[CLIENT_LOGO]

[Application_Name(2)]

Installation, Configuration, & Deployment Guide for {Application(1)}® Automated Lending API ({Application(2)})

Version 1.0.0

December 2015

Information contained in this document is confidential and proprietary to {CLIENT_NAME} and is licensed, not sold to our customers. The information may be used only in accordance with the provisions of the applicable agreement between the parties for the software or service that is the subject of this document, and the license is contingent upon the associated rights to use the underlying software or service. In no event may this document or the information herein be used to copy any portion of the system described herein or to design another system to accomplish similar results.

This document is not intended as a substitute for formal education in the regulatory requirements of banking, banking operations, lending, lending operations, or other topics generally applicable to financial institutions. Your financial institution is solely responsible for configuring and using the software or service in a way that meets policies, practices, and laws applicable to your institution, including, without limitation: (1) Options and selections made on prompts; (2) entries in the software program; (3) program setup; and (4) documents produced by the software. Information in this document is subject to change without notice and does not represent a commitment on the part of {CLIENT_NAME}.

The information contained in this document and the general guidance of {CLIENT_NAME} staff does not take the place of qualified compliance personnel or legal counsel within your institution. {CLIENT_NAME} CANNOT RENDER LEGAL, ACCOUNTING OR PROFESSIONAL SERVICES TO YOUR INSTITUTION. THE INFORMATION CONTAINED HEREIN IS GENERAL IN NATURE AND DOES NOT CONSTITUTE LEGAL ADVICE OR A LEGAL OPINION. CONSULT YOUR LEGAL COUNSEL FOR LEGAL ADVICE SPECIFIC TO YOUR SITUATION OR CIRCUMSTANCES OR TO ANSWER ANY LEGAL QUESTIONS.

Contents

About This Guide

Purpose

The **{Application_Name(2)}** (**{Application(2)}**) is designed to support the integration of {CLIENT_NAME}'s {Application(1)}® product with other applications to enable automation of the document preparation task within an automated loan processing environment. The solution allows the loan origination, decisioning, loan closing, document preparation, and document distribution steps to be integrated into a system capable of end-to-end, straight-through processing.

The *Installation, Configuration, & Deployment Guide for {Application(1)}® Automated Lending API ({Application(2)})* contains basic information about, not surprisingly, how to install, configure, and deploy the interface with the {Application_Name(1)} running inside {Application(2)} (??).

NOTE: Although {Application(2)} is originally based on Microsoft's® Component Object Model (COM) and still supports programmatic access to {Application(1)} as COM objects and fields, it now also supports the same programmatic access using e**X**tensible **M**arkup **L**anguage (XML) for over 200 objects and over 12,000 fields. Additionally, the interface enables the user to enforce {Application(1)} business rules.

Overview

The ALS is implemented in the Version 2.0 {Application_Name(4)} executable file (`HFARSN02.EXE`) and is a Microsoft Windows™ socket listener service that achieves the following:

1. Accepts specific XML documents from the socket;
2. Routes the request to the {Application_Name(2)} ({Application(2)}) COM/COM+ objects; and
3. Returns the response from the {Application(2)}.

The ALS runs on Windows Server 2008 R2, 2012 R2, Windows 7 (x86 or x64), or Windows 8/10. The ALS accepts the following XML documents:

- `LPLTransaction (LPLTransaction.XSD)`
- `LPLExecute (LPLExecute.XSD)`
- `LPLGetAsXML (LPLGetAsXML.XSD and LPLGetAsXMLResponse.XSD)`
- `LPLAutomation (LPLAutomation.XSD)`
- `LPLCalcEngine (LPLCalcEngine.XSD)`

External applications communicate with the ALS using Windows sockets. The service listens for connections on a main thread: When it accepts a connection, ALS starts a process thread to handle the connection. The settings in the `[ALS]` section of the `CFIPRO.INI` file in the `\Windows` or `\WINNT` directory, which control the port the service listens through, the length of time it "waits" for data once a connection has been established, the priority the thread runs at, and the thread cache size.

The ALS parameters – which must include the term *[ALS]* with the parameter – and their defaults are as follows:

Parameter	Default
`Port=<value>`	*2025*
`TimeOut=<value in milliseconds>`	*30000 (30 seconds)*
`Priority=<Lowest|Lower|Normal|Higher|Highest>`	*Normal*
`ThreadCacheSize=<value>`[1]	*10*

[1] Refers to the number of threads that are cached for reuse.

