System-to-System Developer's Guide

Revision: 3.1

Date: December 16, 2003

PURPOSE

This document describes the extension of the **smbXML** (**s**mall and **m**edium **b**usiness e**X**tensible **M**arkup **L**anguage) integration functionality to support the bi-directional exchange of business data between NetSuite products and third-party systems.

smbXML was developed to facilitate the exchange of business information between trusted, third-party, XML-enabled, business applications and the NetSuite product family. smbXML was originally implemented as a user initiated exchange of XML data via flat-file export and import. This was performed by selecting the file to import through the standard NetSuite web-browser interface, and is referred to as **smbXML UI Import**.

System-to-System (s2s) integration enables trusted systems to add, update and query data in an integrated NetSuite account in real-time, or in batch, without user intervention. Most often this is done in real-time, in response to an event occurring in the integrated third-party system (e.g.: a purchase being made in a third-party e-commerce application, that sends transaction, customer and item information to the NetSuite product regarding the sale).

System-to-System integration supports the same records and actions as the smbXML UI Import capability, available in the standard user interface.

TABLE OF CONTENTS

TABLE OF CONTENTS **2**

1 OVERVIEW **3**

1.1 CUSTOMERS AND APPLICATION PROVIDERS 3
1.2 DOCUMENT ROADMAP 3

2 IMPLEMENTATION OVERVIEW **4**

2.1 SUPPORTED PLATFORMS AND NETSUITE DEVELOPMENT SUPPORT 4
2.2 ARCHITECTURE 4
2.2.1 Two-Way Secure Socket Layer (SSL) *4*
2.2.2 Additional Certificate Verification *4*
2.2.3 Data Exchange Steps *4*
2.3 PRE-IMPLEMENTATION CHECKLIST 5
2.3.1 Integration Partner will Provide NetSuite *5*
2.3.2 NetSuite will Provide Integration Partner *5*
2.4 IMPLEMENTATION CYCLE 5

3 SETUP CLIENT CERTIFICATE **8**

3.1 JAVA PLATFORM 8
3.2 MICROSOFT DEVELOPMENT PLATFORM 10

4 IMPLEMENTATION DETAILS **13**

4.1 SMBXML 13
4.1.1 Related Documentation *13*
4.1.2 DTD Elements *13*
4.1.3 Many to One *14*
4.2 URLS AND HOSTNAMES 14
4.2.1 Post URL *14*

5 SINGLE SIGNON ACCOUNT MAPPING **15**

5.1 DO I NEED SINGLE SIGNON? 15
5.2 THE PROBLEM 15
5.3 THE SOLUTION 15
5.4 FURTHER RESOURCES AND DOCUMENTATION 15

6 USER INTERFACE OPTIONS **16**

APPENDIX A – REFERENCES **17**

APPENDIX B – JAVA CODE SAMPLE **18**

APPENDIX C – MICROSOFT .NET C# CODE SAMPLE **19**

1 OVERVIEW

1.1 CUSTOMERS AND APPLICATION PROVIDERS

Generally, there are two types of integration partners, **customers** and **application providers**.

Customers are themselves users of a NetSuite product, and are interested in integrating their NetSuite application data with that of an external, third-party, application's data. An example would be a NetSuite customer that ran an external, custom, ecommerce application, that feeds sales information (transactions, customers, items, etc.) into their NetSuite account.

Application Providers are providers of applications and services to end-customers, who are themselves customers of NetSuite. Application providers are interested in integrating their customers' application data, stored within the application provider's datacenter, with their customers' NetSuite data. An example would be a third-party Point-of-Sale (POS) provider, that wanted to transmit their customers' sales information (transactions, customer, items, etc.) into the NetSuite accounts of their end-customers. In this scenario, the application provider acts as an agent on behalf of our mutual customer.

Throughout this document, customers and application providers will be called **integration partners**, collectively.

Before proceeding with this document, you should know whether you are a customer or an application provider. For more explanation on the difference between customers and application providers, please refer to the **Customer v. Application Provider** document available with the NetSuite Developer Program Technical Documentation, or with your NetSuite Account Manager.

1.2 DOCUMENT ROADMAP

NetSuite's System-to-System security model uses several technologies and methods to ensure secure communication and enforce appropriate integration partner access to user data. The following sections will describe:

- Implementation overview that includes platforms supported by NetSuite, the System-to-System architecture, an implementation checklist, and an overview of the implementation cycle.
- Step-by-step instructions for the creation of a keystore containing the client certificate
- Overview of smbXML and the DTD elements specific to System-to-System data exchange
- Brief overview of the **NetSuite Single Signon (SSO)** technology used to map accounts between systems, and enable application provider access to a user's NetSuite data
- User interface options relevant to System-to-System integration

2 IMPLEMENTATION OVERVIEW

2.1 SUPPORTED PLATFORMS AND NETSUITE DEVELOPMENT SUPPORT

While our integration technology to establish System-to-System connectivity is based on standards and can be used from any development platform, NetSuite officially supports the following two development platforms.

