds::ProcessLogonCustomIcds

```
BOOL ProcessLogonCustomIcds(
    XIcd& p_oIcd_i,
    CDatabase* p_pDb_i,
    BOOL& p_fAck_o,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to process custom ICDs in the Logon thread. Custom Icds can be sent to the Logon thread by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base. This plug-in is called from the ProcessExtendedIcds method in the XthreadSvLogonEx class.

If the custom ICD is guaranteed, the logic to generate an RfAck can be put in the plug-in or, using the passed in fAck flag and the DHTICDPR table, the RfAck can be generated automatically by the base code.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD object
- Pointer to a database connection
- Acknowledgement flag
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

ASvProcessMiscCustomIcds::ProcessMiscCustomIcds

```
BOOL ProcessMiscCustomIcds(
    XIcd& p_oIcd_i,
    CDatabase* p_pDb_i,
    BOOL& p_fAck_o,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to process custom ICDs in the miscellaneous thread. Custom Icds can be sent to the miscellaneous thread by adding the appropriate code to the ProcessCustomIcds plug-in. The base version of this method simply returns FALSE, since there are no custom ICDs in the base. This plug-in is called from the ProcessExtendedIcds method in the XThreadSvMiscEx class.

If the custom ICD is guaranteed, the logic to generate an RfAck can be put in the plug-in or, using the passed in fAck flag and the DHTICDPR table, the RfAck can be generated automatically by the base code.

Any errors that occur should be logged within the plug-in. If FALSE is returned, it is assumed to be an invalid ICD.

Return:

TRUE: Custom ICD was processed

FALSE: No plug-in code or invalid custom ICD found

Parameter(s):

- Custom ICD object
- Pointer to a database connection
- Acknowledgement flag
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Switch on the ICD ID and call methods to process the custom ICDs

ASvReScheduleOrderProcess::ReScheduleOrderProcess

```
void ReScheduleOrderProcess(
    XFoEx& p_oFoEx_i,
    BOOL& p_fReschedule_io,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This plug-in is called from the ProcessMfFieldOrderIcd method in the XThreadSvFoEx class to indicate if the order may need to be rescheduled.

Return:

Parameter(s):

- Field order object
- Reschedule flag
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Set/Fill with the appointment window code based on the schedule start time / early start time
- Reset Reschedule flag if order should not be rescheduled

ASvSetCreatedUpdatedFoFields::SetCreatedUpdatedFoFields

```
void SetCreatedUpdatedFoFields(
    XFoEx& p_oFoEx_io,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This method is used to manipulate the field order fields as they were received from the Router prior to inserting/updating the order in the database. This plug-in is called from the ProcessMfFieldOrderIcd method in the XThreadSvFoEx class prior to calling the Insert/Update database method.

Return:

Parameter(s):

- Field order object
- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Set/Fill with the meter rate code based on the status or order type
- Set Taken by, District, Division, or Service Area
- Duplicate/Spit Address
- Set default Remarks
- Add column/table value(s) to match with Oracle Utilities Mobile Workforce Management database

ASvProcessMfCompletionIcd::ProcessMfCompletionIcd

```
BOOL ProcessMfCompletionIcd(
    XFoSchedule& p_oFoSch_i,
    XFoEx& p_oFoEx_o,
    BOOL& p_fUseFoEx_i,
    CDatabase* p_pDb_i,
    XThreadSvBase* p_pThread_i,
    XSvThreadData* p_pThreadData_i
)
```

Remarks:

This plug-in is called from the ProcessMfCompletionIcd method in the XThreadSvFoEx class. It is called immediately before the database update call.

Returns:

Parameter(s):

- Schedule object in database
- Field Order object from Mf
- Indicator to use data from Mf

- Pointer to a database connection
- Pointer to the thread
- Pointer to the global server thread data

Example:

- Set completion date time based on the data from Mf
- Empty/blank the Crew field if it's voided?

ARtrCustomCodeDll

The Router application utilizes the ARtrCustomCodeDll.dll library for processing Plug-in functionalities. Plug-in functionalities will be the primary methodology used by the Implementation team. These are the different functions calls:

ARtrProcessCustomIcds::ConvertCustomTransactions

```
BOOL ConvertCustomTransactions(
const CString& p_strDataFormat_i,
const CString& p_strOrigTransId_i,
const CString& p_strConvertToId_i,
XIcd& p_oMsgToConvert_i,
int& p_iErrorCode_o,
const CString& p_strExtConnName_i,
XList<XIcd>&     p_IcdListing_o,
XRtrAckWaitBox* p_pNonAckedMsgLog_io
)
```

Remarks:

This method is used to process custom ICDs created by the project implementation teams in the router. This enables the project implementation team to process or format custom ICDs to the customer specific xml or other flat file messages. Inside of this function, custom-switching code can be written to process multiple custom ICDs and call individual custom created classes to post-process the data and format any individual custom ICD to the appropriate xml or other format. The plug-in also provides access to the the AckBox (XRtrAckWaitBox object) so the custom transactions can be guaranteed.

