Building Block Configuration Guide
SAP Emarsys Starter Pack
March 2023
English

CUSTOMER

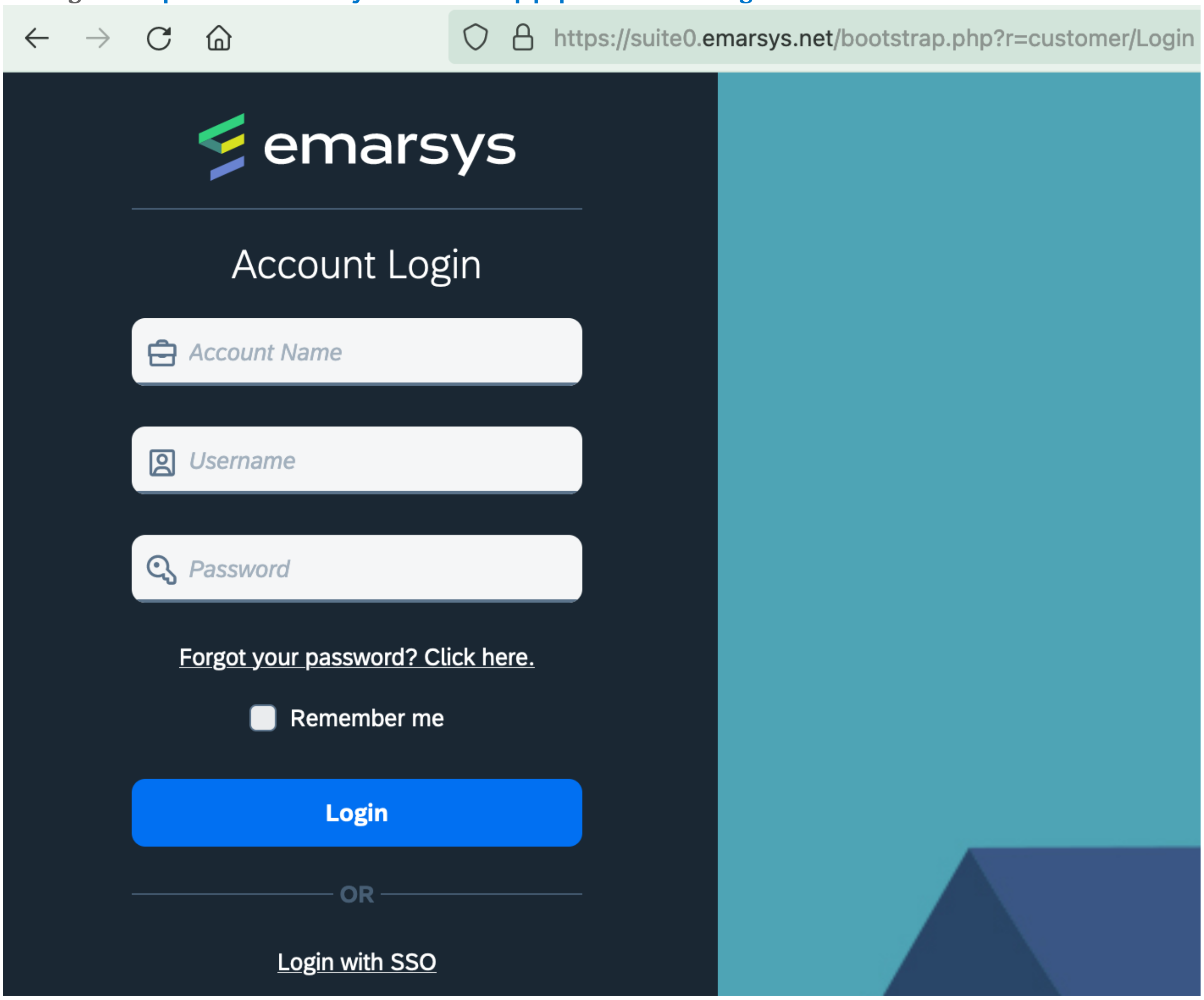
Cloud Integration Starter Pack for Integration with SAP Emarsys (Sample - Contact Replication)