For example:

`[Insert appropriate example here.]`

The ideal value of ThreadCacheSize depends on the number and frequency of client socket requests received by the server socket:

- If `ThreadCacheSize` is too low, the server socket spends more time freeing and creating threads when accepting client connections.
- If `ThreadCacheSize` is too high, the server socket may unnecessarily allocate memory for threads that are never reused.

When the ALS receives a document, it performs the following operations in a processing thread:

1. Verify that the XML Document Type is one of the supported types. If it is not, the server closes the socket connection and returns no error.

2. Validates the `<MainOfficeNumber>` element value.

3. Processes the XML Document based on its Document Type.

4. Sends the response to the client over the same socket connection, indicating success or failure.

5. "Waits" for another request or for the client to disconnect.

Response XML Documents

The response to the LPLTransaction, LPLExecute, and LPLAutomation Document Types is an HFSTransResponse XML document and is defined in the HFSTransResponse.XSD file.

The <ResponseCode> element contains a value of ***0001,* indicating success,** or ***0016,* indicating failure**. *0001* indicates that ALS processed the XML document successfully, without generating an error message (??), and a text message indicating the action performed is contained in the <ResponseMessage> element. Additionally, if the request was an LPLAutomation document and included an LPLGetAsXML document, the XML response is contained in the <Response> element. Failure indicates that ALS could not complete the request, with the reason(s) contained in the <ErrorSource> and <DiagnosticErrorMessage> or <ErrorStack> element values.

The LPLCalcEngine XML document returns the same Document Type that was submitted. The LPLCalcEngine document contains the results of the calculation. It might also include an ErrorMessage value indicating why ALS could not perform the calculation or a WarningMessage element in the LPLCalcEngineFields node indicating a warning such as the final payment being due on a Sunday or holiday.

The service returns the LPLGetAsXML XML document request as an LPLGetAsXMLResponse document (LPLGetAsXMLResponse.XSD).

Registering and Accessing the Service

The following are critical points for registering and accessing the {Application_Name(1)}/{Application_Name(4)}, known together as the “Service”:

- Perform registration of Microsoft’s® Component Object Model (COM), the object model documented in the `LPLTransaction.XSD` XML schemas, when first using the Service after having installed or updated it. This registration requires Administrator rights when you enable User Account Control (UAC), which helps prevent unauthorized changes to your computer.
- Use the configuration tool found at GSS/Tools/Advanced Diagnostics/Configuration Manager, then run `...\CFI\HALCMN01.EXE`.
- Make the CFI folder the current working folder when accessing the COM DLL files directly in code to ensure that you can resolve/reconcile (??) all other DLL dependencies.
- When accessing the COM DLL files (??) through the ALS, the caller/user is not aware of the presence or location of the {Application(1)} and its {Application(2)} components.

Installing and Deploying the {Application_Name(4)})

The following are critical points for consideration of installing and deploying the ARS:

- All ARS components are included with the {Application(1)} installation and updates.
- All license codes activate the ARS.
- If necessary, you can update or fix the ARS application separately from downloading any {Application(1)} updates.
- The executable file, `HFARSN02.EXE`, is located in the `SDR:\SPP\CFI` directory that is established during the initial installation. If distributed processing is in effect, you must manually copy the file to each distributed site via the `RDR\RPP\CFI` folders. (??)
  - `SDR`: Start Drive, `SPP`: Start Path Prefix
  - `RDR`: Run Drive, `RPP`: Run Path Prefix

To install the ARS, run HFARSN02.EXE/Install.

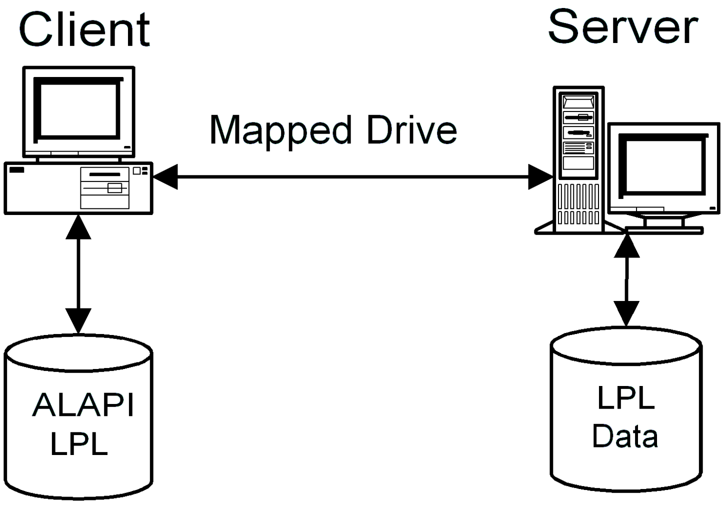


Figure 1: Deployment for Direct COM Access for a "Thick Client" (??)