- Java platform
- Microsoft .NET platform

If you're using one of these platforms, NetSuite will be able to provide a command line driven test client that is designed to test SSL connectivity to NetSuite, post smbXML documents to NetSuite, and to serve as an example for the development of the integration partner's application code. The tool includes the source code.

Currently, the primary development expertise of NetSuite Professional Services are with the above mentioned platforms, and Professional Services ability to advise and consult will be strongest for integration partners whose implementation is in these platforms. If the client application to be implemented uses a different language or platform, as long as this environment supports two-way SSL communications, there should not be any problems in implementation since there are no NetSuite proprietary libraries that need to be used.

2.2 ARCHITECTURE

2.2.1 TWO-WAY SECURE SOCKET LAYER (SSL)

Put simply, System-to-System integration allows data (in the form of an smbXML document) to be posted to, or queried from, NetSuite accounts via an HTTPS post.

NetSuite uses a **Two-Way Secure Socket Layer** authentication methodology, for System-to-System communications, that ensures that both parties are who they claim to be. The client is required to validate NetSuite's server certificate, and to provide a client certificate in order to authenticate themselves to NetSuite.

NetSuite's Two-Way Secure Socket Layer (SSL) uses industry standard 128-bit SSL encryption, as well as certificate hash comparison. Server authentication takes place over an SSL connection initiated with NetSuite's key pair signed by a mutually trusted key signer (in this case RSA). The certificate-based client authentication uses a client generated key pair, the public key of which is signed by and possessed by NetSuite. For details on certificate-based SSL, please see **Appendix A** of this document.

Concisely, Two-Way SSL allows for the following:

- The Client verifies that the Server is who it claims to be.
- The Server verifies that the Client is who it claims to be.

2.2.2 ADDITIONAL CERTIFICATE VERIFICATION

As a further check on authentication, once the XML request document is received, NetSuite performs secondary validation of the client certificate by comparing a hash of the certificate to a previously stored hash, taken during the signing phase.

This second validation of the certificate is performed to ensure that the certificate being presented matches identically the certificate originally provided by the partner.

2.2.3 DATA EXCHANGE STEPS

1. integration partner system generates a valid smbXML document

2. integration partner opens an HTTPS connection to the NetSuite smbXML server
3. during the SSL handshake
   a. the integration partner verifies NetSuite's identity by validating the NetSuite server certificate, signed by RSA
   b. the NetSuite server verifies the identity of the integration partner by requesting a client-side certificate, signed by NetSuite
   c. the smbXML server returns a symmetric key, encrypted using the integration partner's public key, used to establish SSL communication
   d. SSL is established
4. partner system transmits smbXML document to NetSuite in the body of an HTTP Post
5. the smbXML server performs secondary validation of the certificate, by comparing a hash of the certificate to a previously stored hash of the certificate
6. the smbXML server parses the smbXML document and determines whether the partner system has the appropriate permissions to perform the requested action(s) in the account specified
7. the smbXML server processes the requests and returns the results in the body of an HTTP Response with:
   e. any appropriate success or error messages
   f. the results of any query requests

2.3 PRE-IMPLEMENTATION CHECKLIST

Prior to beginning development and testing of XML data transmission, NetSuite and the integration partner will exchange certificates, public keys, URLs and partner IDs.

