Oxlo OnRamp™

Installation and Developers Guide

Copyright © 2004-2007, Oxlo Systems Inc.
All rights reserved

Oxlo Systems Inc.
11001 West 120th Avenue, Suite 300
Broomfield, CO 80021
720-890-7545

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

Version Number: 20060310

Table of Contents

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

1 Section I—Oxlo OnRamp Installation Guide

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

2 Overview

This document provides an overview of Oxlo OnRamp™, Oxlo's DSP communications package. It provides descriptions of what OnRamp is, how it works, what it provides and what a DSP must do to implement OnRamp. The target audiences for this document are technical staff and management of Oxlo's DSP Partners.

2.1 Supported Platforms

OnRamp is tested and certified on the following platforms:

- Windows® Server 2003™
- Windows® XP™
- Windows® 2000™
- Windows® NT™ Server 4.0
- SCO Open Server ™ v5.0.5-5.0.7
- SCO Open Server™ v6.0.0
- Linux® RedHat 3.0
- Linux® RedHat 4.0
- Mandrake 9.1,9.2
- Mandrake 10.0-10.2
- Mandriva 2006
- Java

Additionally, Oxlo will certify and support OnRamp on these additional platforms, based on client needs, such as:

- Linux® (Debian, Redhat, Mandrake, etc.)
- Mixed systems (e.g. mixing Windows and Linux/Unix platforms)

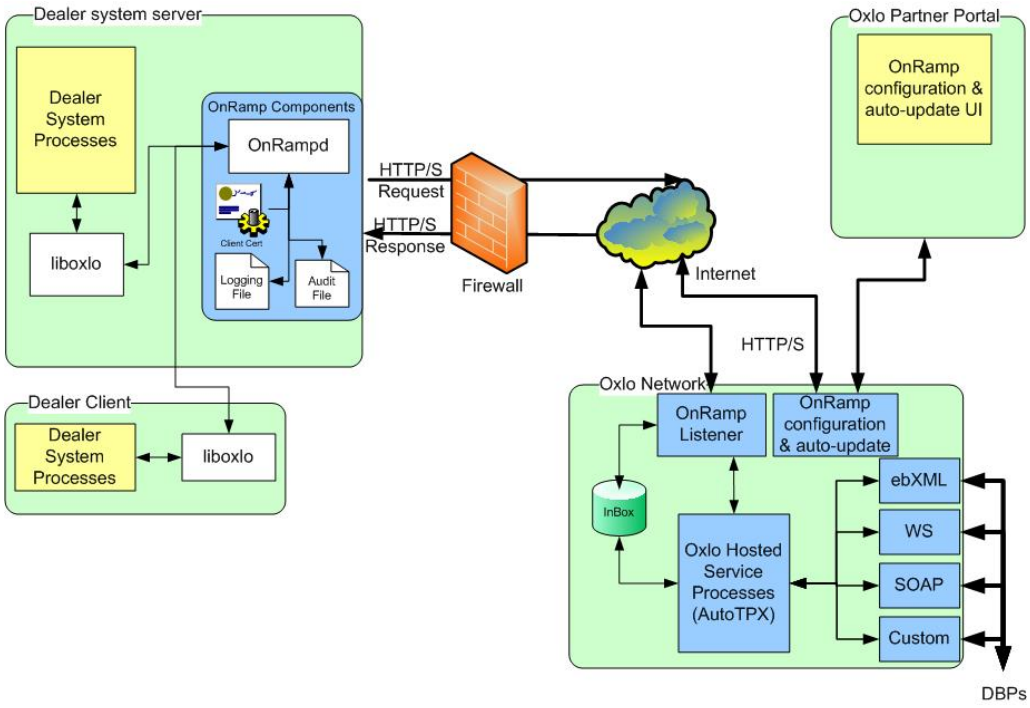
Please contact your Oxlo account manager directly regarding certification and support for these additional platforms.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

3 What is OnRamp?

Oxlo provides Dealer Systems Providers (DSPs) with a package for communicating between their applications and Oxlo. This package, called OnRamp, is provided to DSP partners for the purposes of accelerating and standardizing communications with Oxlo. Oxlo, in turn, implements the disparate and unique communications protocols required by Dealer Business Partners such as OEMs and Lenders (DBPs). OnRamp need only be implemented a single time to gain access to all DBPs connected to the Oxlo Automotive Retail Network, thus eliminating costly development and maintenance of multiple communications protocols.

The following picture illustrates the high-level components of OnRamp and their relationship to dealer systems.



OnRamp is a simple, secure, reliable, multi-platform communications product, consisting of a set of software libraries and executables that provide a single, consistent API. OnRamp is “Firewall Friendly,” being designed to live behind firewalls, and does not require any special networking setup such as static IP addresses or active listeners.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

OnRamp implements a secure and reliable messaging protocol that includes the following features:

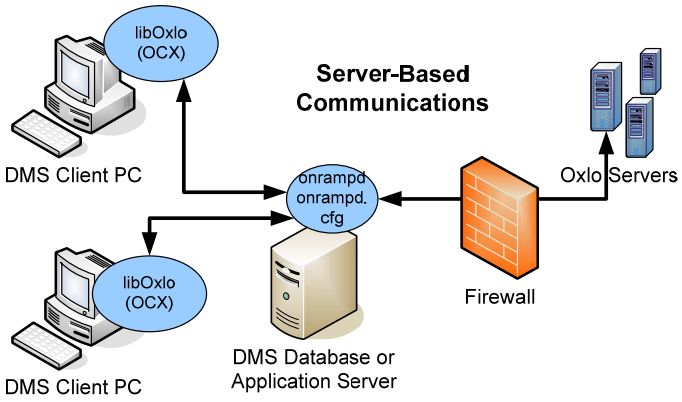
- Duplicate message detection and elimination
- Non-repudiation—so that it can be verified that the sender and the recipient were, in fact, the parties who send or receive a message
- Audit Trail tracking of every message
- Active monitoring of connectivity with integrated systems
- Synchronous/Asynchronous two-way communications
- Automatic payload compression and decompression
- Firewall friendly capabilities—so that communications are only initiated on a request/response basis from dealer systems
- Auto-update and remote management
- User friendly status messages

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

4 OnRamp Deployment Topologies

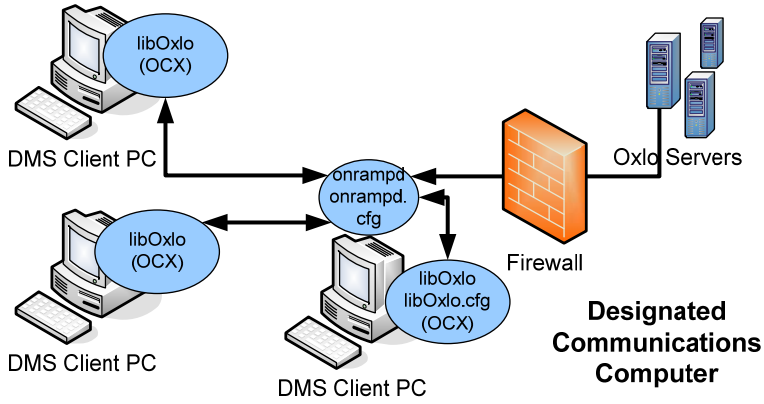
4.1 Server Based Communications

In a server-based deployment model, the OnRamp server components will be installed on the main application server and the OnRamp client components will be installed on other computers wishing to use the communications channel.



4.2 Designated Communications Computer

In environments without a centralized server, any machine on the network may be used as the communications server for OnRamp, including a machine that is also used as a client PC. The chosen communications machine will have both OnRamp server components and OnRamp client components installed on it, and the other machines on the network will have OnRamp client components installed on them.



Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

5 Installation and Key Components

OnRamp is composed of 2 components, liboxlo and onrampd, that must be installed. The Windows platform also has additional API wrappers such as the ActiveX control and java wrapper.

An OnRamp installation can be classified as either an OnRamp client or an OnRamp server. An OnRamp Server is the computer in your deployment that will send and receive all messages to and from the Oxlo central communications hub. Any system that accesses OnRamp functionality via calls to the liboxlo API is classified as a client. This is typically a DMS, F&I or CRM system. These can be installed on the same machine if it is to serve both uses. OnRamp components are intended to be bundled with the DSP's software and installed as part of the DSP's install/upgrade process.

5.1 OnRamp Installation Procedures

OnRamp components are installed in a single directory for both Unix and Windows installations. The following steps will guide you through the OnRamp installation.

5.1.1 Windows

OnRamp Server Installation - Complete these steps only on systems designated as an OnRamp communications server.

- First unzip OnRamp2xx_2xx to a single directory (for example C:\OnRamp)
- Open a command-line window (start button -> run – cmd.exe)
- Change working directory (cd) to the directory where you extracted the OnRamp files
- Execute "onrampd.exe –i" - The command should return "Service installed successfully." If this is not the response, contact Oxlo Technical support for assistance in troubleshooting.
- Open the Windows Service Control Manager by choosing Start/Run and entering the command "services.msc /s"
- The newly installed service will be listed as "Oxlo OnRamp Service". Locate the service in the list of services, select the service, right-click and choose Properties.
- Set the Startup Type for the service to be Automatic.
- Close the Properties window to apply the setting.
- Setup configuration files
    - Copy the onrampd.cfg provided to you by Oxlo Systems into C:\OnRamp
    - Change the value of the 'OnRampId' line to the OnRamp id provided to you by Oxlo Systems. An OnRampId identifies a unique installation of OnRamp. **Note: all further updates to onrampd.cfg are managed centrally by Oxlo and should not be configured by hand.**
    - Copy the *dspname*.p12 provided to you by Oxlo Systems into C:\OnRamp
- Start the OnRamp service from the Windows Service Control Manager

Client (liboxlo) Installation - Complete these steps on all systems that will access libolxo. This may include the communications server.

- First unzip OnRamp2xx_2xx to a single directory (for example C:\OnRamp)
- Register the OnRamp ActiveX control
    - Open a command-line window (start button -> run – cmd.exe)
    - Change working directory (cd) to the directory where where you extracted the OnRamp files
    - Execute "regsvr32 OnRampActiveX.ocx" – The command should return "DllRegisterServer in OnRampActiveX.ocx succeeded." If this is not the response, contact Oxlo Technical support for assistance in troubleshooting.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

- Update system PATH environment variable - This step updates the PATH environment variable so that applications can locate liboxlo.dll without having to copy it to multiple locations on the file system.
    - Open the “System Properties” dialog box (Start button->Control Panel->System)
    - Select the “Advanced” tab
    - Click on the “Environment Variables” button at the bottom
    - Select “Path” in the “System variables” pane
    - Click the “Edit” button
    - Edit the “variable value” by adding “C:\OnRamp;” (or your specific installation location) to the beginning of the string. (note the ‘ ; ‘ at the end of C:\OnRamp;)
    - Click OK on to apply changes
- Set “onrampd-location” environment variable. This environment variable identifies the server location where your OnRamp service is running.
    - Open the “System Properties” dialog box (Start button->Control Panel->System)
    - Select the “Advanced” tab
    - Click on the “Environment Variables” button at the bottom
    - Click on “New” at the bottom under “System Variables”
    - Variable name = “onrampd-location” (this is case sensitive, do not include the quotes)
    - Variable value = *Servername or IP address for your OnRamp server* e.g. “localhost”

5.1.2 Linux/Unix

Note: SCO OSR 504, 505 and 506 must have prngd installed for OnRamp to work properly. OSR 507 and 6.0 have a random number generator installed by default.