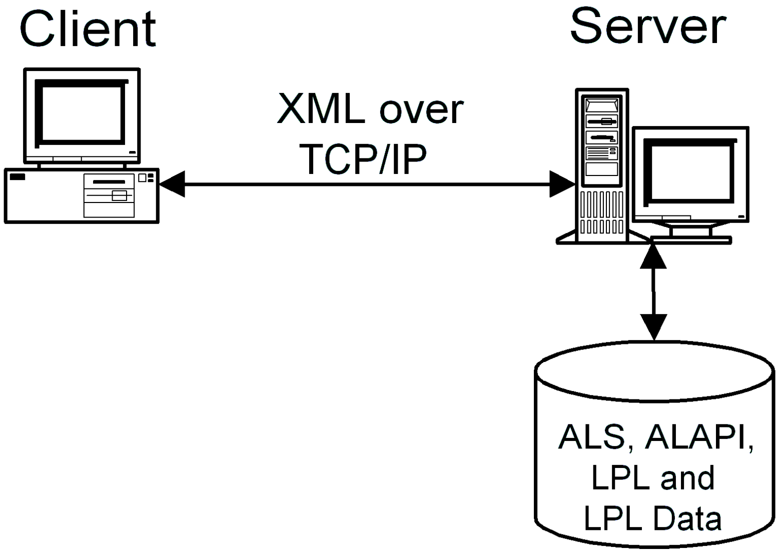


Figure 2: Deployment for ALS Access for "No Client" (??)

Uninstalling the ARS

To uninstall the ARS, run `HFARSN02.EXE/Uninstall`.

Configuring the Service

You can use the default login account of `LocalSystem` to process XML requests that do not require access to printer or network resources. However, if printer or network resources are required to process the XML request, you must use an account with the appropriate printer and network rights.

If an XML request includes one for printing, then the selected account must also include a printer set up in Windows. You must further select that printer in {Application(1)} Lending (LPL) on the server that the LPL service is running on. For more information on printer setup in LPL, consult your LPL documentation.

The login account (`LocalSystem` or another) requires the following privileges:

- Policy Rights: *Log on as a Service*
- Read privileges in the following directories: Windows System DLLs
- Read/Write privileges to the following directories: `CFI` Directory and subdirectories where General System Setup (GSS) and LPL are installed
- Read/Write privileges to this Registry key: `HKEY_CLASSES_ROOT`

To change the login account, do the following:

1. Use the Windows Service Control Manager to select the *HFS/{CLIENT_NAME} (??) Automated Lending Service*. The Windows Services dialog box displays.
2. In the dialog box, select the *Log On* tab.
3. Click the *This Account* radio button.
4. Enter the user ID and password for the user account under which the service is to run.

You can control other configuration parameters by including additional tags in the `CFIPRO.INI` file. Those tags are as follows:

- `OutOfProcess=<True|False>`: When set to *True*, LPL processes each thread in a separate instance of running `HAPSEN01.EXE` to eliminate memory management overhead when all threads are running in the process space of the service itself. If the service is handling more than five concurrent requests over an extended period of time, you must include this tag and set it to *True*.
- `StatusCheckInterval=<value in milliseconds>`: When you include this parameter, it overrides the default value of *120,000* (two minutes) that controls the interval during which LPL tests each thread’s status for idleness. If it is idle, LPL increments and tests the thread’s idle counter against the `FreeAfterIdleCount` tag value. If the counter value exceeds the `FreeAfterIdleCount` value, LPL terminates the thread and releases its objects.

NOTE: Please consult with {CLIENT_NAME} before overriding the default value.

- `FreeAfterIdleCount=<integer value>`: When you include this parameter, it overrides the default value of *3*. As described above, LPL uses the `FreeAfterIdleCount` value in conjunction with the `StatusCheckInterval` parameter.

NOTE: Please consult with {CLIENT_NAME} before overriding the default value.

Starting the {Application_Name(1)}

1. Use the Windows Service Control Manager to select the *HFS/{CLIENT_NAME} (??) Automated Lending Service*. The Windows Services dialog box displays.