2.3.1 INTEGRATION PARTNER WILL PROVIDE NETSUITE

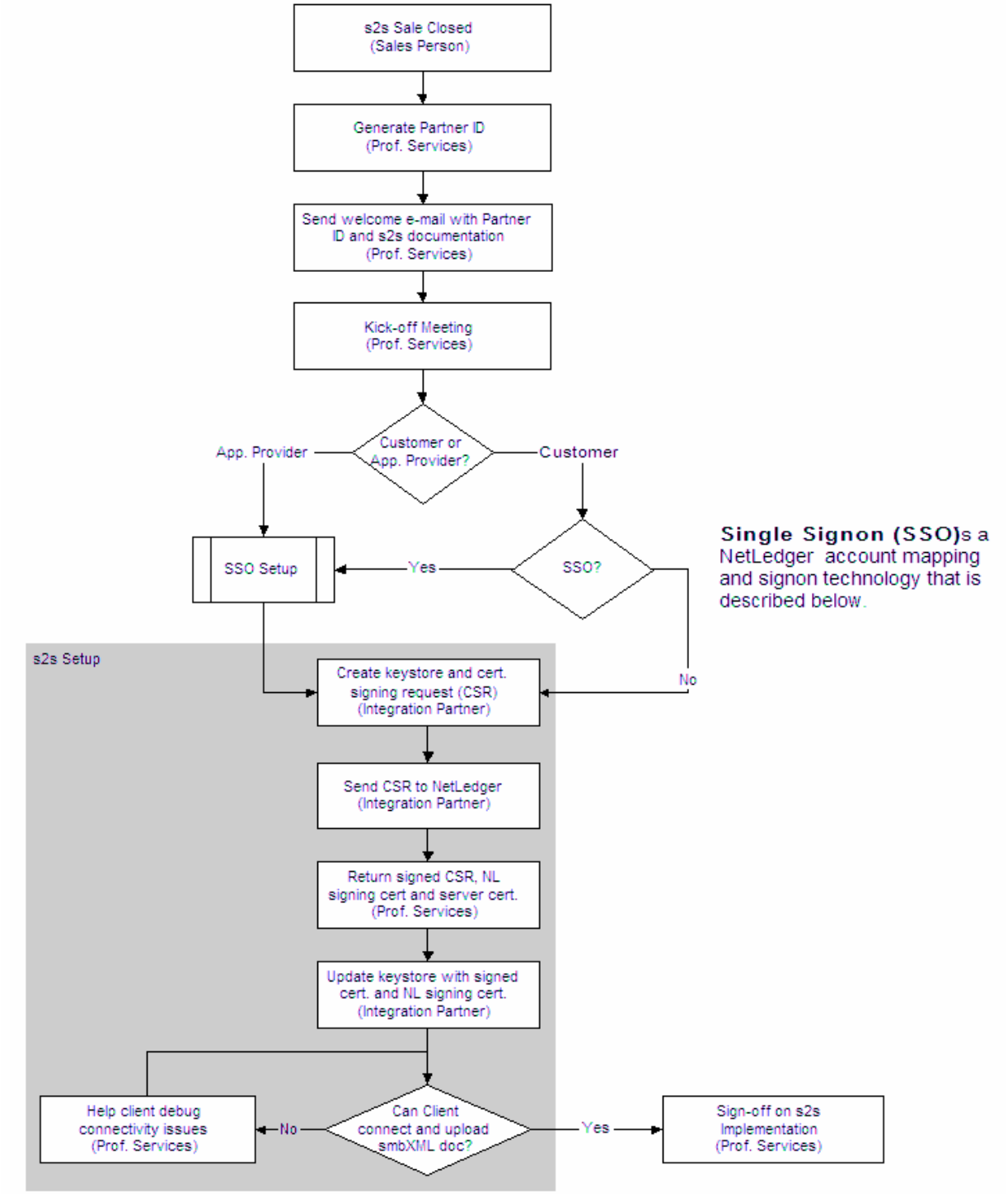
- *certificate signing request (CSR)* – signed by NetSuite, for establishing client authentication
- *1024-bit public key* – for use in Single Signon **(application providers only)**
- *company name* – used as the partnerAccount, discussed below under DTD Elements **(customers only)**

2.3.2 NETSUITE WILL PROVIDE INTEGRATION PARTNER

- *unique partner ID* – used during System-to-System and Single Signon communication, discussed below under DTD Elements and Single Signon
- *signed client certificate* – the signed client certificate signing request
- *NetSuite signing certificate* – the NetSuite certificate used to sign the client CSR
- *RSA signing certificate* – the public RSA certificate that signed our server certificate, discussed above under the Architecture Overview, SSL handshake section
- *post URL* – the production post URL, used by the client to post smbXML documents to NetSuite

2.4 IMPLEMENTATION CYCLE

The following workflow details the implementation steps integration partners undertake, and the NetSuite organizations responsible for each step.



Following is a detailed list of steps:

1. Create a sample smbXML document.
   - Ensure that it is valid against the current version of the smbXML DTD.
   - Ensure that it can be imported into NetSuite using the smbXML UI Import (Transactions > Other > Import XML Document).

2. Setup certificates for Two-way SSL communications.
   - Note that an integration partner may register more than one certificate with NetSuite in the event that the integration partner is using multiple servers for data exchange with NetSuite, although this is not required.
3. Write and test the necessary code in order to open a SSL connection.
   - Please see **Appendix B** for code samples.
   - Ensure that the sample smbXML document used above is used for testing.
   - Note that smbXML documents posted via System-to-System have a required header, discussed below in the DTD Elements section, that is optional during an smbXML UI Import.
4. Write your application specific code to generate smbXML documents and send them to NetSuite using the SSL connection.
5. Implement Single Signon Account Mapping
   - If you are an application provider, this is a mandatory step.
   - If you are a customer, this step is optional, unless you want to provide Single Signon capability to your employees.
6. Arrange for NetSuite Professional Services to sign-off on your implementation.