Content

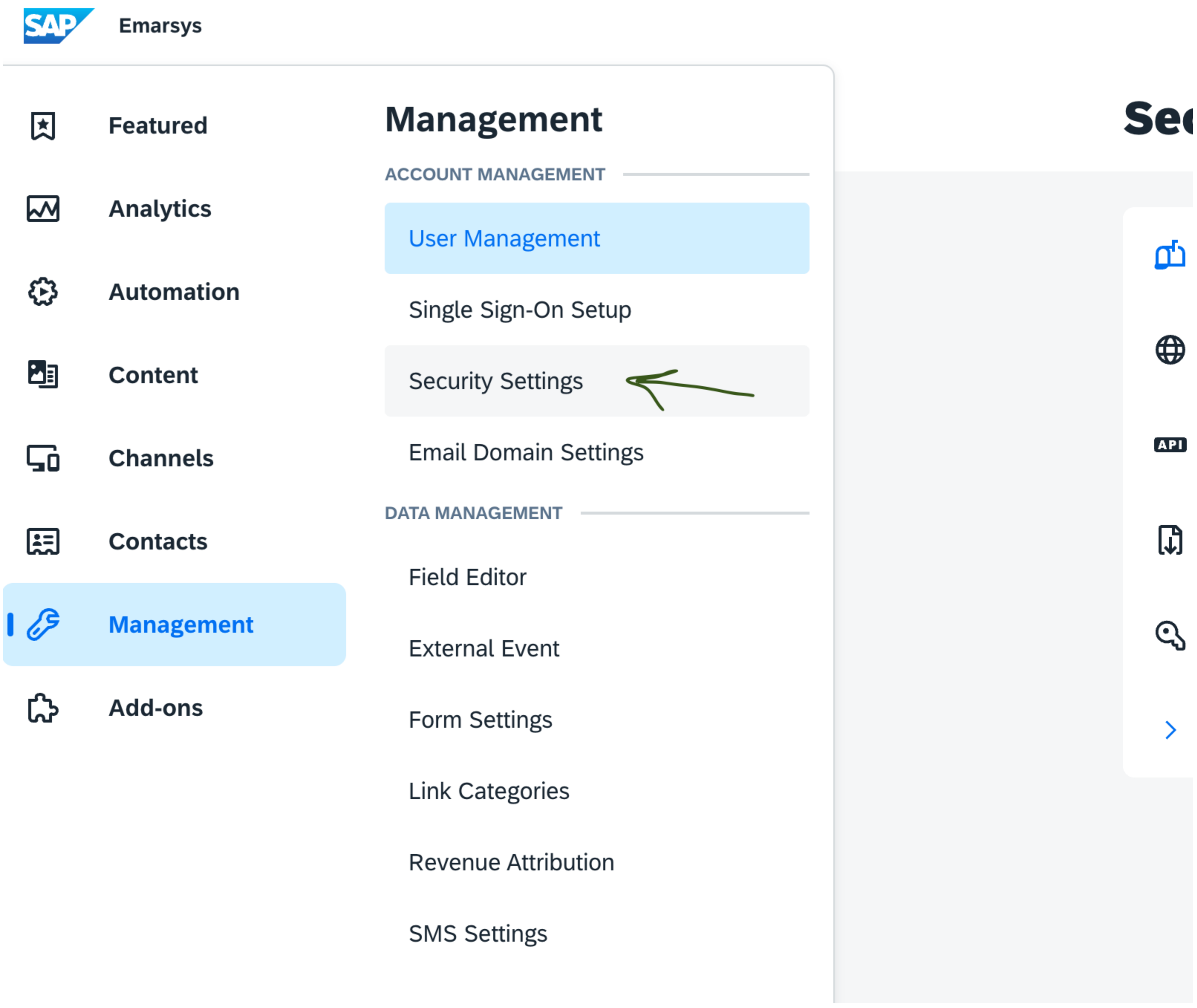
1 Prerequisites

SAP Emarsys Setup

1. **Login to https://suite0.emarsys.net/bootstrap.php?r=customer/Login**


https://suite0.emarsys.net/bootstrap.php?r=customer/Login
emarsys
Account Login
Account Name
Username
Password
Forgot your password? Click here.
Remember me
Login
OR
Login with SSO

2. **Navigate to Management -> Security Settings (you need to have Administrator or Account Owner right)**