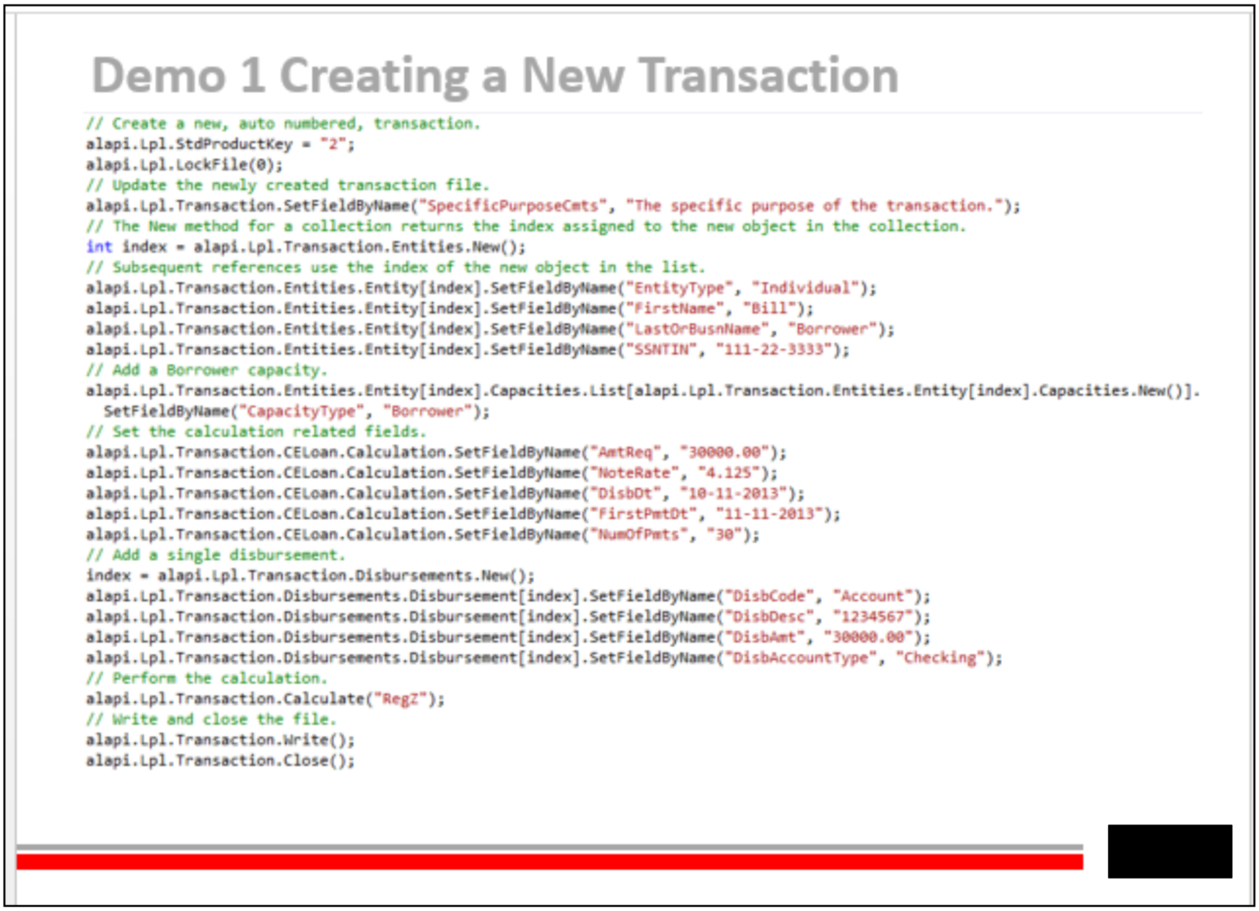
2. In the dialog box, click *Start*.

Stopping the ALS

1. Use the Windows Service Control Manager to select the *HFS/{CLIENT_NAME} (??) Automated Lending Service*. The Windows Services dialog box displays.