3 SETUP CLIENT CERTIFICATE

As discussed previously, NetSuite's Two-Way Secure Socket Layer authentication uses public/private key technology to authenticate clients against NetSuite servers and also to authenticate NetSuite servers to connecting clients. This authentication uses a client generated key pair the public key of which is signed by and possessed by NetSuite. Authentication takes place over an SSL connection initiated with NetSuite's key pair signed by a mutually trusted key signer (in this case RSA). The SSL portion of this process guarantees to the client that NetSuite is NetSuite and the NetSuite authentication portion of this process guarantees to NetSuite that the client is the client who they claim to be.

There are three certificates that are used in NetSuite's Two-Way Secure Socket Layer authentication:

1. the integration partner's client certificate, signed by us (which is the signed CSR)
2. the NetSuite certificate used to sign your CSR (**NLCACert4Partners.der**)
3. the RSA certificate used to sign our server certificate (**RSASignedCERT.cer**)

The certificates from 1 and 2 above are stored within your keystore – the same one that you used to generate the CSR in the first place. *A keystore contains your public and private key-pair used to create the CSR, as well as various certificates. The following sections describe the construction and use of keystores in greater detail.*

The RSA certificate, from 3, is used at the beginning of an SSL handshake, when you are verifying our server certificate, to assure yourselves that we are who you think we are. You may obtain the RSA certificate directly from our site (using a browser and by clicking the IE security lock icon), from RSA directly, or wait for us to send it to you (we always send a copy of all three certificates when replying to CSRs).

The following steps outline the process in detail. Note that these steps use a fictitious company named **Client, Inc.** You should be able to use these commands exactly as they appear by replacing the example references to Client, Inc, and its related password and company information.

If the integration partner has more than one NetSuite account that they need to enable system-to-system access for, they have the following options:

- They may register a certificate for each NetSuite account.
- They may use a single certificate across all accounts. Note that unlike server certificates, a client certificate is not bound to a specific physical machine, and therefore, a certificate can be copied to other machines.

3.1 JAVA PLATFORM

When using the Java platform, creating a client certificate is best achieved by using Java's native support for certificates. Following is a detailed list of steps required to create a certificate, get it signed by NetSuite, and install the certificate.

Step 1: Install the Java 2 Platform, Standard Edition JDK version 1.4 or higher

You will need the keytool program that comes with this version of the JDK in order to manage keystores. Once the JDK has been installed, and the bin directory of the JDK has been added to the system path, you should be able to run the keytool command at the command prompt.

Step 2: Generate a new public/private key pair in a new keystore as follows

A new public/private key pair is generated in a new keystore. The keystore should be stored in a secure location.

In the following example, a 2048-bit RSA private/public key pair with an alias of *client* and a password of *mykeypass* is generated and stored in the file *client.keystore*, which is protected with the password *mystorepass*.

```
keytool -genkey -dname "cn=client.com, ou=Engineering, o=Client, c=US,
st=California" -alias "client" -keypass mykeypass -keystore "client.keystore" -
storepass mystorepass -keyalg "RSA" -keysize 2048
```

The *–dname* parameter specifies the following. If you omit this parameter, the tool will prompt you for this information.

Name	Description
CN	Valid domain name for your company
O	Company name
OU	Organizational Unit
C	Country
ST	State

Step 3: Create a Certificate Signing Request (CSR)

The public key generated above needs to be signed by a Certificate Authority (CA). The new key pair is used to generate a Certificate Signing Request or CSR. A CSR is a file that is sent to the certification authority for signing; it contains the public key that needs to be signed in a special format. Note that the CA in this case is NetSuite.

```
keytool -keystore client.keystore -keypass mykeypass -storepass mystorepass -
alias client -certreq -file client.csr
```

The above command uses a keystore named *client.keystore* with a password of *mykeypass* to generate a CSR named *client.csr* for the key with an alias of *client*.

Step 4: E-mail the CSR to NetSuite

NetSuite will act as the CA and sign the certificate. It will also extract a hash of the certificate and store it internally so that a secondary check can be performed on all incoming requests.