6 Download the random number generator from SCO/Caldera utilizing ftp and the following URL: ftp://ftp2.caldera.com/pub/skunkware/osr5/vols/prngd-0.9.23-VOLS.tar

- Untar the package to a temp directory
- Run the program “custom”, choose ‘Software’, ‘Install New’, ‘From Host’ at the prompts.
- Select ‘Media Image’ as your Media Device.
- Type in the path to the temp directory where you untarred the package.
- Select ‘prngd’ as the software to install
- To start prngd, use the following command line: “/usr/local/sbin/prngd /var/run/egd-pool”
- You can verify that the random number generator is started by finding it in the process list: "ps -ef | grep prngd"
- OnRamp looks for the random number generator at one of the following locations: /var/run/egd-pool, /dev/egd-pool, or /etc/egd-pool

OnRamp Server Installation - Complete these steps only on systems designated as an OnRamp communications server

- extract the OnRamp2xx_2xx.tar file into /opt dir (creates /opt/OnRamp directory)
    - su or login as root
    - cd /opt
    - tar xvf <full-path-to-OnRamp2xx_2xx.tar>
- setup symbolic link for liboxlo.so to liboxlo.so.2.0x.0 (ln -s liboxlo.so.2.0x.0 liboxlo.so)
- setup LD_LIBRARY_PATH to include directory of liboxlo.so link -this allows your application to find the liboxlo.so library without maintaining multiple copies of the library
- Add the following lines to .bashrc or .profile for user that the calling (DSP) application runs as "export LD_LIBRARY_PATH=/opt/OnRamp:$LD_LIBRARY_PATH"

- Setup configuration files
    - Copy the onrampd.cfg provided to you by Oxlo Systems into /opt/OnRamp
    - Change the value of the 'OnRampId' line to the OnRamp id provided to you by Oxlo Systems. Systems. **Note: all further updates to onrampd.cfg are managed centrally by Oxlo and should not be configured by hand.**
    - Copy the *dspname*.p12 provided to you by Oxlo Systems into /opt/OnRamp
- start onrampd with ./onrampd as a user that can daemonize, open accept and outgoing sockets

Client (liboxlo) Installation - Complete these steps on all systems that will access libolxo. This may include the communications server.

- extract the OnRamp2xx_2xx.tar file into /opt dir (creates /opt/OnRamp directory)
    - su or login as root
    - cd /opt
    - tar xvf <full-path-to-OnRamp2xx_2xx.tar>
- setup symbolic link for liboxlo.so to liboxlo.so.2.0x.0 (ln -s liboxlo.so.2.0x.0 liboxlo.so)
- setup LD_LIBRARY_PATH to include directory of liboxlo.so link -this allows your application to find the liboxlo.so library without maintaining multiple copies of the library
- Add the following lines to .bashrc or .profile for user that the calling (DSP) application runs as
    - "export LD_LIBRARY_PATH=/opt/OnRamp:$LD_LIBRARY_PATH"
    - "export onrampd_location= *Servername or IP address for your OnRamp server* e.g. "export onrampd_location=localhost"

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

7 Upgrading OnRamp (version 1.3x_1.3x to 2.0x_2.0x)

Upgrading the OnRamp from version 1xx to 2xx is a simple process. All OnRamp 2.0 files live in a single directory (ie c:\OnRamp or /opt/OnRamp). This file structure eliminates multiple library versions and other inaccuracies in deployment. This is a one time change to an existing OnRamp 1.x deployment.
Note: Before beginning an upgrade please provide Oxlo systems with a copy your current octd.cfg file. For development/integration installations please contact your Integration Services Consultant. For production installs please contact Oxlo Operations (877.463.1965).

7.1 Windows

- Deregister the service
    - Shutdown the octd service
    - Open a command-line window (start button -> run – cmd.exe)
    - Change working directory (cd) to the directory where octd.exe is located
    - Execute “octd.exe –d”
- Deregister the activeX control if you have installed and registered it in the past.
    - Stop all applications using the Oct Active X control.
    - Open a command-line window (start button -> run – cmd.exe)
    - Change working directory (cd) to the directory where OCT_ActiveX.ocx is located.
    - Execute “regsvr32 /u OCT_ActiveX.ocx”
- Remove PATH entry for liboxlo.dll
- Remove all instances of liboxlo.dll, octd.exe, and OCT_ActiveX.ocx, testPing.exe, testMenu.exe, liboxlo.cfg and octd.cfg.
- Follow the OnRamp 2.0 install instructions.

7.2 Unix

- Stop the octd daemon
- Remove LD_LIBRARY_PATH entry for liboxlo.so location
- Remove all instances of octd , liboxlo.so/liboxlo.so.1.32.0, liboxlo.cfg and octd.cfg.
- Follow the OnRamp 2.0 install instructions.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

8 Configuration

OnRamp is controlled by a set of configuration files that specify settings for OnRamp to function. While the configuration files are simple text files, the information should not be edited by hand. Oxlo will supply a set of configuration files to its partners. The configuration files are managed by Oxlo Systems and will automatically be updated upon starting onrampd.

OnRamp utilizes two specific configuration files:

8.1 onrampd.cfg

The onrampd.cfg file is used to set parameters for OnRamp operation. Most settings in the onrampd.cfg file are not meant to be edited by partners. Oxlo systems will distribute an OnRampId to partners for every unique installation of OnRamp. This is the only parameter that should be edited by hand. All other updates to onrampd.cfg are managed remotely by Oxlo Support.

8.2 userMessageMap.cfg

The userMessageMap.cfg file contains end-user friendly error messages that can be customized to a particular installation. By implementing a simple error reporting mechanism, a DSP can provide comprehensive error messages to its users that can be tailored to be specific to the screens or activities that the end-user is interacting with as part of communications. This file is not editable and it automatically updated with parameters managed by Oxlo Support.

8.3 Auto Update of Onramp Configuration

As described above configuration files are managed centrally by Oxlo Support and are auto updated at the dealer site. Additionally OnRamp auto update allows Oxlo support to push software updates to OnRamp. Please note that this auto update functionality only allows Oxlo to update OnRamp files and poses no risk to other applications on the host system.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

9 Section II—Oxlo Developers Guide

10 Key Concepts

To implement a communications module using OnRamp, several key concepts should be understood. These concepts are:

- Sessions
- Sender IDs
- Application IDs
- InBoxes
- Document Types
- Document Versions
- Response Handling

The following paragraphs cover these important concepts.

10.1 Sessions

Sessions are used by OnRamp to manage memory via a single construct. Sessions are designed to be "single-use", that is a session is created, used and destroyed for each individual message being sent or received. Sessions contain dynamic memory that is allocated specifically for a messaging transaction, and once that transaction is finished, the memory should be freed to prevent errors from subsequent inadvertent use. Additionally, Sessions contains state and parameter information about a particular messaging transaction that should be disposed of when that transaction is completed.

The API functions that relate to initializing and cleaning up sessions are the following (see the API reference for a complete list of parameters):
createSession()
destroySession()

note: The OXLO_SESSION structure that is passed in to these two functions should be allocated from the hosting software's memory management. The structure itself is **NOT** allocated/created by the createSession function.

Comment [GG1]: Are we referring to the OS here? Is this more info then our partners need or will understand

10.2 SenderIDs

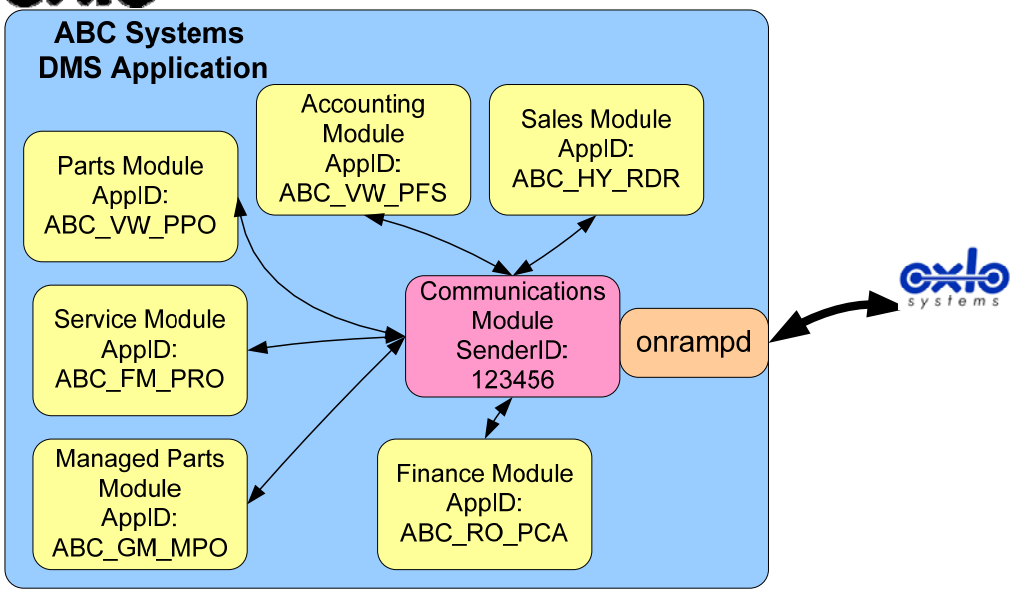
Sender IDs play 2 important roles in OnRamp (and subsequently in Oxlo's hosted service environment): they identify who originated a message (so that specific validation and data completion rules can be applied) and they form half of the 2-part key that identifies an InBox (see section 10.4). Oxlo will assign a unique Sender ID for each installation of OnRamp, typically one per dealer. Oxlo will also assign a SenderID specifically for its partners that should be used during development and testing. Note that Sender ID's are critical for correct functioning of OnRamp and Oxlo's hosted service environment, therefore only SenderID's that Oxlo has assigned should be used.

When communications code is running, the SenderID should be set programmatically per session using the API function setSenderID().

10.3 Application IDs

Application IDs are the second part of a 2-part key (the other being the Sender ID) that OnRamp and Oxlo's hosted service environment use to manage, route and identify messages flowing through the system. Specifically, business rules, InBoxes, and other configuration information rely on correct Sender IDs and Application IDs. While Sender IDs typically identify a single dealer, Application IDs are used to identify the individual application or functional module that is effecting the communications. The following picture illustrates this:

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.



This diagram portrays a hypothetical dealer that utilizes a DMS application from vendor ABC Systems that has 6 different application modules. The different modules of the DMS application all use specific Application IDs that refer to their functional area, but the same Sender ID (assigned to the dealer). Messages being sent by the various modules will be received by Oxlo's hosted service environment and processed according to their destination (VW, GM, RouteOne, etc.) Messages received from Dealer Business Partners (DBPs) such as Ford, GM, RouteOne, etc., will be routed to the InBox and indexed based on the Sender ID and Application ID. For example, Acknowledge Parts Order messages received from a DBP will be routed to the InBox and indexed by 123456/ABC_VW_APO, while Credit Decisions received from a financing source will be routed to the InBox and indexed by 123456/ABC_RO_PCD. As a result of using multiple Application IDs, the various modules of the DMS application can effectively filter out messages that they don't care about (e.g. the Parts module won't have to worry about Credit Decision messages).

> Application IDs will be assigned during the integration process. Your integration consultant will provide you with the specific application IDs required for your integration project.