2. In the dialog box, click *Stop*.

Appendix B: Creating a Transaction

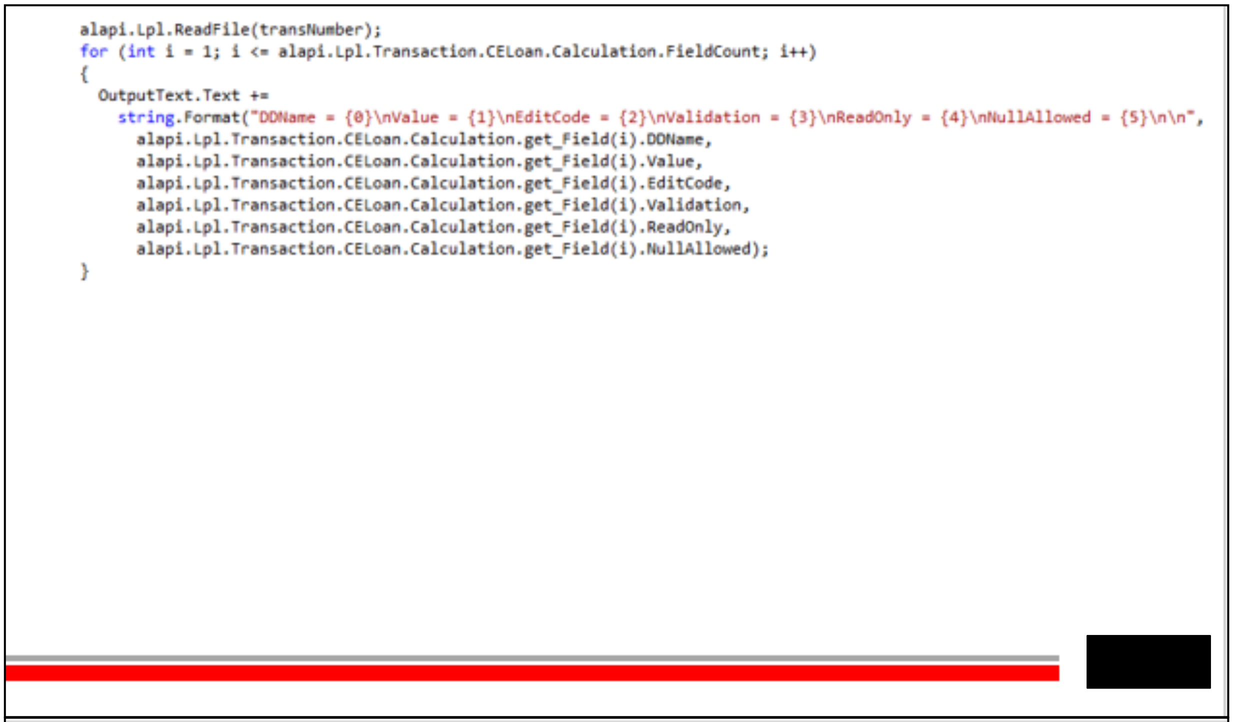


Demo 1 Creating a New Transaction

```
// Create a new, auto numbered, transaction.
alapi.Lpl.StdProductKey = "2";
alapi.Lpl.LockFile(0);
// Update the newly created transaction file.
alapi.Lpl.Transaction.SetFieldByName("SpecificPurposeCmts", "The specific purpose of the transaction.");
// The New method for a collection returns the index assigned to the new object in the collection.
int index = alapi.Lpl.Transaction.Entities.New();
// Subsequent references use the index of the new object in the list.
alapi.Lpl.Transaction.Entities.Entity[index].SetFieldByName("EntityType", "Individual");
alapi.Lpl.Transaction.Entities.Entity[index].SetFieldByName("FirstName", "Bill");
alapi.Lpl.Transaction.Entities.Entity[index].SetFieldByName("LastOrBusnName", "Borrower");
alapi.Lpl.Transaction.Entities.Entity[index].SetFieldByName("SSNTIN", "111-22-3333");
// Add a Borrower capacity.
alapi.Lpl.Transaction.Entities.Entity[index].Capacities.List[alapi.Lpl.Transaction.Entities.Entity[index].Capacities.New()].
  SetFieldByName("CapacityType", "Borrower");
// Set the calculation related fields.
alapi.Lpl.Transaction.CELoan.Calculation.SetFieldByName("AmtReq", "30000.00");
alapi.Lpl.Transaction.CELoan.Calculation.SetFieldByName("NoteRate", "4.125");
alapi.Lpl.Transaction.CELoan.Calculation.SetFieldByName("DisbDt", "10-11-2013");
alapi.Lpl.Transaction.CELoan.Calculation.SetFieldByName("FirstPmtDt", "11-11-2013");
alapi.Lpl.Transaction.CELoan.Calculation.SetFieldByName("NumOfPmts", "30");
// Add a single disbursement.
index = alapi.Lpl.Transaction.Disbursements.New();
alapi.Lpl.Transaction.Disbursements.Disbursement[index].SetFieldByName("DisbCode", "Account");
alapi.Lpl.Transaction.Disbursements.Disbursement[index].SetFieldByName("DisbDesc", "1234567");
alapi.Lpl.Transaction.Disbursements.Disbursement[index].SetFieldByName("DisbAmt", "30000.00");
alapi.Lpl.Transaction.Disbursements.Disbursement[index].SetFieldByName("DisbAccountType", "Checking");
// Perform the calculation.
alapi.Lpl.Transaction.Calculate("RegZ");
// Write and close the file.
alapi.Lpl.Transaction.Write();
alapi.Lpl.Transaction.Close();
```