Management

ACCOUNT MANAGEMENT

User Management

Single Sign-On Setup

Security Settings

Email Domain Settings

DATA MANAGEMENT

Field Editor

External Event

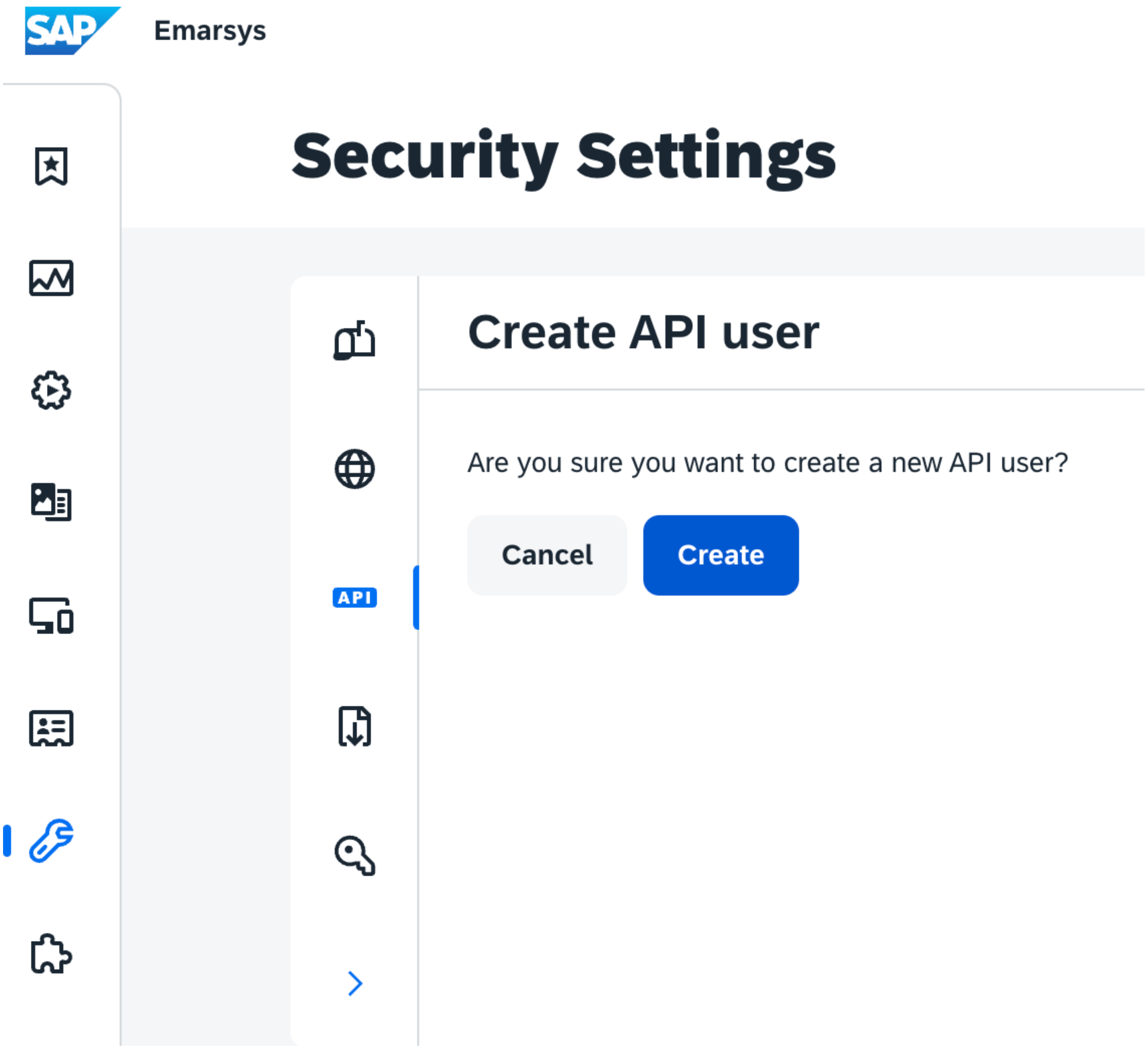
Form Settings

Link Categories

Revenue Attribution

SMS Settings

3. **From Security Settings list choose API users**

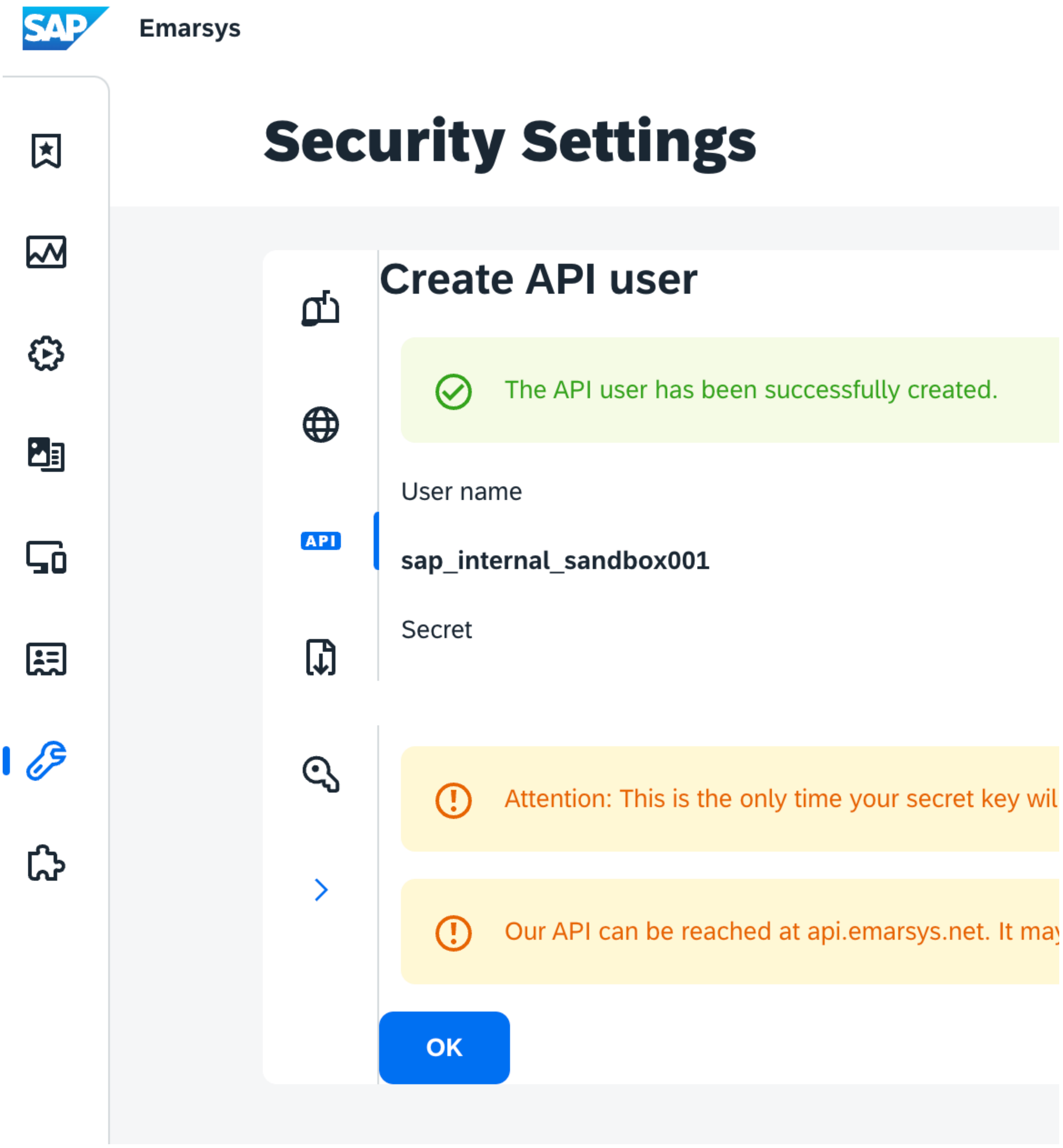


Security Settings

Create API user

Are you sure you want to create a new API user?

4. After pressing Create, the API user will be created.
IMPORTANT: Please note down the Username and Secret as the secret will not be visible in the future.



Security Settings

Create API user

The API user has been successfully created.

User name

sap_internal_sandbox001

Secret

Attention: This is the only time your secret key wil

Our API can be reached at api.emarsys.net. It may

OK

2 Documentation

This guide illustrates the necessary steps for setting up an SAP Emarsys System user and configuring an integration flow to create and read a contact via Emarsys API.

Authentication

SAP Emarsys API uses WSSE authentication over SSL to keep the data secure. WSSE authentication is not a standard HTTP authentication mechanism. So, it needs to get generated and passed in as a custom HTTP header (X-WSSE) header within each HTTP request. The header generated from the created username and secret in SAP Emarsys.