10.4 InBoxes

As the previous 2 sections have indicated, messages received from DBPs are processed by Oxlo's hosted service environment and placed into the InBox indexed by the combination of SenderID and ApplicationID. When checking for or retrieving messages, OnRamp must be told the SenderID and ApplicationID to correctly identify the messages in question. OnRamp identifies messages by Message IDs that are returned from the API call getMessageIds. To retrieve a message, a valid Message ID must be provided to either the getMessageAsBuffer or getMessageAsFile function.

Oxlo's hosted service environment is always ready to receive messages from DBPs and place them into the InBox. Each message received is assigned a unique message ID. To manage the messages in the InBox, DSP applications should "delete" messages when they are finished processing the message. Deleting a message

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

will prevent that message id from being returned by subsequent calls to getMessageIds. If a message is not deleted, then subsequent calls to getMessageIds will include that message id in the returned list.

10.5 Document Types and Versions

When sending messages via OnRamp, both the type of the message (the document type) and the version of the message must be specified. The document version is set programmatically per session using the API function setDocVersion(). The document type can only be set programmatically as part of a sendFile() or sendBuffer() call.

10.6 Response Handling

OnRamp provides a set of calls to provide detailed response messages and support the automation of response and error handling. All function calls of OnRamp utilize a simple return code that can be used programmatically to determine whether an error has occurred. Non-zero return values indicate an error condition (See the OnRamp API guide for a complete list of error codes). In the event that an error has occurred, additional functions can be utilized to get more information (see the OnRamp API guide for a complete listing of the parameters for each of these calls):

- getResponseCode – This function returns the HTTP code from the last HTTP request. Typical return values will be 200, 404, 500, etc. Note that this function can be called whether an error has occurred or not. This response code should not be used to determine whether or not the last operation succeeded or failed.
- getResponseReason – This function returns a pointer to the message string associated with the HTTP response. Typically this message will be "OK" for successful requests, or will contain error text corresponding to the actual error. Note that OnRamp has allocated memory to hold the response reason, and that once the current session is deleted, the memory will be freed. Applications wishing to use the response reason should copy it to their own memory space to avoid programmatic failures.
- getResponseData – This function returns the body of the last session's response. The body will only contain data if the specific function call must return data to the caller (for example, getMessageIDs and getMessageAsBuffer). Note that OnRamp has allocated memory to hold the response message, and that once the current session is deleted, the memory will be freed. Applications wishing to use the Response Data should copy it to their own memory space to avoid programmatic failures.
- getUserMessage – This function returns a "user-friendly" version of an error response. OnRamp utilizes response reason as a key to look up the user friendly message in the userMessageMap.cfg file.

Comment [GG2]: Updates around this for correlated message

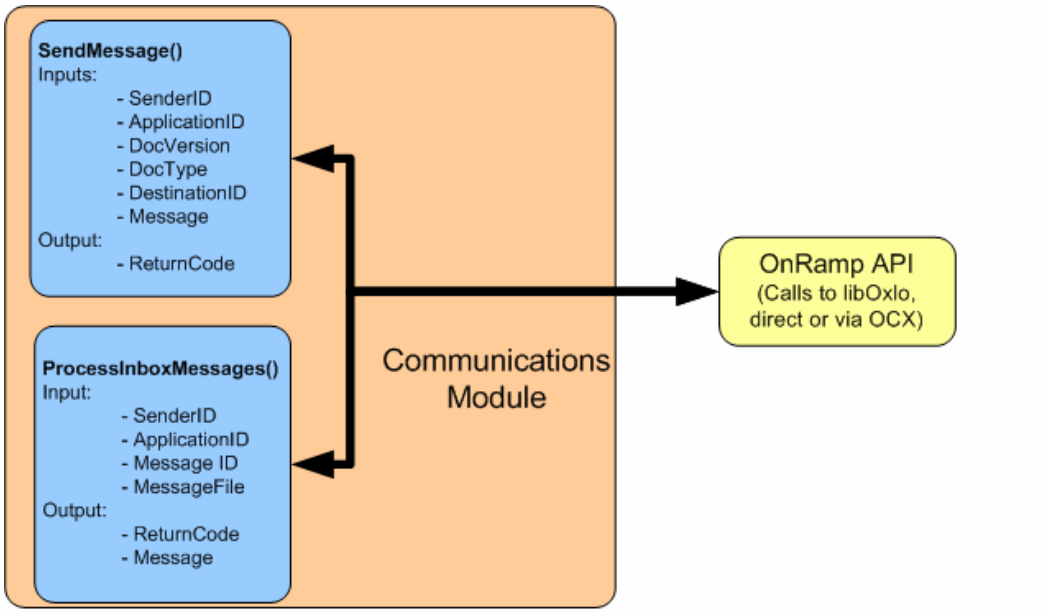
Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

11 Simple Communications Module

Implementing a simple OnRamp communications module will consist of writing code to support two basic functions:

- Sending messages
- Receiving messages

The following diagram illustrates the architecture of a basic communications module:



11.1 Sending Messages

11.1.1 SendMessage Description

The structure of the SendMessage function encapsulates all the management necessary to create and destroy an Oxlo Session, set the correct parameters for message transmission, send the message and handle the corresponding response.

The parameters passed to the function are as follows:

- SenderID
- ApplicationID
- DocVersion
- DocType
- DestinationID
- MessageFile/Buffer

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

High level flow of SendMessage implementation is as follows:

- Create session
  - If session creation fails log/display an error message
  - If successful move on to set parameters
- Set session parameters
  - SenderId
  - ApplicationId
  - DocVersion
- Call SendFile/SendBuffer with required parameters; MessageFile/Buffer, DestinationId, DocType
  - If SendFile response is success (function return = 0) log/show success
  - If SendFile response is not success (function returns ≠ 0) log response and handle appropriately
    - If -20 GetUserMessage and display to end user (if an end user is waiting on transaction) *or*
    - Implement retry logic *or*
    - Escalate
- Destroy session

As the above example shows, sending messages involves establishing a session, setting session variables, invoking a method to send the message, and then destroying the session. The OxloSession object is used to hold buffers and other information about a single message transaction, so it should be destroyed after a single use. Please see Appendix A for SendMessage sample code.

11.2 Receiving Messages

11.2.1 ProcessInboxMessages Description

The structure of the ProcessInboxMessages function encapsulates all the management necessary to create and destroy an Oxlo Session, set appropriate session parameters, check whether any message are waiting for download, if message(s) are waiting download the corresponding payloads and delete them from the InBox.

The parameters passed to the function are as follows:

- SenderID
- ApplicationID

High level flow of ProcessInboxMessages implementation is as follows

- Create session1
  - If session creation fails log/display an error message
  - If successful move on to set parameters
- Set session parameters
  - SenderId
  - ApplicationId
- Call getMessageIds with session1
  - If getMessageIds response is success (function return = 0) log/show success
    - For each id returned from getMessageIds
    - Create session2
    - Call getMessageAsBuffer or getMessageAsFile
      - If getMessageAsBuffer/getMessageAsFile succeeded (function return = 0) then

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

        - process payload (retrieved using getResponseData())
            - Determine the payload type(s) retrieved
            - Save the payload as the appropriate file type and pass to the calling application for processing
        - delete message from inbox
            - Create session3
            - Set session parameters
                - SenderId
                - ApplicationId
            - Call deleteMessage with messageId to delete
                - If deleteMessage succeeded (function return = 0) log/show success
                - If deleteMessage failed (function return ≠ 0) then log error
            - Destory session 3
        - If getMessageAsBuffer/getMessageAsFile failed (function returns ≠ 0) then log error
    - Destroy session2
- If getMessageIds response is not success (function returns ≠ 0) log response and handle appropriately
    - user is waiting on transaction *or*
    - Implement retry logic *or*
    - Escalate
- Destroy session1

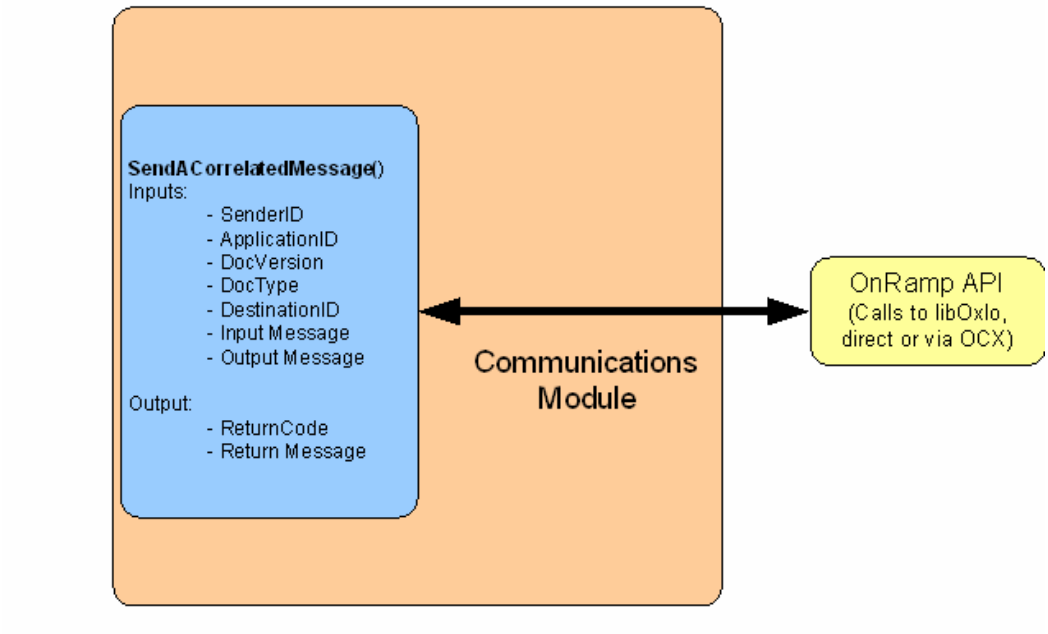
As the above example shows, receiving messages involves establishing a session, setting session variables, invoking a method to identify messages available for retrieval, processing the available messages and deleting them from the inbox. The OxloSession object is used to hold buffers and other information about a single message transaction, so it should be destroyed after a single use. Please note how in the above example a session is created and destroyed for every call to OnRamp within the function. Please see Appendix A for ProcessInboxMessages sample code.

12 Correlated Messaging

Comment [GG3]: Add verbiage here around correlated messaging, give example

For some business processes the simple messaging implementation described above is not appropriate. In the simple messaging model sending and receiving messages for a given business process are either independent of one another or loosely coupled. For example a parts order message may be sent and a corresponding parts shipment notification may be received at no specified time interval and with no direct correlation token. A correlation token is usually a simple element such as a message or conversation identifier that exists in all related messages for a given business process. In a correlated business messaging scenario for each message sent the system will expect a specific return message/messages, usually within a specified time frame, that reference the correlation token of the original message. A common example here is the Chrylser Financial AutoOrigination system. In this case an AOInput or ProcessCreditContract message is sent to Chrylser from the dealer system. The dealer system will always receive either a ConfirmBOD or AcknoweldgeCreditContract response that references the original messageId created on transmission. For scenarios like this Oxlo has created a method for the DSP system to handle the more complex model of correlated messaging.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.