The e-mail should be sent to certSignReq@netsuite.com with the CSR file as an attachment. It should include the following:

- Subject: Certificate Signing Request for <Company Name>
- Body:
  - Partner ID: The partner ID assigned by NetSuite
  - NetSuite Account Number(s): The accounts that require System-to-System access. May include a production account and a test account. You can obtain your NetSuite account number as follows: NetSuite Home > Set Up Synchronization (Under Settings Portlet) > Account number is displayed in text box.
  - Partner Account Name: Provide a logical name that you would like your NetSuite account provided above to be referred by **(customers only).** For example, if your company name is ABC Corp, you might have ABC_Production and ABC_Test as your account names. Solution providers do not provide a Partner Account, unless they have made special arrangements with Professional Services to manually configure account mapping; otherwise, Partner Account is determined dynamically, as established via Single Signon Account Mapping, discussed in the Single Signon section.
  - The Certificate Signing Request as an attachment.

Step 5: Update the keystore with the NetSuite certificate and the signed certificate

NetSuite will e-mail you the signed client certificate as well as NetSuite's self-signed, signing certificate (named *NLCACert4Partners.der*), that was used to sign your client certificate. You will need to import both of these certificates into your keystore. First import NetSuite's signing certificate into your keystore as a trusted CA. When asked if you want to trust this certificate, answer yes.

In the following example, the NetSuite certificate named *NLCACert4Partners.der* is imported into the keystore named *client.keystore* as a trusted CA.

```
keytool -keystore client.keystore -storepass mystorepass -alias CA -keypass
mykeypass -import -file NLCACert4Partners.der
```

You will see a response similar to the following, and will be prompted to trust the certificate.

```
Owner: EMAILADDRESS=dwilliams@netsuite.com, O="NetSuite, Inc.", ST=California,
L=San Mateo, C=US,
CN=system.NetSuite.com
Issuer: EMAILADDRESS=dwilliams@netsuite.com, O="NetSuite, Inc.", ST=California,
L=San Mateo, C=US,
 CN=system.NetSuite.com
Serial number: 1
Valid from: Fri Aug 11 08:53:00 PDT 2000 until: Mon Aug 09 08:53:00 PDT 2010
Certificate fingerprints:
         MD5:  65:17:3D:5B:3C:F4:AB:45:D8:CA:EC:E2:BC:CC:9D:51
         SHA1: ED:60:C9:73:17:DC:13:1B:AB:0E:41:7C:28:18:DE:99:51:81:71:DB
Trust this certificate? [no]:  y
Certificate was added to keystore
```

Next, import the signed client certificate into your keystore. The keytool will check the signatories of the certificate to ensure that their signatures can be validated. In this case, since the NetSuite certificate (public key) is already in the same keystore, this validation is successful.

In the following example, the signed certificate named *client.der* is imported into the keystore named *client.keystore* with a password of *mystorepass* under the alias *client* with a password of *mykeypass*.

```
keytool -keystore client.keystore -storepass mystorepass -alias "client" -
keypass mykeypass -import -file client.der
```

3.2 MICROSOFT DEVELOPMENT PLATFORM

This section details how to setup certificates using OpenSSL that are intended for use primarily within the Microsoft .NET platform. These directions can also be used to setup certificates for use with other Microsoft development platform such as Active Server Pages (ASP), Visual C++, Visual Basic, as well as other external Microsoft Windows based development platforms (for example, Delphi) although these platforms are not officially supported by NetSuite. Please see section 2.1 entitled Supported Platforms and NetSuite Development Support for more details on platforms supported by NetSuite.

Following is a detailed list of steps using OpenSSL to create a certificate, getting it signed by NetSuite, and installing the certificate.

Step 1: Install OpenSSL

Download the latest release of OpenSSL in one of the following formats and install. Note that NetSuite recommends installing Cygwin as it is the quickest and most reliable option.

- Install the latest version of Cygwin from http://www.cygwin.com and ensure that the OpenSSL option is checked before installing. This is the recommended option as Cygwin comes with the required configuration files for OpenSSL.

- Install a pre-compiled version of OpenSSL in executable format for Windows. Note that the official site http://www.openssl.org has only the source and does not contain a pre-compiled version for Windows. Find a mirror site such as http://hunter.campbus.com/ to download the zip file that contains *openssl.exe* and a couple of related DLLs. This requires setting up an OpenSSL configuration file that you need to find elsewhere on the Internet.

Once installed, ensure that these executables are in your system path.

Step 2: Create a Certificate Signing Request (CSR)

Generate a key pair with a RSA 1024 bit private key, and a Certificate Signing Request (CSR) as follows. Note that in this example, the private key is stored in *client.key* and the certificate expires in 10 years. You will also be prompted for a password to protect you private key.

```
openssl req -new -keyout client.key -out client.csr -days 3650
```

A CSR is a file that is sent to the certification authority for signing; it contains the public key that needs to be signed in a special format. Note that the CA in this case is NetSuite.