Return:

TRUE: Custom ICD was converted successfully

FALSE: Custom ICD was not converted successfully

Parameter(s):

- Data Format as specified in the router.ini file. For example: XML or text
- The original transaction ID from the originating application.
- The destination or receiving application transaction ID
- The original ICD that needs to be converted
- Error Code 0 means there was no error. Error Code -1 means there is an error occurred and the base class XMessageRouter will check for the error that occurred and post it to the error log. Any other error codes will tell the router to ignore the message and no need to further process it.
- External Connection for this message to be sent to.
- The converted ICD should be added the p_IcdListing_o to be placed in the appropriate destination.

Example:

Switch on the original ICD ID and call custom methods to convert them to either XML or other format messages and place the converted ICD to the return list.

ARtrCustomConversion::PreConvertFSMSOrderIssue

```
BOOL PreConvertFSMSOrderIssue(
const CString& p_strDataFormat_i,
int& p_iErrorCode_o,
const CString& p_strDstConnName_i,
const CString& p_strExtConnName_i,
XFoEx& p_oFoEx_io
)
```

Remarks:

This plug-in is called from the ConvertFSMSOrderIssue method in the XRtrConversions class. This method is used to preprocess orders created in Oracle Utilities Mobile Workforce Management before sending to the external systems. This allows the implementation team to change the data in the XFoEx object if it's necessary and can set the error code greater than 0 and that will tell the router to ignore the message. For example, you can check for the service point in the XFoEx object and set the p_iErrorCode_o to be greater than 0 and the message will be ignored and will not be sent to the external systems. Please compare the Dst Conn and Ext Connection Name to make sure that you would like to ignore the transaction.

Return:

TRUE: Everything was fine and no errors occurred

FALSE: Errros occurred and check for the error.

Parameter(s):

- Data Format as specified in the router.ini file. For example: XML or text
- Error Code 0 means there was no error. Error Code -1 means there is an error occurred and the base class XMessageRouter will check for the error that occurred and post it to the error log. Any other error codes will tell the router to ignore the message and no need to process it further.
- External destination Connection for this message to be sent to. This field should be the current destination from the external connection list. . For example, orders created in Oracle Utilities Mobile Workforce Management may need to be sent to the CSS, OMS, SCHED. But this field will only contain CSS if this message pass is for CSS only. This field is populated by the settings in the DHTTXNPR table. This determines from the ICD level which ICD should go to which external connections.
- The full external connection list will contain the list of external connections specified in the DHTFOTYP table. So if it is specified in the DHTFOTYP external application list with CSS,OMS,SCHED. This field will contain that full list. This field determines from the field order level which field order type should go to which external connection.
- The XFoEx object

Example:

- Check for specific fields in the XFoEx object and make data changes or ignore the message if certain criteria are met.

ARtrCustomConversion::PreConvertFSMSOrderStatus

```
BOOL PreConvertFSMSOrderStatus(
const CString& p_strDataFormat_i,
int& p_iErrorCode_o,
const CString& p_strDstConnName_i,
const CString& p_strExtConnName_i,
XFoEx& p_oFoEx_io
)
```

Remarks:

This plug-in is called from the ConvertFSMSFoStatus method in the XRtrConversions class. This method is used to preprocess Oracle Utilities Mobile Workforce Management order status before sending to external systems. This allows the implementation team to change the data in the XFoEx object if it's necessary and can set the error code greater than 0 and that will tell the router to ignore the message. For example, you can check for the service point in the XFoEx object and set the p_iErrorCode_o to be greater than 0 and the message will be ignored and will not be sent to the external systems. Please compare the Dst Connection Name and Ext Connection name to make sure that you would like to ignore the transaction.

Return:

TRUE: Everything was fine and no errors occurred

FALSE: Errros occurred and check for the error.