12.1.1 SendCorrelatedMessage Decsription

High level flow of SendACorrelatedMessage implementation is as follows:

- Create session
  - If session creation fails log/display an error message
  - If successful move on to set parameters
- Set session parameters
  - SenderId
  - ApplicationId
  - DocVersion
- Call SendCorrelatedMessageAsFile/Buffer with required parameters; MessageFile/Buffer, DestinationId, DocType
  - If SendFile response is success (function return = 0)
    - SaveSessionResponseData as file/buffer; this contains the returned, correlated mesage
  - If SendFile response is not success (function returns ≠ 0) log response and handle appropriately
    - If -20 GetUserMessage and display to end user (if an end user is waiting on transaction) *or*
    - Implement retry logic *or*
    - Escalate
- Destroy session

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

13 Application Programming Interface

This section describes in additional detail the Application Programming Interface of the Oxlo Communications Toolkit known as OnRamp. The OnRamp package consists of a linkable library that provides communications functionality, a Windows Service (or a Unix daemon) that utilizes the library to provide heartbeat and audit capabilities, sample source code and documentation. The purpose of OnRamp is to minimize the amount of time a DSP will need to invest to communicate between their application and the Oxlo's hosted service environment. By using OnRamp, developers at a DSP will be able to invoke simple API calls without regard to communications protocols, header formats, auditing and management. Should the implementation of OnRamp change in the future, the API will be preserved so that DSP's will not need to adapt their code to changing communications protocols

13.1 Overview

The DLL's expose several C methods that allow the user of the DLL to test connectivity, send documents (both from a buffer as well as from a file) to an Oxlo hosted service environment.

13.2 Oxlo Data Structures

- OXLO_SESSION – This is a data structure that contains session and state information. It is initialized, maintained and used by the OnRamp methods to eliminate the need for external memory management for submitted data and/or results. No client applications need manipulate or manage any of the information within this data structure.

13.3 List of Methods

The following methods are the complete public interface to Oxlo OnRamp™. The methods can be divided into 5 sections:

- Session Management
- Accessors for HTTP response information
- Utility functions
- Administrative functions
- Message sending functions

```
/* create and destroy Oxlo sessions to simplify *
** memory management of connections and result data */
• int createSession(OXLO_SESSION *);
• void destroySession(OXLO_SESSION * session);

/* accessors for http response code, data and reason */
• unsigned getResponseCode(OXLO_SESSION * session);
• unsigned char * getResponseData(OXLO_SESSION * session);
• unsigned char * getResponseReason(OXLO_SESSION * session);
• unsigned char * getUserMessage(OXLO_SESSION * session);
• char * getMessageId(OXLO_SESSION * session);

• int sendAuditPing(OXLO_SESSION * session);
• int liboxloVersion(void);
• int octdVersion(void); (deprecated replaced by onRampdVersion)
• int onRampdVersion(void);

/* functions to override default values from configuration */
```

- `void setSenderId(OXLO_SESSION * session, char *inSenderId);`
- `void setApplicationId(OXLO_SESSION * inSession, char *inAppId);`
- `void setDocVersion(OXLO_SESSION * session, float inDocVersion);`

```
/* functions to send either a file or a buffer */
```

- `int sendFile(OXLO_SESSION * session, char *fname, char *destId, char *docType);`
- `int sendBuffer(OXLO_SESSION * session, const unsigned char *data, const long dataLen, char *destId, char *docType);`

```
/* functions to check, retrieve and delete documents from the Inbox */
```

- `int getMessageIds(OXLO_SESSION * session);`
- `unsigned long ** getMessageIdArray(OXLO_SESSION * session)`
- `int getMessageAsBuffer(OXLO_SESSION * session, unsigned long msgId);`
- `int getMessageAsFile(OXLO_SESSION * session, unsigned long msgId, char * filename);`
- `int deleteMessage(OXLO_SESSION * session, unsigned long msgId);`

13.3.1 Error Codes

```
  0 = ONRAMP_SUCCESS = connectivity is good or operation succeeded
  1 = ONRAMP_GENERIC_ERROR = Generic error
  2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
  3 = ONRAMP_AUTH_FAILED = User authentication failed on server
  4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
  5 = ONRAMP_CONNECT_FAILED = Could not connect to server
  6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
  7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
  8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
  9 = ONRAMP_REDIRECT_REQUESTED = Redirect request

  0 = ONRAMP_SUCCESS = successfully completed operation
 -1 = ONRAMP_CANNOT_READ_CONFIG = could not read configuration file (liboxlo.cfg)
 -2 = ONRAMP_CANNOT_READ_LICENSE = could not read License from configuration file
 -3 = ONRAMP_CANNOT_READ_SENDERID = could not read DefaultSenderId from
      configuration file
 -4 = ONRAMP_CANNOT_READ_APPID = could not read ApplicationId from
      configuration file
 -5 = ONRAMP_CANNOT_READ_DOCVERSION = could not read DefaultDocVersion from
      configuration file
 -6 = ONRAMP_CANNOT_READ_AUTOBOD_FORMAT = could not read AutoBODFormat from
      configuration file
 -7 = ONRAMP_CANNOT_READ_SCHEME = could not read Scheme from configuration file
 -8 = ONRAMP_CANNOT_READ_HOST = could not read Host from configuration file
 -9 = ONRAMP_CANNOT_READ_PORT = could not read Port from configuration file
-10 = ONRAMP_CANNOT_READ_BOD_URL = could not read BOD-Location from
      configuration file
-11 = ONRAMP_CANNOT_READ_AUDIT_PING_URL = could not read AUDIT-PING-Location
      from configuration file
-12 = ONRAMP_CANNOT_READ_INBOX_URL = could not read INBOX-Location from
      configuration file
-13 = ONRAMP_CANNOT_READ_MSG_DOMAIN = could not read Message-Domain from
      configuration file
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
-19 = ONRAMP_CANNOT_OPEN_SOCKET_LAYER = could not initialize socket layer
-20 = ONRAMP_REQ_REJECTED = request was rejected
-21 = ONRAMP_REQ_DEFERRED = request was deferred
-22 = ONRAMP_CANNOT_OPEN_FILE = could not open file
-23 = ONRAMP_CANNOT_ATTACH_BODY = could not attach file as body
-24 = ONRAMP_CANNOT_ATTACH_BUFFER = could not attach buffer as body
-25 = ONRAMP_CANNOT_ALLOC_ENOUGH_MEMORY = malloc failure for sending buffer(s)
-26 = ONRAMP_CANNOT_DETERMINE_FILE_SIZE = cannot get file size from file system.
-30 = ONRAMP_ERR_HASH_FAILED = could not generate hash value of file
-90 = ONRAMP_UNABLE_TO_FIND_ONRAMPD = unable to connect to onrampd
-91 = ONRAMP_UNABLE_TO_AUTH_ONRAMPD = unable to authenticate with onrampd
-92 = ONRAMP_IPC_SEND_ERROR = unable to send request to onrampd
-93 = ONRAMP_IPC_RCV_ERROR = unable to process response from onrampd
-94 = ONRAMP_NOT_ENOUGH_INFO_TO_PROCESS = unable to process request
      not enough info to complete processing
-95 = ONRAMP_INVALID_SESSION
-98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
-99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid
```

13.3.2 Session Management

13.3.3 createSession

The createSesion function is called to begin a connection session to the Oxlo hosted service environment. Upon success a pointer is returned to an OXLO_SESSION structure. This structure maintains the state, session data, response code and response data. Always call destroySession before exiting to clean up any allocated memory or connections.

Return values:

0 = ONRAMP_SUCCESS = successfully created an oxlo session
-1 = ONRAMP_CANNOT_READ_CONFIG = could not read configuration file (liboxlo.cfg)
-2 = ONRAMP_CANNOT_READ_LICENSE = could not read License from configuration file
-3 = ONRAMP_CANNOT_READ_SENDERID = could not read DefaultSenderId from configuration file
-4 = ONRAMP_CANNOT_READ_APPID = could not read ApplicationId from configuration file
-5 = ONRAMP_CANNOT_READ_DOCVERSION = could not read DefaultDocVersion from configuration file
-6 = ONRAMP_CANNOT_READ_AUTOBOD_FORMAT = could not read AutoBODFormat from configuration file
-7 = ONRAMP_CANNOT_READ_SCHEME = could not read Scheme from configuration file
-8 = ONRAMP_CANNOT_READ_HOST = could not read Host from configuration file
-9 = ONRAMP_CANNOT_READ_PORT = could not read Port from configuration file
-10 = ONRAMP_CANNOT_READ_BOD_URL = could not read BOD-Location from configuration file
-11 = ONRAMP_CANNOT_READ_AUDIT_PING_URL = could not read AUDIT-PING-Location from configuration file
-12 = ONRAMP_CANNOT_READ_INBOX_URL = could not read INBOX-Location from configuration file

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

-13 = ONRAMP_CANNOT_READ_MSG_DOMAIN = could not read Message-Domain from configuration file
-19 = ONRAMP_CANNOT_OPEN_SOCKET_LAYER = could not initialize socket layer

```
int createSession(OXLO_SESSION *);
```

13.3.4 destroySession

The destroySession function takes a pointer to an OXLO_SESSION structure. destroySession closes the connection and frees all of the allocated resources associated with this session including response data, response codes and session resources.

Important: This releases the response data. The response data must have been copied to a local memory before you destroy the session if you need to process the response further.

```
void destroySession(OXLO_SESSION * session);
```

13.3.5 HTTP Responses

13.3.6 getResponseCode

The getResponsecCode function returns the HTTP code from the last HTTP request.
Return Value:

unsigned value of the http code returned from last request

```
unsigned getResponseCode(OXLO_SESSION * session)
```

13.3.7 getResponseData

The getResponseData function returns a pointer to the body of the last HTTP response.
Return Value:

unsigned char * to the body of the last HTTP response.

```
unsigned char * getResponseData(OXLO_SESSION * session)
```

13.3.8 getResponseReason

The getResponseReason function returns a pointer to the string that was returned with the response.
Return Value:

Unsigned char * to the reason returned with the response code.

```
unsigned char * getResponseReason(OXLO_SESSION * session);
```

13.3.9 getUserMessage

The getUserMessage function returns a pointer to the user message associated with the http reason returned with the response. This user message is mapped to a reason in the userMessageMap.cfg file. There may be a global userMessageMap.cfg file for the onrampd as well as a localized version located in the directory of the calling process of liboxlo. The localized version overrides the user message populated by the global user message map.
Return Value:

```
unsigned char * getUserMessage(OXLO_SESSION * session);
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

13.3.10 getMessageId

The getMessageId function returns a pointer to a zero-terminated char string that is the message id for the last message sent for this session. This method is an accessor to the message id information stored within the session structure.
Return Value:
char * to a zero-terminated string that represents the message id for the last message sent on this session.
char * getMessageId(OXLO_SESSION * session);

13.3.11 getTimeout

The getTimeout function is an accessor to the timeout value in seconds for a correlated send. This is only valid after a sendCorrelatedMessage or a SendFile/SendBuffer.
Return Value:
int – timeout to wait for the corresponding response message in seconds

- `int getTimeout(OXLO_SESSION * session);`

13.3.12 getPollingInterval

The getPollingInterval function is an accessor to the polling interval in seconds for a correlated send. This is only valid after a sendCorrelatedMessage or a SendFile/SendBuffer.
Return Value:
int – polling interval to check for corresponding response message in seconds.

- `int getPollingInterval(OXLO_SESSION * session);`

13.3.13 getExpectedMessageCount

The getExpectedMessageCount function is an accessor to the number of expected messages to wait for in relation to a correlated send. This is only valid after a sendCorrelatedMessage or a SendFile/SendBuffer.
Return Value:
int – number of messages to look for in the inbox for the original message sent.

- `int getExpectedMessageCount(OXLO_SESSION * session);`

13.3.14 Utility Functions

13.3.15 sendAuditPing

Test connectivity between this client and the endpoint specified in the local profile. Calling this method starts a new session that attempts to connect to the Oxlo hosted service environment using HTTPS (not ICMP/Ping).
Return values:
0 = ONRAMP_SUCCESS = connectivity is good
1 = ONRAMP_GENERIC_ERROR = Generic error
2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
3 = ONRAMP_AUTH_FAILED = User authentication failed on server
4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
5 = ONRAMP_CONNECT_FAILED = Could not connect to server
6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
   7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
   8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
   9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
 -20 = ONRAMP_REQ_REJECTED = request was rejected
 -21 = ONRAMP_REQ_DEFERRED = request was deferred
 -90 = ONRAMP_UNABLE_TO_FIND_OCTD = unable to connect to octd
 -98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
 -99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid

int sendAuditPing(OXLO_SESSION * session)
```