As part of this request, you will be prompted for information that will be inserted into your certificate. Following is a sample log that illustrates this information.

```
Generating a 1024 bit RSA private key
...........++++++
.....................................++++++
writing new private key to 'client.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:CA
Locality Name (eg, city) []:San Mateo
Organization Name (eg, company) [Internet Widgits Pty Ltd]:XYZ Inc
Organizational Unit Name (eg, section) []:Engineering
Common Name (eg, YOUR name) []:XYZ Inc
Email Address []:info@xyz.com
```

Step 3: E-mail the CSR to NetSuite

NetSuite will act as the CA and sign the certificate. It will also extract a hash of the certificate and store it internally so that a secondary check can be performed on all incoming requests.

The e-mail should be sent to certSignReq@netsuite.com with the CSR file as an attachment. It should include the following:

- Subject: Certificate Signing Request for <Company Name>
- Body:
  - Partner ID: The partner ID assigned by NetSuite

- NetSuite Account Number(s): The accounts that require System-to-System access. May include a production account and a test account. You can obtain your NetSuite account number as follows: NetSuite Home > Set Up Synchronization (Under Settings Portlet) > Account number is displayed in text box.
- Partner Account Name: Provide a logical name that you would like your NetSuite account provided above to be referred by **(customers only).** For example, if your company name is ABC Corp, you might have ABC_Production and ABC_Test as your account names. Solution providers do not provide a Partner Account, unless they have made special arrangements with Professional Services to manually configure account mapping; otherwise, Partner Account is determined dynamically, as established via Single Signon Account Mapping, discussed below.
- The Certificate Signing Request as an attachment.

Step 4: Add NetSuite certificate and the signed certificate to Certificate Store

NetSuite will e-mail you the signed client certificate (named *client.pem* in this example), as well as NetSuite's self-signed certificate (named *NLCACert4Partners.der*), that was used to sign your client certificate.

Double click on the *NLCACert4Partners.der* certificate to install it as a trusted root CA certificate under the Trusted Root Certification Authorities store for the current user logged into Windows as follows:

- Click **Install Certificate …**
- Click **Next** in the *Welcome to the Certificate Import Wizard* dialog box
- Select the *Automatically select the certificate store based on type of certificate* radio button and click **Next**
- Click **Finish**

Next, convert your certificate into the PKCS12 format by executing the following OpenSSL command. The PKCS12 format is a certificate store that stores both your signed certificate and private key in one file.

```
openssl pkcs12 -export -clcerts -in client.pem -inkey client.key -out client.p12
```

Double click on the *client.p12* file to install the certificate in the Personal Certificate Store under the current user logged into Windows as follows:

- Click **Next** in the *Welcome to the Certificate Import Wizard* dialog box
- Click **Next** in the *File Import* screen
- Check the *Mark this key as exportable* check box and click **Next**
- Select the *Automatically select the certificate store based on type of certificate* radio button and click **Next**
- Click **Finish**

You can verify that these certificates were imported successfully by using Internet Explorer and selecting *Tools* > *Internet Options* > *Content* > *Certificates*, and looking at the *Personal* and *Trusted Root Certification Authorities* tabs.

4 IMPLEMENTATION DETAILS

4.1 SMBXML

4.1.1 RELATED DOCUMENTATION

smbXML is used for all communications between the integration partner and NetSuite. The following smbXML-related documentation may be found with the NetSuite Developer Program Technical Documentation on the NetSuite website (http://www.netsuite.com > Partners > Developer Program).

- DTD
- release notes
- smbXML Developer's Guide
- example import files
- customer v. application provider document
- methodology documents
- DTD and release note archive

The current DTD for smbXML is available at the following URL:

- **http://www.netsuite.com/xml/dtd/smb_X_XX.dtd**
  where X_XX represents the current DTD revision number

4.1.2 DTD ELEMENTS

The smbXML DTD is used for both smbXML UI Import, using the standard web-based interface, as well as for System-to-System data exchange. In order to support System-to-System smbXML, there have been two minor changes to the smbXML DTD. The existing <smbxml> element has been modified to include a header element, required for System-to-System data exchange, but which remains optional/ignored during smbXML UI import. The new header element, referenced in the <smbxml> element, has been added and defined as:

<!ELEMENT smbxml ((header?, request+) | response+)>

<!ELEMENT header (partnerId, partnerAccount)>

Where,

- partnerId is the integration partner's unique identifier, as provided to the partner by NetSuite (typically five numeric digits long)
- partnerAccount is the integration partner's unique account name for the target company to post data to in NetSuite
  - In the case of a customer integration, partnerAccount is the customer's company name, for their own account, and is provided once during the initial setup phase. For a customer, the partnerId will always be the same value.
  - In the case of an application provider integration, partnerAccount is the application provider's company name, for the shared customer account, and is dynamically determined based upon which mutual customer's account they wish to post data into..