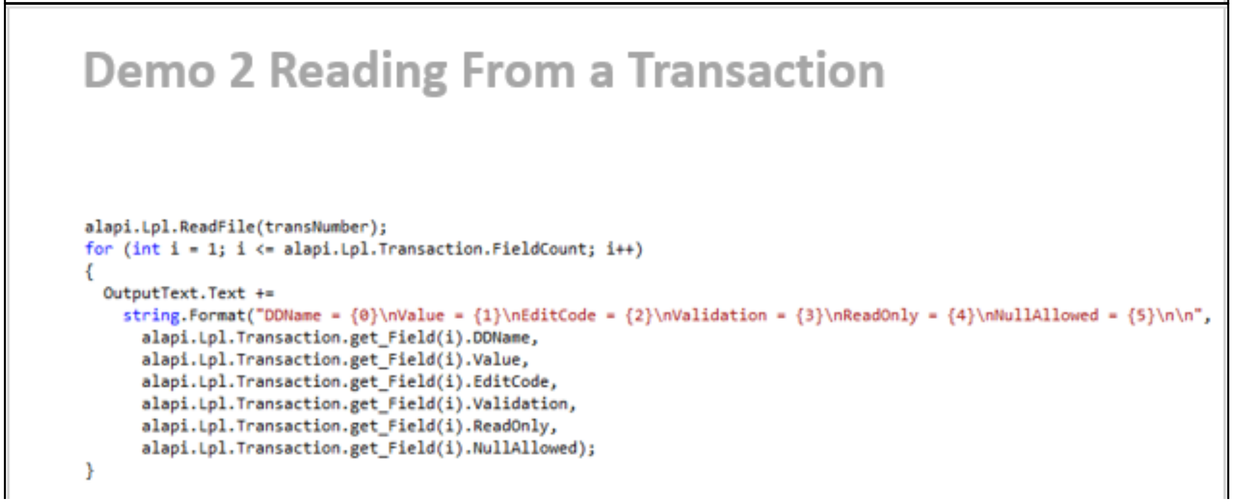
[redacted]

Appendix C: Reading from a Transaction



```
alapi.Lpl.ReadFile(transNumber);
for (int i = 1; i <= alapi.Lpl.Transaction.CELoan.Calculation.FieldCount; i++)
{
  OutputText.Text +=
    string.Format("DDName = {0}\nValue = {1}\nEditCode = {2}\nValidation = {3}\nReadOnly = {4}\nNullAllowed = {5}\n\n",
      alapi.Lpl.Transaction.CELoan.Calculation.get_Field(i).DDName,
      alapi.Lpl.Transaction.CELoan.Calculation.get_Field(i).Value,
      alapi.Lpl.Transaction.CELoan.Calculation.get_Field(i).EditCode,
      alapi.Lpl.Transaction.CELoan.Calculation.get_Field(i).Validation,
      alapi.Lpl.Transaction.CELoan.Calculation.get_Field(i).ReadOnly,
      alapi.Lpl.Transaction.CELoan.Calculation.get_Field(i).NullAllowed);
}
```