13.4 liboxloVersion

Get the version of this communication client. This allows the user of this dll to determine the client version and therefore the set of methods that are avail.
Return value:
version number (132 for version 1.32)

```
int liboxloVersion(void)
```

13.5 onrampdVersion

Get the version of the onrampd service/daemon. This allows the user of this dll to determine the onrampd version.
Return value:
version number (134 for version 1.34)

```
   0 = ONRAMP_SUCCESS = connectivity is good
   1 = ONRAMP_GENERIC_ERROR = Generic error
   2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
   3 = ONRAMP_AUTH_FAILED = User authentication failed on server
   4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
   5 = ONRAMP_CONNECT_FAILED = Could not connect to server
   6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
   7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
   8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
   9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
 -20 = ONRAMP_REQ_REJECTED = request was rejected
 -21 = ONRAMP_REQ_DEFERRED = request was deferred
 -90 = ONRAMP_UNABLE_TO_FIND_ONRAMPD = unable to connect to onrampd
-100 = ONRAMP_LICENSE_INVALID = license invalid

int onrampdVersion(void)
```

13.5.1 Administrative Functions

13.6 setSenderId (session, inSessionId)

The setSenderId function allows the user to override the default senderId that is loaded from the configuration file. This allows the user in a hosted model to specify who the sender of this file/BOD.
Return value: void

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
void setSenderId(OXLO_SESSION * session, char *inSenderId);
```

13.7 setApplicationId(session, inAppId)

The setApplicationId function allows the user to override the default ApplicationId that is loaded from the configuration file.
Return value: void

```
void setApplicationId(OXLO_SESSION * inSession, char *inAppId);
```

13.8 setDocVersion (session, inDocVersion)

The setDocVersion function allows the user to override the default docVersion that is loaded from the configuration file.
Return value: void

```
void setDocVersion(OXLO_SESSION * session, float inDocVersion);
```

13.8.1 Message Sending Functions

13.8.1.1 sendFile (session, file name, destId, docType)

The sendFile function allows the user of this dll to send a file to the Oxlo hosted service environment. There are two variants of this function, one of which takes an extra parameter to signify which document version file represents. If the parameter is not specified, the document version delivered to the Oxlo hosted service environment is 1.0. This will aid tracking changes in document format. This is used for transformation and mapping purposes.
Return value:

```
  0 = ONRAMP_SUCCESS = connectivity is good
  1 = ONRAMP_GENERIC_ERROR = Generic error
  2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
  3 = ONRAMP_AUTH_FAILED = User authentication failed on server
  4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
  5 = ONRAMP_CONNECT_FAILED = Could not connect to server
  6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
  7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
  8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
  9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
-20 = ONRAMP_REQ_REJECTED = request was rejected
-21 = ONRAMP_REQ_DEFERRED = request was deferred
-22 = ONRAMP_CANNOT_OPEN_FILE = could not open file
-23 = ONRAMP_CANNOT_ATTACH_BODY = could not attach file as body
-30 = ONRAMP_ERR_HASH_FAILED = could not generate hash value of file
-90 = ONRAMP_UNABLE_TO_FIND_OCTD = unable to connect to octd
-98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
-99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid
```

- ```
  int sendFile(OXLO_SESSION * session, char *fname, char *destId,
          char *docType);
  ```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.
  ```

13.8.1.2 sendBuffer (session, data, data length, destId, docType)

The sendBuffer function allows the user of this dll to send a document from a memory buffer. There are two variants of this function, one of which takes an extra parameter to signify which document version file represents. If the parameter is not specified, the document version delivered to the Oxlo hosted service environment is 1.0. This will aid tracking changes in document format. This is used for transformation and mapping purposes.

Return value:

0 = ONRAMP_SUCCESS = connectivity is good
1 = ONRAMP_GENERIC_ERROR = Generic error
2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
3 = ONRAMP_AUTH_FAILED = User authentication failed on server
4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
5 = ONRAMP_CONNECT_FAILED = Could not connect to server
6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
-20 = ONRAMP_REQ_REJECTED = request was rejected
-21 = ONRAMP_REQ_DEFERRED = request was deferred
-24 = ONRAMP_CANNOT_ATTACH_BUFFER = could not attach buffer as body
-30 = ONRAMP_ERR_HASH_FAILED = could not generate hash value of file
-90 = ONRAMP_UNABLE_TO_FIND_OCTD = unable to connect to octd
-98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
-99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid

- int sendBuffer(OXLO_SESSION * session, const unsigned char *data,
  const long dataLen, char *destId, char *docType);

13.8.1.3 sendCorrelatedMessageFromFile

The sendCorrelatedMessageFromFile function sends a message to Oxlo Systems, retrieves polling information (timeout, polling interval, number of expected messages, etc), and starts polling the inbox to retrieve the correlated response message.

Return value:

0 = ONRAMP_SUCCESS = connectivity is good
1 = ONRAMP_GENERIC_ERROR = Generic error
2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
3 = ONRAMP_AUTH_FAILED = User authentication failed on server
4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
5 = ONRAMP_CONNECT_FAILED = Could not connect to server
6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
-20 = ONRAMP_REQ_REJECTED = request was rejected
-21 = ONRAMP_REQ_DEFERRED = request was deferred
-22 = ONRAMP_CANNOT_OPEN_FILE = could not open file
-23 = ONRAMP_CANNOT_ATTACH_BODY = could not attach file as body
-24 = ONRAMP_CANNOT_ATTACH_BUFFER = could not attach buffer as body
-30 = ONRAMP_ERR_HASH_FAILED = could not generate hash value of file
-31 = ONRAMP_NO_CORRELATION_MSG_ID = the correlation messageid is incorrect
-32 = ONRAMP_CORRELATION_TIMEOUT = could not retrieve the required number of messages within the timeout period.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

-90 = ONRAMP_UNABLE_TO_FIND_ONRAMPD = unable to connect to onrampd
-91 = ONRAMP_UNABLE_TO_AUTH_ONRAMPD = unable to authenticate with onrampd
-92 = ONRAMP_IPC_SEND_ERROR = unable to send request to onrampd
-93 = ONRAMP_IPC_RCV_ERROR = unable to process response from onrampd
-94 = ONRAMP_NOT_ENOUGH_INFO_TO_PROCESS = unable to process request not enough info to complete processing
-95 = ONRAMP_INVALID_SESSION
-98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
-99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid

- `int sendCorrelatedMessageFromFile(OXLO_SESSION * session, int numRespSessions, char *fname, char *destId, char *docType);`

13.8.1.4 sendCorrelatedMessageFromBuffer

The sendCorrelatedMessageFromBuffer function sends a message to Oxlo Systems, retrieves polling information (timeout, polling interval, number of expected messages, etc), and starts polling the inbox to retrieve the correlated response message.
Return value:
0 = ONRAMP_SUCCESS = connectivity is good
1 = ONRAMP_GENERIC_ERROR = Generic error
2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
3 = ONRAMP_AUTH_FAILED = User authentication failed on server
4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
5 = ONRAMP_CONNECT_FAILED = Could not connect to server
6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
-20 = ONRAMP_REQ_REJECTED = request was rejected
-21 = ONRAMP_REQ_DEFERRED = request was deferred
-24 = ONRAMP_CANNOT_ATTACH_BUFFER = could not attach buffer as body
-30 = ONRAMP_ERR_HASH_FAILED = could not generate hash value of file
-31 = ONRAMP_NO_CORRELATION_MSG_ID = the correlation messageid is incorrect
-32 = ONRAMP_CORRELATION_TIMEOUT = could not retrieve the required number of messages within the timeout period.
-90 = ONRAMP_UNABLE_TO_FIND_ONRAMPD = unable to connect to onrampd
-91 = ONRAMP_UNABLE_TO_AUTH_ONRAMPD = unable to authenticate with onrampd
-92 = ONRAMP_IPC_SEND_ERROR = unable to send request to onrampd
-93 = ONRAMP_IPC_RCV_ERROR = unable to process response from onrampd
-94 = ONRAMP_NOT_ENOUGH_INFO_TO_PROCESS = unable to process request not enough info to complete processing
-95 = ONRAMP_INVALID_SESSION
-98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
-99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid

- `int sendCorrelatedMessageFromBuffer(OXLO_SESSION * session, int numRespSessions, const unsigned char *data, const long dataLen, char *destId, char *docType);`

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

13.8.1.5 getMessageIds (session)

The getMessageIds retrieves a list of message ids of messages that are waiting in the Inbox. The message id's are returned in the body of the http response and are listed one per line. This is accessible using the getResponseData(session) method and converting each messages id from text (a string) to an unsigned long.
Return value:

0 = ONRAMP_SUCCESS = connectivity is good
1 = ONRAMP_GENERIC_ERROR = Generic error
2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
3 = ONRAMP_AUTH_FAILED = User authentication failed on server
4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
5 = ONRAMP_CONNECT_FAILED = Could not connect to server
6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
-20 = ONRAMP_REQ_REJECTED = request was rejected
-21 = ONRAMP_REQ_DEFERRED = request was deferred
-90 = ONRAMP_UNABLE_TO_FIND_OCTD = unable to connect to octd
-98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
-99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid

- int getMessageIds(OXLO_SESSION * session);

13.8.1.6 getMessageIdArray(session)

The getMessageIdArray function is a helper function for converting the message id list from the http body (text/string form) to an array of unsigned long integers for use in the getMessagexxx and deleteMessage functions.
Return value: unsigned long int ** getMessageIdArray; array of unsigned long integers.