4.1.3 MANY TO ONE

NetSuite accepts smbXML data from multiple integrated partner accounts into a single NetSuite account. This supports activities such as a NetSuite customer aggregating cash sales, from multiple external Webstores, into a single NetSuite account; or aggregating smbXML purchase orders and bills from multiple exchanges, such as the Oracle exchange. NetSuite does not currently support multiple accounts, from the same integration partner application, integrating with a single NetSuite account. For example, while NetSuite supports data from Company A in Partner 1, and Company B in Partner 2, both feeding into the same NetSuite account ACCT X, NetSuite does not support data from Company A in Partner 1, and Company B in Partner 1, both feeding into the same NetSuite account ACCT X.

Note that the current implementation also does not provide built in support to enable different integration partners to elegantly share common data in a single NetSuite account. This will be addressed in a forthcoming specification. This new specification, **Multi-Partner Synchronization Support**, will provide a revision system for each smbXML accessible object (customer, vendor, inventory items, cash sale, invoice, etc.) and will prevent integration partners from inadvertently overwriting another partner's edits to records (or to those made in the UI, for that matter).

4.2 URLS AND HOSTNAMES

4.2.1 POST URL

The URL that is used to establish a System-to-System connection with NetSuite and post smbXML documents to is as follows:

https://partners.netsuite.com/s/SmbXml?pid=<partnerId>&pacct=<partnerAccount>

- The <partnerId> is the 5-digit identifier assigned by NetSuite to the partner/customer for partner mapping purposes. This is the same partnerId that is contained within the smbXML header.
- The <partnerAccount> is the partner's unique company account name for the current request. This is the same partnerAccount that is contained within the smbXML header.

Following is an example where the partner Id is 22999 and the partner account name is ShutterFly:

https://partners.netsuite.com/s/SmbXml?pid=22999&pacct=ShutterFly

5 SINGLE SIGNON ACCOUNT MAPPING

5.1 DO I NEED SINGLE SIGNON?

If you are an application provider, yes. For an application provider, SSO is an essential part of System-to-System integration.

If you are a customer, no. For a customer, SSO has nothing to do with System-to-System integration. A customer may still wish to implement SSO as it will allow customer employees to move from the customer application into their NetSuite account without the need to login (SSO is currently included with the System-to-System integration setup and maintenance fees).

If you are an application provider, or you are a customer that is considering implementing SSO for its seamless login capabilities, then read on further in this section; otherwise you may move on to the next section.

5.2 THE PROBLEM

In order for an integration partner to post data into a NetSuite account, the partner must identify the target account in NetSuite. NetSuite does not want integration partners to have to maintain lists of NetSuite's internal account identifiers. Consequently, NetSuite requires the integration partner to identify the target company with a unique identifier of their own choosing, passed via the partnerAccount element of the smbXML document header element as discussed in section 4.1.2. NetSuite then maintains a mapping between the integration partner company identifiers, and NetSuite's own internal identifiers.

In the case of customers, there is only ever one identifier, that of the customer itself. In the case of application providers, there may be many shared companies, each with their own unique identifiers in both the application provider system and within NetSuite. For customers, NetSuite enters the mapping table entries manually; this would be prohibitive in the case of application providers, and, therefore, an automated solution is used: **Single Signon Account Mapping**.

5.3 THE SOLUTION

NetSuite's Single Signon (SSO) technology was originally developed to allow shared customers to move between trusted, integrated applications, without the need to re-authenticate (i.e. re-present account/password information).

NetSuite's Single Signon technology uses RSA cryptography to exchange information securely, and establish trust between NetSuite and its integration partners. Trust and account mapping are established when the application provider Web application redirects the shared customer to an XML linking portal. This initial account mapping and trust must be initiated by the administrator of the NetSuite account. This ensures that smbXML data exchange is a conscious decision by the NetSuite account administrator. Once the account mapping is established, XML exchanges between the application provider and NetSuite are permitted on behalf of the shared customer.

5.4 FURTHER RESOURCES AND DOCUMENTATION

The redirect string may be generated using a java library provided by NetSuite, or through partner-specific tools developed to construct a Single Signon compliant redirect string. Using the NetSuite tools will reduce costs and development time associated with testing a new implementation of token generation and encryption.

Complete documentation on NetSuite's SSO technology can be found in NetSuite's **Single Signon Developer's Guide,** available from your NetSuite Account Manager or Professional Services Manager. The SSO integration utilities and software license will only be provided to integration partners that have signed NetSuite's Mutual Non-Disclosure Agreement. In order to use the SSO integration utilities, the integration partner must agree to the terms of the Single Signon license agreement.