Username Token: Indicates that the authentication method of WSSE is token-based.

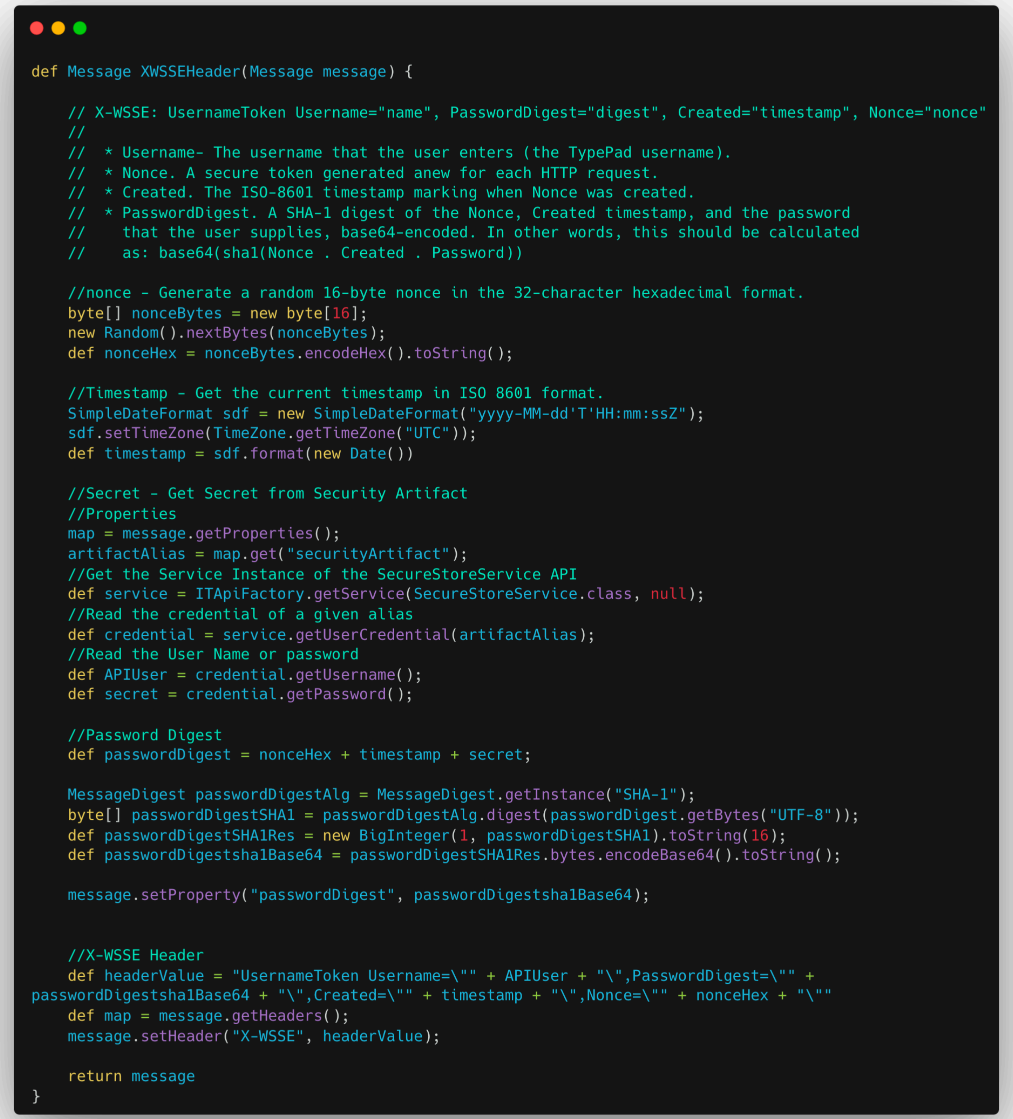
Username

Nonce: A random value ensuring that the request is unique, so it cannot be replicated by any other unknown party. This string is always 16 bytes long and must be represented as a 32-character hexadecimal value.

All the above-mentioned elements are concatenated into a single line string and then assigned to X-WSSE HTTP header.