- unsigned long ** getMessageIdArray(OXLO_SESSION * session)

13.8.1.7 getMessageAsBuffer(session, msgId)

The getMessageAsBuffer function is used to retrieve a message from the Inbox and place it into a buffer.
Return value:

0 = ONRAMP_SUCCESS = connectivity is good
1 = ONRAMP_GENERIC_ERROR = Generic error
2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
3 = ONRAMP_AUTH_FAILED = User authentication failed on server
4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
5 = ONRAMP_CONNECT_FAILED = Could not connect to server
6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
-20 = ONRAMP_REQ_REJECTED = request was rejected
-21 = ONRAMP_REQ_DEFERRED = request was deferred
-90 = ONRAMP_UNABLE_TO_FIND_OCTD = unable to connect to octd
-98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
-99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

- int getMessageAsBuffer(OXLO_SESSION * session,
  unsigned long msgId);

13.8.1.8 getMessageAsFile(session, msgId, filename)

The getMessageAsFile function is used to retrieve a message from the Inbox and write it to a file.
Return value:

```
   0 = ONRAMP_SUCCESS = connectivity is good
   1 = ONRAMP_GENERIC_ERROR = Generic error
   2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
   3 = ONRAMP_AUTH_FAILED = User authentication failed on server
   4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
   5 = ONRAMP_CONNECT_FAILED = Could not connect to server
   6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
   7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
   8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
   9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
 -20 = ONRAMP_REQ_REJECTED = request was rejected
 -21 = ONRAMP_REQ_DEFERRED = request was deferred
 -90 = ONRAMP_UNABLE_TO_FIND_OCTD = unable to connect to octd
 -98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
 -99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid
```

- int getMessageAsFile(OXLO_SESSION * session,
  unsigned long msgId, char * filename);

13.8.1.9 deleteMessage(session, msgId)

The deleteMessage function allows the user to delete a message from the Inbox.
Return value:

- int deleteMessage(OXLO_SESSION * session, unsigned long msgId);

```
   0 = ONRAMP_SUCCESS = connectivity is good
   1 = ONRAMP_GENERIC_ERROR = Generic error
   2 = ONRAMP_HOST_NOTFOUND = Server or proxy hostname lookup failed
   3 = ONRAMP_AUTH_FAILED = User authentication failed on server
   4 = ONRAMP_PROXY_AUTH_FAILED = User authentication failed on proxy
   5 = ONRAMP_CONNECT_FAILED = Could not connect to server
   6 = ONRAMP_CONNECT_TIMEOUT = Connection timed out
   7 = ONRAMP_PRECONDITION_FAILED = The precondition failed
   8 = ONRAMP_ASYNC_RETRY_REQUEST = Retry request
   9 = ONRAMP_REDIRECT_REQUESTED = Redirect request
 -20 = ONRAMP_REQ_REJECTED = request was rejected
 -21 = ONRAMP_REQ_DEFERRED = request was deferred
 -90 = ONRAMP_UNABLE_TO_FIND_OCTD = unable to connect to octd
 -98 = ONRAMP_CERTIFICATE_DECRYPT_FAILED = certificate decryption failed
 -99 = ONRAMP_CERTIFICATE_COULD_NOT_ATTACH = could not attach client certificate
-100 = ONRAMP_LICENSE_INVALID = license invalid
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

13.8.1.10 getRetrievedMessageType (session)

The getRetrievedMessageType function returns a pointer to a null-terminated string that is the type of message that was just retrieved from the inbox using this session. This can be used to determine how to parse or save the payload.

Return value:

pointer to a null-terminated string that describes the message type.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

14 Appendix A – Code Samples

The following samples are intended to provide best practice guidance for an Oxlo partner in developing an OnRamp™ communications module. Please be aware that each of these samples includes detailed comments to describe where the developer must implement their own application specific code, for example calling a common logging module. Additionally the outer logic of calling the communications code will likely vary across application modules. Is this an automated back end process (GM RIM), or an interactive user driven process (RouteOne) should be considered in planning specific error handling, calling retry logic, etc… Samples have been provided in C, C# and Visual Basic to provide structure and syntax in a variety of programming languages. If you require additional guidance in another language please contact your Oxlo Integration consultant for support. These examples are intended to provide a reusable, core structure that may be called from any application that leverages OnRamp. However, if you feel that these examples do not meet your integration specific needs please contact Oxlo to discuss other implementation options.

14.1 C Sample Code

```
/*
  * Copyright (c) 2004-2007 Oxlo Systems, Inc.. All Rights Reserved.
  *
  * OXLO SYSTEMS, INC. MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
  * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
  * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
  * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. OXLO SYSTEMS INC. SHALL NOT BE LIABLE FOR
  * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
  * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*/

#include "liboxlo.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#ifndef bool
  #define bool unsigned char
#endif

#ifndef false
  #define  false 0
  #define  true   !false
#endif

/* forward declarations */
int  SendMessage(char *SenderID, char *ApplicationID, float DocVersion,
            char *DocType, char *DestinationID, char *MessageFile);
int  SendCorrelatedMessage(char *SenderID, char *ApplicationID, float DocVersion,
```

```
char *DocType, char *DestinationID, char *MessageFile, char *ResponseFileName);
bool ProcessInBoxMessages(char *SenderID, char *ApplicationID);
void logResult(const char *inMessage, int inFunctionReturnCode, OXLO_SESSION *inSession);
bool RetrieveProcessAndDeleteMessage(unsigned long inMsgId, char * inSenderID, char * inApplicationID);
bool DeleteMessage(unsigned long inMsgId, char * delSenderID, char * delApplicationID);
bool ProcessMessage(const char *inMessageType, const char *inMessageContent, OXLO_SESSION *inSession);
void showUserMessage(OXLO_SESSION *inSession);

/* SendMessage send a message/file to Oxlo and returns an integer code.
     All return codes are defined in liboxlo.h
     The success code is: ONRAMP_SUCCESS=0
*/
int SendMessage(char *SenderID,
  char *ApplicationID,
  float DocVersion,
  char *DocType,
  char *DestinationID,
  char *MessageFile)
{
  // Create a session to use to send this message
  OXLO_SESSION session;
  int retCode = createSession(&session);
  if (retCode != ONRAMP_SUCCESS)
  {
    // log error message to stderr or to log file
    logResult("Unable to createSession", retCode, NULL);
    return retCode;
  }

  // Now set up session parameters
  setSenderId(&session, SenderID);
  setApplicationId(&session, ApplicationID);
  setDocVersion(&session, DocVersion);

  if (retCode != ONRAMP_SUCCESS)
  {
    // if the retCode is ONRAMP_REQ_REJECTED then the user
    // message (getUserMessage function) tells you more about
    // the error.
    if (retCode == ONRAMP_REQ_REJECTED)
    {
      // Call to user message error handling routine
      showUserMessage(&session);
      logResult("SendFile failed", retCode, &session);
      destroySession(&session);
      // return error status
      return retCode;
    } else {
```

```
        // log error message to stderr or to log file
        logResult("SendFile failed", retCode, &session);
        destroySession(&session);
        // return error status
        return retCode;
      }
    }

    // Now free up the memory used in this session
    destroySession(&session);

    // return good status
    // return ONRAMP_SUCCESS - retCode should equal ONRAMP_SUCCESS
    return retCode;
}


/* show the user an error message */
void showUserMessage(OXLO_SESSION *inSession)
{
    /* if this has an interactive user then show the message to the user
     * if this is an automated send then remove the printf line that follows
     * DSP needs to implement logic to determine if this should be logged to the user's
     * screen, or to a log file (automated batch process)
    */

    printf("The following error occured while attempting to send this message: %s\n",
          getUserMessage(inSession));
}

/* logResult - log a message and related result information */
void logResult(const char *inMessage, int inFunctionReturnCode, OXLO_SESSION *inSession)
{
    /* in this case I'll just write the message to stderr;
       you could write it to a log file.
    */
    if (inSession != NULL) {
       fprintf(stderr,
            "%s returned: %d with responseCode: %d responseReason: %s userMessage: %s responseData: %s\n",
            inMessage, inFunctionReturnCode,
            getResponseCode(inSession), getResponseReason(inSession),
            getUserMessage(inSession), getResponseData(inSession));
    } else {
       fprintf(stderr, "%s returned: %d\n", inMessage, inFunctionReturnCode);
    }
}

/* SendACorrelatedMessage sends a message to Oxlo and retrieves the corresponding response message from the inbox.
```

```
    All return codes are defined in liboxlo.h
    The success code is: ONRAMP_SUCCESS=0
    If the return code is ONRAMP_SUCCESS then the corresponding message from the inbox
    has been saved to disk as the filename given 'ResponseFileName'

*/
int SendCorrelatedMessage(char *SenderID,
  char *ApplicationID,
  float DocVersion,
  char *DocType,
  char *DestinationID,
  char *MessageFile,
  char *ResponseFileName)
{
  // Create a session to use to send this message
  OXLO_SESSION session;
  int retCode = createSession(&session);
  if (retCode != ONRAMP_SUCCESS)
  {
    // log error message to stderr or to log file
    logResult("Unable to createSession", retCode, NULL);
    return retCode;
  }

  // Now set up session parameters
  setSenderId(&session, SenderID);
  setApplicationId(&session, ApplicationID);
  setDocVersion(&session, DocVersion);

  // Now send the message
  // the 2nd parameter 1 - indicates that 1 response is expected
  retCode = sendCorrelatedMessageFromFile(&session, 1, MessageFile, DestinationID, DocType);
  if (retCode != ONRAMP_SUCCESS)
  {
    // if the retCode is ONRAMP_REQ_REJECTED then the user
    // message (getUserMessage function) tells you more about
    // the error.
    if (retCode == ONRAMP_REQ_REJECTED)
    {
      // Call to user message error handling routine
      showUserMessage(&session);
      logResult("SendFile failed", retCode, &session);
      destroySession(&session);
      // return error status
      return retCode;
    } else {
      // log error message to stderr or to log file
      logResult("SendFile failed", retCode, &session);
```

```
        destroySession(&session);
        // return error status
        return retCode;
      }
    }

    // now save the response data to 'ResponseFileName'
    // if saveSessionResponseDataToFile fails we clean up the session and return that retCode.
    retCode = saveSessionResponseDataToFile(&session, ResponseFileName);

    // Now free up the memory used in this session
    destroySession(&session);

    // return good status
    // return ONRAMP_SUCCESS - retCode should equal ONRAMP_SUCCESS
    return retCode;
}

/* ProcessInBoxMessages
     - retrieves messageid's for messages waiting in the inbox to be processed,
       then retrieves the message and deletes it.
     - end users MUST fill in the 'ProcessMessage' function below
*/
bool ProcessInBoxMessages(char *SenderID, char *ApplicationID)
{
    bool bResult = true;
    unsigned long msgId = 0;

    // Create a session to use to send this message
    OXLO_SESSION msgIdsSession;
    int retCode = createSession(&msgIdsSession);
    if (retCode != ONRAMP_SUCCESS)
    {
      // log error message to stderr or to log file
      logResult("Unable to createSession in ProcessInBoxMessages", retCode, NULL);
      return false;
    }

    // Now set up session parameters
    setSenderId(&msgIdsSession, SenderID);
    setApplicationId(&msgIdsSession, ApplicationID);

    // Now retrieve the message ids
    retCode = getMessageIds(&msgIdsSession);
    if (retCode == ONRAMP_SUCCESS)
    {
      char * msgIdsPayload = (char *)getResponseData(&msgIdsSession);
      char * msgIdPtr = strtok(msgIdsPayload, "\r\n");
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
    while (msgIdPtr) {
      /* convert string to unsigned long msgId - if atol fails it returns 0 */
      msgId = (unsigned long)atol(msgIdPtr);
      if (msgId != 0) {
        bResult &= RetrieveProcessAndDeleteMessage(msgId,SenderID,ApplicationID);
      }
      msgIdPtr = strtok(NULL, "\r\n");
    }
  } else {
    // log error message to stderr or to log file
    logResult("getMessageIds failed", retCode, &msgIdsSession);
    bResult = false;
  }

  // Now free up the memory used in this session
  destroySession(&msgIdsSession);

  // return the status
  return bResult;
}


/* RetrieveProcessAndDeleteMessage - retrieves and processes the message specified by inMsgId
*/
bool RetrieveProcessAndDeleteMessage(unsigned long inMsgId, char * inSenderID, char* inApplicationID)
{
  OXLO_SESSION   retrieveMsgSession;
  /* create a session - if it fails then return false */
  int retCode = createSession(&retrieveMsgSession);
  if (retCode != ONRAMP_SUCCESS) {
    // log error message to stderr or to log file
    logResult("Unable to createSession in RetrieveProcessAndDeleteMessage", retCode, NULL);
    return false;
  }