Parameter(s):

- Data Format as specified in the router.ini file. For example: XML or text
- Error Code 0 means there was no error. Error Code -1 means there is an error occurred and the base class XMessageRouter will check for the error that occurred and post it to the error log. Any other error codes will tell the router to ignore the message and no need to process it further.
- External destination Connection for this message to be sent to. This field should be the current destination from the external connection list. . For example, orders created in Oracle Utilities Mobile Workforce Management may need to be sent to the CSS, OMS, SCHED. But this field will only contain CSS if this message pass is for CSS only. This field is populated by the settings in the DHTTXNPR table. This determines from the ICD level which ICD should go to which external connections.
- The full external connection list will contain the list of external connections specified in the DHTFOTYP table. So if it is specified in the DHTFOTYP external application list with CSS,OMS,SCHED. This field will contain that full list. This field determines from the field order level which field order type should go to which external connection.
- The XFoEx object

Example:

- Check for specific fields in the XFoEx object and make data changes or ignore the message if certain criteria are met.

ARtrCustomConversion::PreConvertFSMSOrderComplete

```
BOOL PreConvertFSMSOrderComplete(
const CString& p_strDataFormat_i,
int& p_iErrorCode_o,
const CString& p_strDstConnName_i,
const CString& p_strExtConnName_i,
XFoEx& p_oFoEx_io
)
```

Remarks:

This plug-in is called from the ConvertFSMSOrderComplete method in the XRtrConversions class. This method is used to preprocess Oracle Utilities Mobile Workforce Management order completion before sending to external systems. This allows the implementation team to change the data in the XFoEx object if it's necessary and can set the error code greater than 0 and that will tell the router to ignore the message. For example, you can check for the service point in the XFoEx object and set the p_iErrorCode_o to be greater than 0 and the message will be ignored and will not be sent to the external systems. Please compare the Dst Conn and Ext Connection Name to make sure that you would like to ignore the transaction.

Return:

TRUE: Everything was fine and no errors occurred

FALSE: Errros occurred and check for the error.

Parameter(s):

- Data Format as specified in the router.ini file. For example: XML or text
- Error Code 0 means there was no error. Error Code -1 means there is an error occurred and the base class XMessageRouter will check for the error that occurred and post it to the error log. Any other error codes will tell the router to ignore the message and no need to process it further.
- External destination Connection for this message to be sent to. This field should be the current destination from the external connection list. . For example, orders created in Oracle Utilities Mobile Workforce Management may need to be sent to the CSS, OMS, SCHED. But this field will only contain CSS if this message pass is for CSS only. This field is populated by the settings in the DHTTXNPR table. This determines from the ICD level which ICD should go to which external connections.
- The full external connection list will contain the list of external connections specified in the DHTFOTYP table. So if it is specified in the DHTFOTYP external application list with CSS,OMS,SCHED. This field will contain that full list. This field determines from the field order level which field order type should go to which external connection.
- The XFoEx object

Example:

- Check for specific fields in the XFoEx object and make data changes or ignore the message if certain criteria are met.

ARtrDetermineEmergencyOrder::DetermineEmergencyOrder

```
BOOL DetermineEmergencyOrder(
XFoEx* p_pFoEx_i,
BOOL& p_fEmerOrder_o
)
```

Remarks:

This method is used to set the passed in emergency order flag. The flag should be set to TRUE, if the order should be written to the emergency order queue in the Server for processing; otherwise, set the flag to FALSE. If the flag is set, return TRUE; otherwise return FALSE (base implementation) and the order type will be used to make the determination.

Return:

The return value indicates if the passed emergency order flag was set. If the flag was set within the plug-in, return TRUE; otherwise return FALSE.

Parameter(s):

- Point to the Field order object
- Reference to the emergency order flag

Example:

- Set the emergency order flag to indicate whether the order ICD should be written to the emergency order queue or the regular order queue for processing in the Server.

ARtrCustomConversion::PreConvertFoStatusExToClick

```
BOOL PreConvertFoStatusExToClick(
XIcdFoStatusEx* p_pIcd_io
)
```

Remarks:

This plug-in is called from the ConvertMobilityOrderStatusUpdate method in the XRtrConversions class. This method is used to preprocess FoStatusEx ICDs before sending the status data to Click. This allows the implementation team to manipulate the data in the XIcdFoStatusEx object if it's necessary. If the ICD should not be sent to Click, the plug-in should return FALSE and the ICD will be ignored.