"Username Token Username=\"${username}\", PasswordDigest=\"${passwordDigest}\", Nonce=\"${nonce}\", Created=\"${created}\"";

Sample Code:



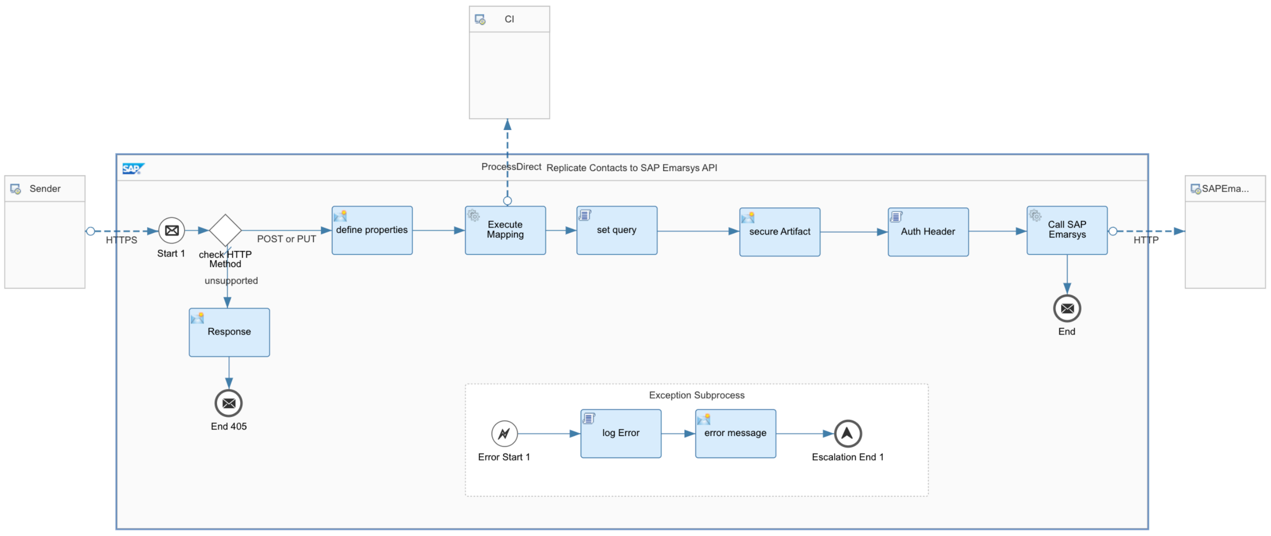
```
def Message XWSSEHeader(Message message) {

    // X-WSSE: UsernameToken Username="name", PasswordDigest="digest", Created="timestamp", Nonce="nonce"
    //
    //  * Username- The username that the user enters (the TypePad username).
    //  * Nonce. A secure token generated anew for each HTTP request.
    //  * Created. The ISO-8601 timestamp marking when Nonce was created.
    //  * PasswordDigest. A SHA-1 digest of the Nonce, Created timestamp, and the password
    //    that the user supplies, base64-encoded. In other words, this should be calculated
    //    as: base64(sha1(Nonce . Created . Password))

    //nonce - Generate a random 16-byte nonce in the 32-character hexadecimal format.
    byte[] nonceBytes = new byte[16];
    new Random().nextBytes(nonceBytes);
    def nonceHex = nonceBytes.encodeHex().toString();

    //Timestamp - Get the current timestamp in ISO 8601 format.
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ssZ");
    sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
    def timestamp = sdf.format(new Date())

    //Secret - Get Secret from Security Artifact
    //Properties
    map = message.getProperties();
    artifactAlias = map.get("securityArtifact");
    //Get the Service Instance of the SecureStoreService API
    def service = ITApiFactory.getService(SecureStoreService.class, null);
    //Read the credential of a given alias
    def credential = service.getUserCredential(artifactAlias);
    //Read the User Name or password
    def APIUser = credential.getUsername();
    def secret = credential.getPassword();

    //Password Digest
    def passwordDigest = nonceHex + timestamp + secret;

    MessageDigest passwordDigestAlg = MessageDigest.getInstance("SHA-1");
    byte[] passwordDigestSHA1 = passwordDigestAlg.digest(passwordDigest.getBytes("UTF-8"));
    def passwordDigestSHA1Res = new BigInteger(1, passwordDigestSHA1).toString(16);
    def passwordDigestsha1Base64 = passwordDigestSHA1Res.bytes.encodeBase64().toString();

    message.setProperty("passwordDigest", passwordDigestsha1Base64);


    //X-WSSE Header
    def headerValue = "UsernameToken Username=\"" + APIUser + "\",PasswordDigest=\"" +
passwordDigestsha1Base64 + "\",Created=\"" + timestamp + "\",Nonce=\"" + nonceHex + "\""
    def map = message.getHeaders();
    message.setHeader("X-WSSE", headerValue);

    return message
}
```