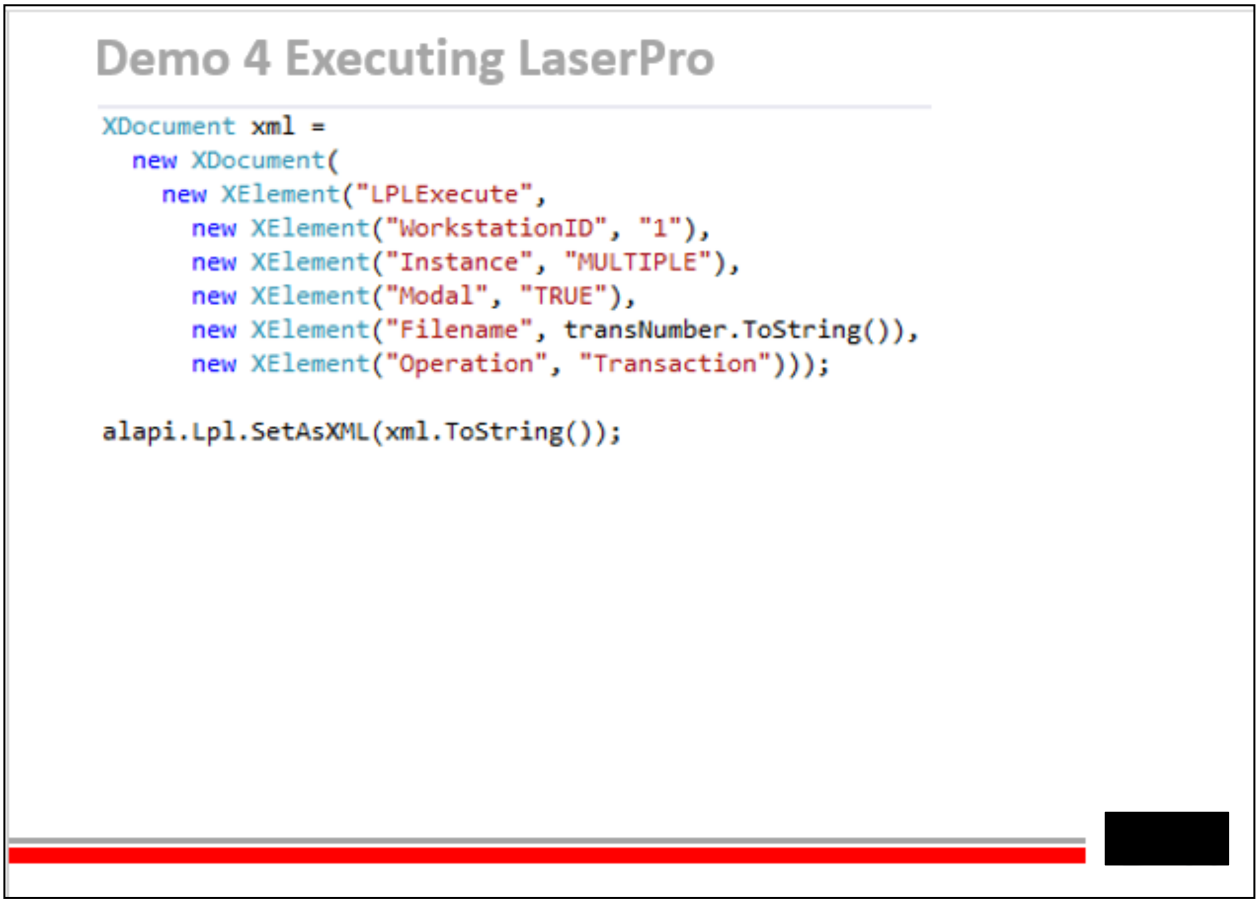
[redacted]



Demo 2 Reading From a Transaction

```
alapi.Lpl.ReadFile(transNumber);
for (int i = 1; i <= alapi.Lpl.Transaction.FieldCount; i++)
{
  OutputText.Text +=
    string.Format("DDName = {0}\nValue = {1}\nEditCode = {2}\nValidation = {3}\nReadOnly = {4}\nNullAllowed = {5}\n\n",
      alapi.Lpl.Transaction.get_Field(i).DDName,
      alapi.Lpl.Transaction.get_Field(i).Value,
      alapi.Lpl.Transaction.get_Field(i).EditCode,
      alapi.Lpl.Transaction.get_Field(i).Validation,
      alapi.Lpl.Transaction.get_Field(i).ReadOnly,
      alapi.Lpl.Transaction.get_Field(i).NullAllowed);
}
```