Return:

TRUE: Send the ICD data to Click

FALSE: Ignore the ICD and do not send to Click

Parameter(s):

- Pointer to the XIcdFoStatusEx ICD

Example:

- Check for specific fields in the XIcdFoStatusEx and determine if the ICD should be ignored
- Make data changes to the ICD data before processing.

ARtrCustomConversion::PreConvertMobilitySchedFoToClick

```
BOOL PreConvertMobilitySchedFoToClick(
XIcdMobilitySchedFo* p_pIcd_io
)
```

Remarks:

This plug-in is called from the ConvertMobilityOrderCr method in the XRtrConversions class. This method is used to preprocess MobilitySchedFo ICDs before sending the order data to Click. This allows the implementation team to manipulate the data in the XIcdMobilitySchedFo object if it's necessary. If the ICD should not be sent to Click, the plug-in should return FALSE and the ICD will be ignored.

Return:

TRUE: Send the ICD data to Click

FALSE: Ignore the ICD and do not send to Click

Parameter(s):

- Pointer to the XIcdFoStatusEx ICD

Example:

- Check for specific fields in the XIcdFoStatusEx and determine if the ICD should be ignored
- Make data changes to the ICD data before processing.

ACustomIcdsDll

The Oracle Utilities Mobile Workforce Management applications utilize the ACustomIcdsDll.dll library for creating and processing custom ICDs. When a new ICD class is created for a project implementation, it will be added to this DLL project. This DLL project is linked in with AwsCustomCodeDll, AsvCustomCodeDll, and ArtrCustomCodeDll.

All references to the custom ICDs will be made through the Plug-ins provided in the custom DLLs.

To ensure that the project teams do not create a custom ICD with the same ID as a base ICD, the project teams are asked to restrict their custom ICD Ids to a specific range. The same holds true with custom transactions that will be sent/received by the Oracle Utilities Mobile Workforce Management Router application; their code should fall in the specified custom code range.

- Custom ICD Ids – 900 through 999
- Custom Inbound Transaction (from external applications) codes – 0900 – 0999
- Custom Outbound Transaction (to external applications) codes – 1900 – 1999

If the project teams adhere to these ranges, there should be no conflicts between the base ICDs/transaction and custom ICDs/transactions.

CECustomCodeDll

The CE Station application utilizes the CECustomCodeDll.dll library for processing Plug-in functionalities. Plug-in functionalities will be the primary methodology used by the Implementation team. These are different functions call:

ACEPreProcessTableDownload::PreProcessTableDownload

```
BOOL PreProcessTableDownload(
CWnd* p_pParent_i
)
```

Remarks:

This method is used to manipulate any validation/customize requirement prior to the table download request icd (XicdUpdateTable) is being sent to the Server application.

Return:

TRUE: Custom process/dialog was processed

FALSE: Invalid custom process, Station will be shut down.

Parameter(s):

- Pointer of the main window.

Example:

- Populate a validation dialog.
- Verify MW user equipment that was previous saved/downloaded.

ACEByteToUni::ParamsToUni

```
static int ParamsToUni(
PBYTE& p_pbSingle,
PBYTE& p_pbUni,
Int p_iCnt
)
```

Remarks:

This method is used to manipulate any data conversion of the icd that is received from the Server application.

Return:

TRUE: Custom process was processed

FALSE: Invalid custom process.

Parameter(s):

- Buffer in Byte
- Buffer in Unicode
- Number of fields is passed in w/ the buffer

Example:

- lXIcd_ID_MAIL_MESSAGE - ACEByteToUni::ParamsToUni(pbData_by,pbData,5);

ACEByteToUni::ByteToUni

```
int ByteToUni(
PBYTE& p_pbSingle,
PBYTE& p_pbUni
)
```

Remarks:

This method is used to manipulate any data conversion of the icd that is received from the Server application.

Return:

TRUE: Custom process was processed

FALSE: Invalid custom process.

Parameter(s):

- Buffer in Byte
- Buffer in Unicode

Example:

- lXIcd_ID_MISSED_APPT_WARNING

ACEUniToByte::ParamsToByte

```
static int ParamsToByte(
PBYTE& p_pbUni,
PBYTE& p_pbSingle,
int p_iLen,
int p_iCnt
)
```

Remarks:

This method is used to manipulate any data conversion of the icd that is being sent to the Server application.

Return:

TRUE: Custom process was processed

FALSE: Invalid custom process.