    // Now set up session parameters
  setSenderId(&retrieveMsgSession, inSenderID);
  setApplicationId(&retrieveMsgSession, inApplicationID);

  /* get message and process it */
  retCode = getMessageAsBuffer(&retrieveMsgSession, inMsgId);
  if (retCode != ONRAMP_SUCCESS) {
    // log error message to stderr or to log file
    logResult("getMessageAsBuffer failed", retCode, &retrieveMsgSession);
    destroySession(&retrieveMsgSession);
    return false;
  }
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
  /* get content of the message that was retrieved */
  /* do not free messageContent - this memory belongs to the retrieveMsgSession struct */
  const char * messageContent = getResponseData(&retrieveMsgSession);
  /* get a character string that describes the type of message just retrieved from the inbox */
  const char * messageType = getRetrievedMessageType(&retrieveMsgSession);

  /* attempt to process this message - this function calls CUSTOMER SPECIFIC code */
  if (ProcessMessage(messageType, messageContent, &retrieveMsgSession) != true) {
    // log error message to stderr or to log file
    logResult("ProcessMessage failed", 0, NULL);
    destroySession(&retrieveMsgSession);
    return false;
  }
  /* attempt to delete the message from the InBox
       - log and escalate if it fails, this would mean that the message was
         processed by the customer code but we were unable to delete it from
         the InBox and therefore next time around we will retrieve and process
         the same message again.
  */
  if (DeleteMessage(inMsgId,inSenderID,inApplicationID) != true) {
    // log error message to stderr or to log file
    logResult("DeleteMessage failed", retCode, NULL);
    destroySession(&retrieveMsgSession);
    return false;
  }

  /* clean up retrieveMsgSession session */
  destroySession(&retrieveMsgSession);
  return true;
}

/* DeleteMessage - deletes the specific message from the InBox */
bool DeleteMessage(unsigned long inMsgId, char* delSenderID, char* delApplicationID)
{
  OXLO_SESSION   deleteMsgSession;
  /* create a session - if it fails then return false */
  int retCode = createSession(&deleteMsgSession);
  if (retCode != ONRAMP_SUCCESS) {
    // log error message to stderr or to log file
    logResult("Unable to createSession in DeleteMessage", retCode, NULL);
    return false;
  }

      // Now set up session parameters
  setSenderId(&deleteMsgSession, delSenderID);
  setApplicationId(&deleteMsgSession, delApplicationID);

  /* delete this message from the InBox */
```

```
  retCode = deleteMessage(&deleteMsgSession, inMsgId);
  if (retCode != ONRAMP_SUCCESS) {
     // log error message to stderr or to log file
     logResult("deleteMessage failed", retCode, &deleteMsgSession);
     destroySession(&deleteMsgSession);
     return false;
  }

  /* clean up retrieveMsgSession session */
  destroySession(&deleteMsgSession);
  return true;
}

/********** TODO *****************
THE CUSTOMER MUST FILL IN THE APPROPRIATE MESSAGE PROCESSING LOGIC HERE!
*/
bool ProcessMessage(const char *inMessageType, const char *inMessageContent, OXLO_SESSION *inSession)
{
  /* TODO: the customer MUST fill in this function.
  *     Do not free inMessageType or inMessageContent.
  *     return true if the message was processed ok.
  *     return false if the message was not processed properly.
  */

 /* example of saving results to different filenames based on inMessageType
  *
  if (strcmp(inMessageType, "ProcessManagedPartsOrder") == 0) {
     retCode = saveSessionResponseDataToFile(inSession, "xxx.PMPO");
  } else if (strcmp(inMessageType, "PrintFinancialStatement") == 0) {
              retCode = saveSessionResponseDataToFile(inSession, "xxx.PFS.PDF");
  }
  */

 int retCode = saveSessionResponseDataToFile(inSession, "xxx.xml");

 if (retCode == ONRAMP_SUCCESS) {
          return true;
 } else {
          return false;
 }

}
```

14.2 C# Sample Code (OCX Implemetation)

```
/*
   * Copyright (c) 2004-2007 Oxlo Systems, Inc.. All Rights Reserved.
   *
```

```
  * OXLO SYSTEMS, INC. MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
  * THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
  * TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
  * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. OXLO SYSTEMS INC. SHALL NOT BE LIABLE FOR
  * ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
  * DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.
*/


using System;
using System.Diagnostics;
//Add message box namespace here
//using System.Windows
using OCT_ACTIVEXLib;

namespace OnRamp
{
   public class OnRamp
   {
      private OCT_ActiveX onRampActiveX;

      public const int ONRAMP_SUCCESS = 0;
      public const int ONRAMP_REQ_REJECTED = -20;
      private const char NEWLINE = '\n';

      public OnRamp()
      {
         onRampActiveX = new OCT_ActiveX();
      }

      /* USER MUST FILL IN THESE METHODS FOR THIS TO WORK */

      private void logMessage(string message)
      {
         // This method needs to be implemented
         //  Send the message to your own logging mechanism
         Trace.WriteLine(message);
      }

      private bool processMessage(string messageType, string messageContent, ref OCT_ActiveX inOnRampActiveX)
      {
         // This method needs to be implemented
         /* TODO: the customer MUST fill in this function.
               *     return true if the message was processed ok.
            *     return false if the message was not processed properly.
              */
```

```
    // example - select a filename based on messageType
    /*int retCode;
        if (messageType == "ProcessManagedPartsOrder")
        {
          retCode = inOnRampActiveX.saveSessionResponseDataToFile("xxx.pmpo.xml");
        } else if (messageType == "PrintFinacialStatement")
        {
          retCode = inOnRampActiveX.saveSessionResponseDataToFile("xxx.pdf");
        }
        */

    //example - write everything to a file
    int retCode = inOnRampActiveX.saveSessionResponseDataToFile(messageType + "somethingUnique.xml");
    if (retCode != ONRAMP_SUCCESS) { return false;  }

    return true;
}

private  void handleUserMessage(string userMessage)
{
    bool interactiveUser = true;
    // This method needs to be implemented
    if (interactiveUser)
    {
       //MessageBox.Show(userMessage);
    }
    else
    {
       logMessage(userMessage);
    }
}

/* END FILL IN THESE METHODS */

public int SendMessage(string SenderID, string ApplicationID, float DocVersion, string DocType, string DestinationID, string MessageFilePath)
{
    int retCode = 0;

    try
    {
       retCode = onRampActiveX.createSession();
       if (retCode == ONRAMP_SUCCESS)
       {
          // Now set up session parameters
          onRampActiveX.setSenderId(SenderID);
```

```
            onRampActiveX.setApplicationId(ApplicationID);
            onRampActiveX.setDocVersion(DocVersion);

            // Now send the message
            retCode = onRampActiveX.sendFile(MessageFilePath, DestinationID, DocType);
            if (retCode != ONRAMP_SUCCESS)
            {
               // if the retCode is ONRAMP_REQ_REJECTED then the user
               // message (getUserMessage function) tells you more about
               // the error.
               if (retCode == ONRAMP_REQ_REJECTED)
               {
                  /* if this has an interactive user then show the message to the user
                              * DSP needs to implement logic to determine if this should be logged to the user's
                              * screen, or to a log file (automated batch process)
                              */
                  string UserMsg = onRampActiveX.getUserMessage();
                  handleUserMessage(UserMsg);
               }
               else
               {
                  // log error message to stderr or to log file
                  logMessage("SendFile failed and returned: " + retCode);
               }
            }
         }
         else
         {
            //log error to a file or standard error
            logMessage("Error Creating Session: " + retCode);
         }
      }
      catch (Exception e)
      {
         //log error to standard error, log file, or anywhere else
         logMessage("SendFile Threw an Exception: " + e.Message);
      }
      finally
      {
         onRampActiveX.destroySession();
      }

      // return status
      return retCode;

   }
```

```
public int SendCorrelatedMessage(string SenderID, string ApplicationID, float DocVersion, string DocType, string DestinationID,
   string MessageFilePath, string ResponseFilePath)
{
   int retCode = 0;

   try
   {
      retCode = onRampActiveX.createSession();

      if (retCode == ONRAMP_SUCCESS)
      {
         // Now set up session parameters
         onRampActiveX.setSenderId(SenderID);
         onRampActiveX.setApplicationId(ApplicationID);
         onRampActiveX.setDocVersion(DocVersion);

         // Now send the message
         retCode = onRampActiveX.sendCorrelatedMessageFromFile(MessageFilePath, DestinationID, DocType);
         if (retCode != ONRAMP_SUCCESS)
         {
            // if the retCode is ONRAMP_REQ_REJECTED then the user
            // message (getUserMessage function) tells you more about
            // the error.
            if (retCode == ONRAMP_REQ_REJECTED)
            {
               // Call to user message error handling routine
               string UserMsg = onRampActiveX.getUserMessage();
               handleUserMessage(UserMsg);
            }
            else
            {
               // log error message to stderr or to log file
               logMessage("SendFile failed and returned: " + retCode);
            }
         }
         else
         {  // retCode was equal to ONRAMP_SUCCESS
            // now save the response data to the ResponseFilePath
            retCode = onRampActiveX.saveSessionResponseDataToFile(ResponseFilePath);
            if (retCode != ONRAMP_SUCCESS)
            {
               //Replace this with a call to your logging
               logMessage("saveSessionResponseDataToFile failed with return code: " + retCode);
            }
         }
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
        }
        else
        {
            //log error to a file or standard error
            logMessage("Error Creating Session: " + retCode);
        }
    }
    catch (Exception e)
    {
        //log error to standard error, log file, or anywhere else
        logMessage("SendFile Threw an Exception: " + e.Message);
    }
    finally
    {
        onRampActiveX.destroySession();
    }

    // return status
    return retCode;

}

public bool ProcessInBoxMessages(string SenderID, string ApplicationID)
{
    int retCode = 0;
    bool result = true;

    try
    {
        //Create a session
        retCode = onRampActiveX.createSession();

        if (retCode == ONRAMP_SUCCESS)
        {

            // Now set up session parameters
            onRampActiveX.setSenderId(SenderID);
            onRampActiveX.setApplicationId(ApplicationID);

            //get the messagesIDs
            retCode = onRampActiveX.getMessageIds();
            if (retCode == ONRAMP_SUCCESS)
            {
                string data = onRampActiveX.getResponseData();
                //message Id's come back in a space delimited format
                string[] messageIds = data.Split(NEWLINE);
```

```
            //walk through each message in the inbox
            for (int i = 0; i < messageIds.Length; i++)
            {
               //convert the message ID from a string to an int
               int msgId = int.Parse(messageIds[i].Trim());
               //call getMessageAsFile, can also use getMessageAsBuffer here
               retCode = onRampActiveX.getMessageAsBuffer(msgId);
               if (retCode != ONRAMP_SUCCESS)
               {
                  logMessage("Error getting message from Inbox: ID=" + msgId + " returnCode =" + retCode);
                  result = false;
               }
               else
               {

                  if (processMessage(onRampActiveX.getRetrievedMessageType(),onRampActiveX.getResponseData(),ref onRampActiveX) == true)
                  {
                     //Delete the message from the inbox
                     retCode = onRampActiveX.deleteMessage(msgId);
                     if (retCode != ONRAMP_SUCCESS)
                     {
                        logMessage("Error Deleting Message from Inbox: ID=" + msgId + " returnCode =" + retCode);
                        result = false;
                     }
                  }
                  else
                  {
                     result = false;
                     logMessage("Error in processMessage");
                  }
               }
            }
         }
         else
         {
            logMessage("Error Getting MessageID's: " + retCode);
            return false;
         }
      }
      else
      {    //log error to a file or standard error
         logMessage("Error Creating Session: " + retCode);
         return false;
      }
   }
```

```
        catch (Exception e)
        {
            logMessage("SendFile Threw an Exception: " + e.Message);
            return false;
        }
        finally
        {
            //delete the session
            onRampActiveX.destroySession();
        }

        // return status
        return result;
    }

  }
}
```

14.3 Visual Basic 6 Sample Code (OCX Implementation)

```
' Copyright (c) 2004-2007 Oxlo Systems, Inc.. All Rights Reserved.
'
' OXLO SYSTEMS, INC. MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
' THE SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
' TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
' PARTICULAR PURPOSE, OR NON-INFRINGEMENT. OXLO SYSTEMS INC. SHALL NOT BE LIABLE FOR
' ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR
' DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES.