Creating the Contact:

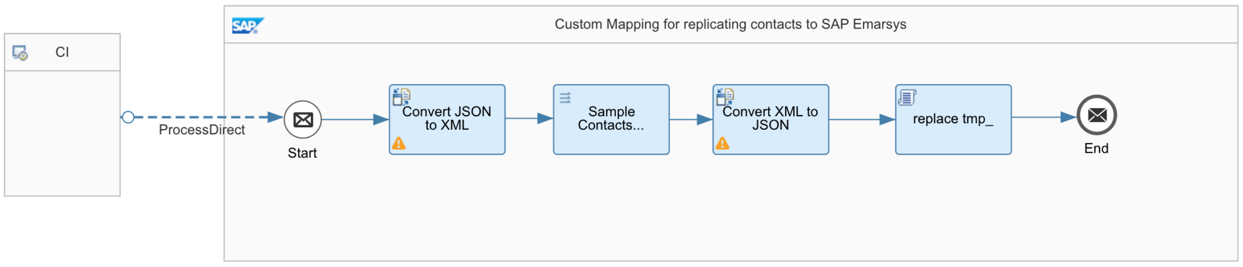
After creating the authentication token, the Integration Flow creates a contact using REST API in JSON format: https://api.emarsys.net/api/v2/contact.

Once contact is created response is sent back to the source system

iFlow - Replicate Contacts to SAP Emarsys API handles the authentication mechanism as called below Sub Flow - **Custom Mapping for replicating Contacts to SAP Emarsys** via Process Direct adapter for creating/overriding the mapping as per requirement.



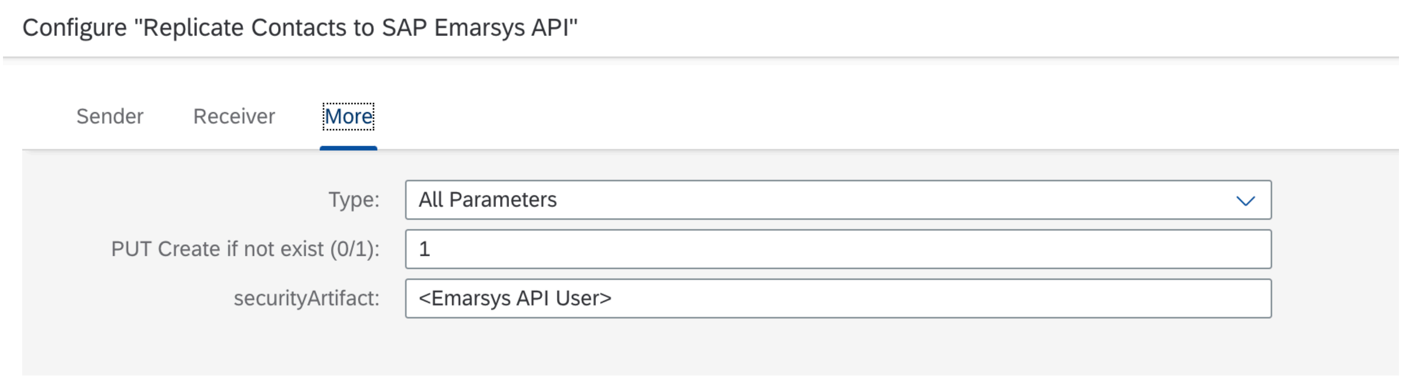
iFlow 1: Replicate Contacts to SAP Emarsys API