Parameter(s):

- Buffer in Unicode
- Buffer in Byte
- Length of the buffer (after conversion)
- Number of fields is passed in w/ the buffer

Example:

- lXIcd_ID_AVL

ACEUniToByte::ParamsToByte

```
int UniToByte(
PBYTE& p_pbUni,
PBYTE& p_pbSingle
)
```

Remarks:

This method is used to manipulate any data conversion of the icd that is being sent to the Server application.

Return:

TRUE: Custom process was processed

FALSE: Invalid custom process.

Parameter(s):

- Buffer in Unicode
- Buffer in Byte

Example:

- lXIcd_ID_AVL

ACECustomWirelessConnectivity::GetConnectivity

```
CString  GetConnectivity()
```

Remarks:

This method is used to obtain the current type of network card connectivity on the particular CE device.

Return:

"SERIAL": This is a serial LAN Active Sync connection.

"USB": This is a serial LAN Active Sync connection.

"WLAN": This is a Wireless LAN network connection.

"GPRS": This is a GPRS WWAN connection.

"OFFLINE: err string": There is no current network connection. The err string portion has the reason (if any) why no connection was obtained.

Parameter(s):

- none

Example:

- ACELogonDlg.cpp: Autodetect of the login type

ACECustomWirelessConnectivity::IsConnected

```
BOOL IsConnected()
```

Remarks:

This method is used to pass a boolean back to the base code determining current network connectivity *status*.

Return:

TRUE: CE device is connected to network.

FALSE: CE device is notconnected to network.

Parameter(s):

- none

Example:

- *XThreadWirelessConnectivity*.cpp: determine if the connection has been lost

ACECustomWirelessConnectivity::IsWired

```
BOOL IsWired()
```

Remarks:

This method is used to pass a boolean back to the base code determining current network wired status. This may vary depending on the customer needs. Certain processes on the CE such as table download are only performed while wired.

Return:

TRUE: CE device is “wired”.

FALSE: *CE* device is not “wired”.

Parameter(s):

- none

Example:

- ACELogonDlg.cpp: Prevent table download if not wired

ACECustomProcs::PreProcessLogon

```
BOOL PreProcessLogon(
  CWnd* p_pParent_I
)
```

Remarks:

This method is used to pass a *boolean* back to the base code prior to logon to determine whether it is OK to proceed with logon. During this call, a custom screen may be displayed to interact with the user to perform custom pre-login actions. This may vary depending on the customer needs. This may include actions such as initiating GPRS or EDACS connectivity.

Return:

TRUE: OK to proceed *with* login.

FALSE: Abort login and *close* program.

Parameter(s):

- Pointer of the main *window*.

Example:

- Starting the GPRS VPN connection *prior* to login.

ACEPostCreatePickupOrder::PostCreatePickupOrder

```
BOOL PostCreatePickupOrder(
CString& p_sOrderNum_i
)
```

Remarks:

This method is used to manipulate the field order fields in the field order flat file before it is sent to the Server application for creation. It may entail truncating fields, combining fields, and/or changing the format of fields.

The parameter will be provided the order number, which is the name of the file in the *orders* directory and the indication of loading the pickup order in detail/completion mode. Any of the modification *will* be I/O to the file, any error occurs will be logged within the function, and the function will be returned as the TRUE/FALSE based on the requirements/critical errors.

Return:

Parameter(s):

Field order number

Example:

- Overwrite the field order/parent number based on the specific criteria that was made from the screen or a particular order/*meter* type.
- Load indication into different field as the *PICKUP*_ORD_FLG
- Default specific custom field(s) from a *table*.
- Prefix/suffix meter number or any field *within* the Set meter record.
- Populate spare columns in the DHTFOCMN record

CECustomIcdsDll

The Oracle Utilities Mobile Workforce Management (CE) applications utilize the CECustomIcdsDll.dll library for creating and processing custom ICDs. When a new ICD class is created for a project implementation, it will be added to this DLL project. This DLL project is linked in with CECustomCodeDll.

All references to the custom ICDs will be made through the Plug-ins provided in the custom DLLs.

To ensure that the project teams do not create a custom ICD with the same ID as a base ICD, the project teams are asked to restrict their custom ICD Ids to a specific range. The same holds true with custom transactions that will be sent/received by the Oracle Utilities Mobile Workforce Management Router application; their code should fall in the specified custom code range.

- Custom ICD Ids – 900 through 999
- Custom Inbound Transaction (from external applications) codes – 0900 – 0999
- Custom Outbound Transaction (to external applications) codes – 1900 – 1999

If the project teams adhere to these ranges, there should be no conflicts between the base ICDs/transaction and custom ICDs/transactions.