Const ONRAMP_SUCCESS As Integer = 0
Const ONRAMP_REQ_REJECTED As Integer = -20
Const NEWLINE As String = "\n"


Public Function SendMessage(ByVal SenderID As String, ByVal ApplicationID As String, ByVal DocVersion As Single, ByVal DocType As String, ByVal
DestinationID As String, ByVal MessageFilePath As String) As Integer
    Dim retCode As Integer
    Dim sendMessageOnRampActiveX As New OCT_ACTIVEXLib.OCT_ActiveX

    retCode = sendMessageOnRampActiveX.createSession()
    If retCode <> ONRAMP_SUCCESS Then
       'log error to a file or standard error
       logMessage ("Error Creating Session: " + retCode)
       SendMessage = retCode
    End If

    'Now set up session parameters
    sendMessageOnRampActiveX.setSenderId (SenderID)
    sendMessageOnRampActiveX.setApplicationId (ApplicationID)
    sendMessageOnRampActiveX.setDocVersion (DocVersion)

    'Now send the message
    retCode = sendMessageOnRampActiveX.sendFile(MessageFilePath, DestinationID, DocType)
    If retCode <> ONRAMP_SUCCESS Then

       'if the retCode is ONRAMP_REQ_REJECTED then the user
       'message (getUserMessage function) tells you more about
       'the error.
       If retCode = ONRAMP_REQ_REJECTED Then

          'Call to user message error handling routine
```

```
            UserMsg = sendMessageOnRampActiveX.getUserMessage()
            HandleUserMessage (UserMsg)
            logMessage ("User Message Received: " + UserMsg)
         Else

            'log error message to stderr or to log file
            logMessage ("SendFile failed and returned: " + retCode)
         End If

      End If

      sendMessageOnRampActiveX.destroySession

      'return status
      SendMessage = retCode
   End Function

   Public Function SendCorrelatedMessage(ByVal SenderID As String, ByVal ApplicationID As String, ByVal DocVersion As Single, ByVal DocType As String,
ByVal DestinationID As String, ByVal MessageFilePath As String, ByVal ResponseFilePath As String) As Integer
      Dim sendCorrelatedMessageOnRampActiveX As New OCT_ACTIVEXLib.OCT_ActiveX
      Dim retCode As Integer
      retCode = sendCorrelatedMessageOnRampActiveX.createSession()
      If retCode <> ONRAMP_SUCCESS Then
         'log error to a file or standard error
         logMessage ("Error Creating Session in SendCorrelatedMessage: " + retCode)
         SendCorrelatedMessage = retCode
      End If

      'Now set up session parameters
      sendCorrelatedMessageOnRampActiveX.setSenderId (SenderID)
      sendCorrelatedMessageOnRampActiveX.setApplicationId (ApplicationID)
      sendCorrelatedMessageOnRampActiveX.setDocVersion (DocVersion)

      'Now send the message
      retCode = sendCorrelatedMessageOnRampActiveX.sendCorrelatedMessageFromFile(MessageFilePath, DestinationID, DocType)
      If retCode <> ONRAMP_SUCCESS Then
         'if the retCode is ONRAMP_REQ_REJECTED then the user
         'message (getUserMessage function) tells you more about
         'the error.
         If retCode = ONRAMP_REQ_REJECTED Then

            'Call to user message error handling routine
            UserMsg = sendCorrelatedMessageOnRampActiveX.getUserMessage()
            HandleUserMessage (UserMsg)
            logMessage ("User Message Received: " + UserMsg)
         Else
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
            'log error message to stderr or to log file
            logMessage ("SendFile failed and returned: " + retCode)
         End If
      Else
         'save response data to file 'ResponseFilePath'
         retCode = sendCorrelatedMessageOnRampActiveX.saveSessionResponseDataToFile(ResponseFilePath)
         If retCode <> ONRAMP_SUCCESS Then
            logMessage ("SaveSessionResponseDataToFile failed and returned: " + retCode)
         End If
      End If

      sendCorrelatedMessageOnRampActiveX.destroySession

      'return status
      SendCorrelatedMessage = retCode
   End Function

Public Function ProcessInBoxMessages(ByVal SenderID As String, ByVal ApplicationID As String, ByVal directoryPath As String) As Boolean
      Dim messageIdsActiveX As New OCT_ACTIVEXLib.OCT_ActiveX
      Dim retCode As Integer
      Dim bResult As Boolean
      bResult = True

      'Create a session for retrieving messageIds
      retCode = messageIdsActiveX.createSession()

      If retCode <> ONRAMP_SUCCESS Then
         'log error to a file or standard error
         Trace.WriteLine ("Error Creating Session: " + retCode)
         ProcessInBoxMessages = False
      End If

      ' Now set up session parameters
      messageIdsActiveX.setSenderId (SenderID)
      messageIdsActiveX.setApplicationId (ApplicationID)

      'get the messagesIDs
      retCode = messageIdsActiveX.getMessageIds
      If retCode = ONRAMP_SUCCESS Then
         Dim data As String
         data = messageIdsActiveX.getResponseData()
         'message Id's come back in NEWLINE delimited
         Dim messageIds() As String
         messageIds = Split(data, NEWLINE)
```

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

```
        'walk through each message in the inbox
        Dim i As Integer
        For i = LBound(messageIds) To UBound(messageIds) Step 1
            'convert the message ID from a string to an int
            Dim msgId As Long
            msgId = messageIds(i)
            bResult = bResult And RetrieveProcessAndDeleteMessage(msgId, SenderID, ApplicationID)
        Next
    Else
        logMessage ("getMessageIds failed with: " + retCode)
    End If

    'delete the session
    messageIdsActiveX.destroySession

    'return status
    ProcessInBoxMessages = bResult
End Function

Public Function HandleUserMessage(ByVal inMessage As String)
    ' if this is for an interactive user then display it in a message box
    ' else log it to a log file
    Dim isInteractiveUser As Boolean
    isInterativeUser = True
    If isInteractiveUser = True Then
        MsgBox (inMessage)
    Else
        logMessage (inMessage)
    End If
End Function

Public Function logMessage(ByVal inMessage As String)
    ' end user should push this message to THEIR logging system
    Trace.WriteLine (inMessage)
End Function

Private Function RetrieveProcessAndDeleteMessage(msgId As Long, inSenderID As String, inApplicationID As String) As Boolean
    Dim retCode As Integer
    Dim messageContentActiveX As New OCT_ACTIVEXLib.OCT_ActiveX

    Dim messageContent As String
    Dim contentLength As Long
    Dim messageType As String

    retCode = messageContentActiveX.createSession()
    If retCode <> ONRAMP_SUCCESS Then
```

```
        logMessage ("Unable to create session in RetrieveProcessAndDeleteMessage: " + retCode)
        RetrieveProcessAndDeleteMessage = False
    End If

    ' Now set up session parameters
    messageContentActiveX.setSenderId (inSenderID)
    messageContentActiveX.setApplicationId (inApplicationID)

    retCode = messageContentActiveX.getMessageAsBuffer(msgId)
    If retCode <> ONRAMP_SUCCESS Then
        logMessage ("Unable to getMessageAsBuffer in RetrieveProcessAndDeleteMessage: " + retCode)
        messageContentActiveX.destroySession
        RetrieveProcessAndDeleteMessage = False
    End If

    messageContent = messageContentActiveX.getResponseData()

    messageType = messageContentActiveX.getRetrievedMessageType()

    If ProcessMessage(messageType, messageContent, messageContentActiveX) <> True Then
        logMessage ("ProcessMessage Failed")
        messageContentActiveX.destorySession
        RetrieveProcessAndDeleteMessage = False
    End If

    If DeleteMessage(msgId, inSenderID, inApplicationID) <> True Then
        logMessage ("DeleteMessage failed")
        messageContentActiveX.destorySession
        RetrieveProcessAndDeleteMessage = False
    End If

    messageContentActiveX.destorySession
    RetrieveProcessAndDeleteMessage = True

End Function

Private Function DeleteMessage(inMsgId As Long, delSenderID As String, delApplicationID As String) As Boolean

    Dim retCode As Integer
    Dim deleteMessageActiveX As New OCT_ACTIVEXLib.OCT_ActiveX

    retCode = deleteMessageActiveX.createSession()
    If retCode <> ONRAMP_SUCCESS Then
        logMessage ("Unable to createSession in DeleteMessage: " + retCode)
        DeleteMessage = False
    End If
```

```
    ' Now set up session parameters
    deleteMessageActiveX.setSenderId (delSenderID)
    deleteMessageActiveX.setApplicationId (delApplicationID)

    retCode = deleteMessageActiveX.deleteMessage(inMsgId)
    If retCode <> ONRAMP_SUCCESS Then
       logMessage ("Unable to delete message in DeleteMessage: " + retCode)
       deleteMessageActiveX.destroySession
       DeleteMessage = False
    End If

    deleteMessageActiveX.destroySession
    DeleteMessage = True
End Function

Private Function ProcessMessage(inMessageType As String, inMessageContent As String, ByRef inActiveX As OCT_ACTIVEXLib.OCT_ActiveX) As Boolean
    'TODO: the customer MUST fill in this function.
    'return true if the message was processed ok.
    'return false if the message was not processed properly.

    'example of saving results to different filenames based on inMessageType
    'If inMessageType = "ProcessManagedPartsOrder" Then
    '    retCode = inActiveX.saveResponseDataAsFile("xxx.PMPO")
    'ElseIf inMessageType = "PrintFinancialStatement" Then
    '    retCode = inActiveX.saveResponseDataAsFile(inSession, "xxx.PFS.PDF")
    'End If

    Dim retCode As Integer


    retCode = inActiveX.saveResponseDataAsFile("xxx.xml")

    If retCode = ONRAMP_SUCCESS Then
       ProcessMessage = True
    Else
       ProcessMessage = False
    End If
End Function
```

Trademarks

Oxlo, Oxlo OnRamp are all trademarks of Oxlo Systems Inc.

All other trademarks are the property of their respective owners.

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.

Revision History

Version	Description	Author
20060915	Installation and Development Guide original	RRN
20060920	Merged version of installation and dev guides	CW
20061018	Removed duplicate content, added functional code samples	GG
20061020	Minor edits and format fixes	CW
20070227	Updated to include correlated messaging. Updated code samples.	GG

Confidential Information – Do not copy or distribute without written permission of Oxlo Systems Inc.