6 USER INTERFACE OPTIONS

Single Signon may be turned off, for those customers that wish to only enable smbXML data exchange, from the Setup tab > Accounting sub-section > Set Up Accounting link > Web sub-tab, by clearing the Allow Sign On From Partners checkbox.

smbXML data exchange may be turned off, on a integration partner by partner basis, from the Setup tab > Import/Export sub-section > Set Up XML Import link > by clearing the Allow XML From Partners checkbox for the appropriate partner. Note that this section and the corresponding checkboxes will only appear if there are partners that have been enabled for smbXML data exchange for the account in question.

APPENDIX A – REFERENCES

- How SSL Works, http://developer.netscape.com/tech/security/ssl/howitworks.html
- Introduction to Public-Key Cryptography, http://developer.netscape.com/docs/manuals/security/pkin/contents.htm
- Set Up a Certification Authority for Java-based Systems, http://www.devx.com/Java/Article/10185/1954?pf=true
- X.509 Certificates and Certificate Revocation Lists (CRLs), http://java.sun.com/products/jdk/1.2/docs/guide/security/cert3.html
- Implementing Web Site Client Authentication Using Digital IDs, http://www.verisign.com/clientauth/kit/details.html

APPENDIX B – JAVA CODE SAMPLE

Following is a Java code sample taken from the NetSuite smbXMLPost tool that illustrates how a two-way SSL connection is established using a keystore.

```
import javax.net.ssl.*;
import java.net.*;
import java.security.*;
import java.security.cert.*;

// Private key password
String keyPassword = "mykeypass";
// Keystore password
String storePassword = "mystorepass";
// Fully qualified path to keystore file
String keystoreName = "my.keystore";

// Read the keystore file and load the keystore
KeyStore keystore = null;
InputStream keystoreStream = new ByteArrayInputStream( base64Decode(
keystoreName.getBytes() ));
keystore = KeyStore.getInstance("JKS");
keystore.load(keystoreStream, storePassword.toCharArray());

// Initialize the KeyManagerFactory
KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance("SunX509");
keyManagerFactory.init(keystore, keyPassword.toCharArray());

// Create and initialize SSLContext and generate random number
SSLContext ssl = SSLContext.getInstance("SSL");
SecureRandom sr = new SecureRandom();
sr.nextInt();
ssl.init(keyManagerFactory.getKeyManagers(), null, sr);

// Open HTTP connection
URL url = new URL(m_sUrl);
HttpURLConnection conn = (HttpURLConnection) url.openConnection();

// Set connection to include SSL context
SSLSocketFactory sf = ssl.getSocketFactory();
((javax.net.ssl.HttpsURLConnection)conn).setSSLSocketFactory (sf);

// Set other connection parameters and load request data
...

// Connect and push the request into the connection socket
conn.connect();


// This method used above decodes the given byte[] using the
// base64-encoding specified in RFC-2045 (Section 6.8).

public final static byte[] base64Decode(byte[] data)
{
    ...
}
```

APPENDIX C – MICROSOFT .NET C# CODE SAMPLE

Following is a Microsoft .NET C# code sample taken from the NetSuite smbXMLPost tool that illustrates how a two-way SSL connection is established in the .NET platform.

```
// NetSuite URL to post to
string url =
   "https://partners.netsuite.com/s/SmbXml?pid=22999&pacct=ShutterFly";

HttpWebRequest req;
HttpWebResponse res;

// Create request object that connects to NetSuite
req = (HttpWebRequest) WebRequest.Create( url );
req.Method = "POST";
req.ContentType = "text/xml";

// The path to the certificate.
string certFilePath =  "C:\\client.der";

// Load the certificate into an X509Certificate object.
X509Certificate x509Cert = X509Certificate.CreateFromCertFile( @certFilePath );

// Add certificate to request
req.ClientCertificates.Add( x509Cert );

// Read smbXML file and write contents to request stream
StreamReader inFileReader = new StreamReader("c:\\queryAllSalesOrders.xml");
StreamWriter reqWriter = new StreamWriter( req.GetRequestStream() );
string sLine = null;
while ( ( sLine = inFileReader.ReadLine() ) != null )
{
   reqWriter.WriteLine( sLine );
}
inFileReader.Close();
reqWriter.Close();

// Get response
res = (HttpWebResponse) req.GetResponse();
System.Console.Out.WriteLine( "Status Code: " + res.StatusCode );
StreamReader responseReader =
   new StreamReader(res.GetResponseStream(), Encoding.ASCII);

// Put the results in a string and print to console
System.Console.Out.WriteLine("\n");
strResult = responseReader.ReadToEnd();
responseReader.Close();

Console.WriteLine(strResult);
```