Appendix D: Executing {Application(1)}



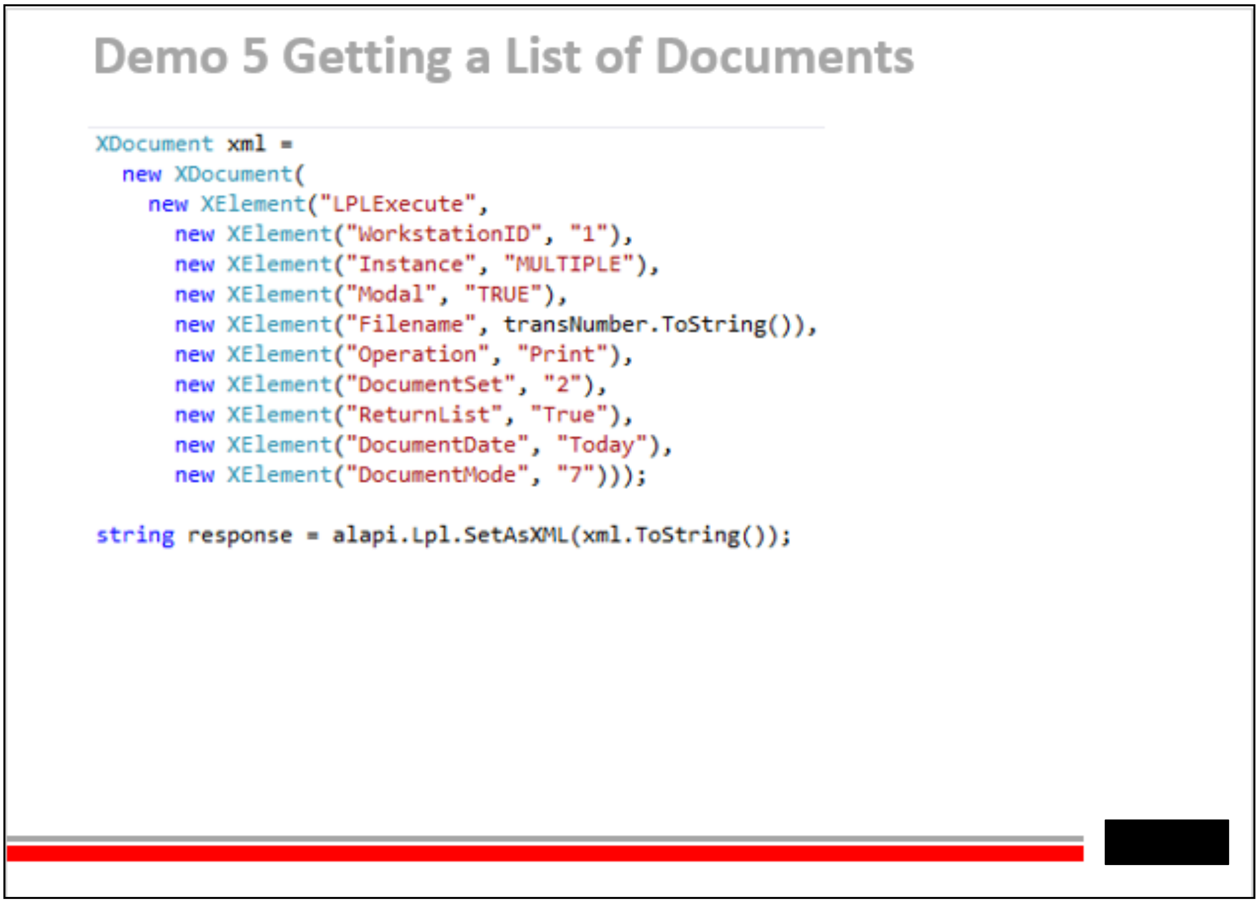
Demo 4 Executing LaserPro

```
XDocument xml =
  new XDocument(
    new XElement("LPLExecute",
      new XElement("WorkstationID", "1"),
      new XElement("Instance", "MULTIPLE"),
      new XElement("Modal", "TRUE"),
      new XElement("Filename", transNumber.ToString()),
      new XElement("Operation", "Transaction")));

alapi.Lpl.SetAsXML(xml.ToString());
```

[redacted]

Appendix E: Getting a List of Documents



Demo 5 Getting a List of Documents

```
XDocument xml =
  new XDocument(
    new XElement("LPLExecute",
      new XElement("WorkstationID", "1"),
      new XElement("Instance", "MULTIPLE"),
      new XElement("Modal", "TRUE"),
      new XElement("Filename", transNumber.ToString()),
      new XElement("Operation", "Print"),
      new XElement("DocumentSet", "2"),
      new XElement("ReturnList", "True"),
      new XElement("DocumentDate", "Today"),
      new XElement("DocumentMode", "7")));

string response = alapi.Lpl.SetAsXML(xml.ToString());
```

[redacted]

Index

{CLIENT_NAME} Support

{CLIENT_NAME} support offers several options to help you get the most out of your software, including a self-service Case Management tool, and phone support.

{CLIENT_NAME} support offers several options to help you get the most out of your software, including a self-service Case Management tool, and phone support.

Please visit the {CLIENT_NAME} Customer Center at https://customercenter.dh.com/ to log into our online self-service Case Management system. If you forgot your password, simply click the Forgot Password link. Once logged into Customer Center, you have the ability to use the Knowledge Center to troubleshoot issues and answer questions.

If your financial institution is not currently using these tools and would like to, please contact {CLIENT_NAME} support for assistance at 1-800-274-7287.

Note: The Financial Modernization Act of 1999, also known as the Gramm-Leach-Bliley Act, or GLB Act, includes provisions to protect consumers' personal financial information held by financial institutions. Therefore, {CLIENT_NAME} support cannot accept data or screen captures that contain personal financial information via e-mail or fax. For information about secure file transfer methods, contact {CLIENT_NAME} support.