iFlow 1: Custom Mapping for replicating Contacts to SAP Emarsys

3 Configuration steps on SAP Cloud Integration

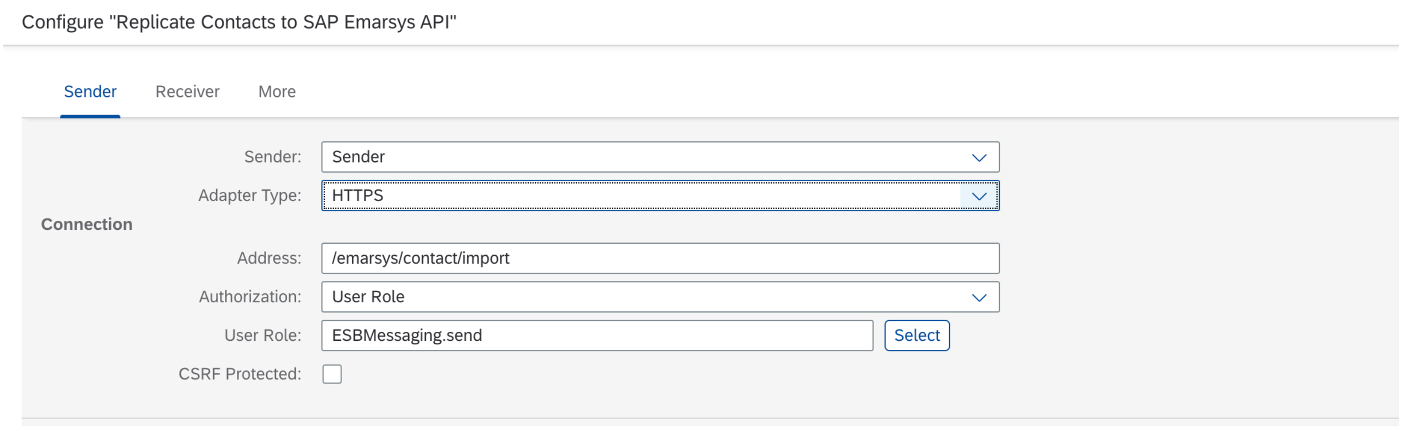
To test the Integration Flow, the following parameters should be configured:


Configure "Replicate Contacts to SAP Emarsys API"

Type:	All Parameters
PUT Create if not exist (0/1):	1
securityArtifact:	<Emarsys API User>

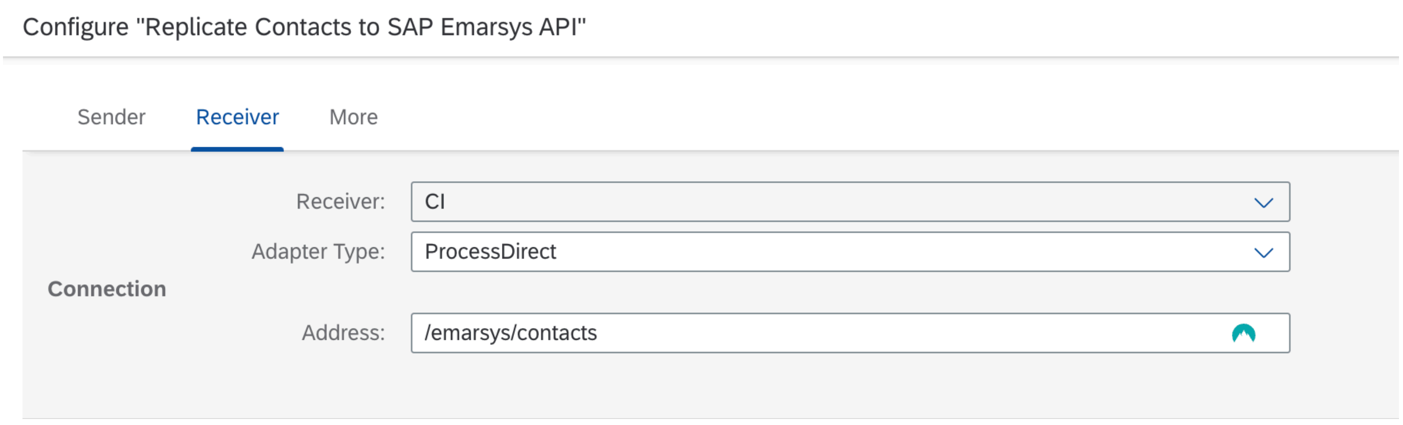
Property	Value
securityArtifact	Credential deployment w/ username & secret of the SAP Emarsys API user
PUT Create if not exist (0/1)	If set to 1, creates a new contact if it does not exist yet

Adapter Configurations


Configure "Replicate Contacts to SAP Emarsys API"

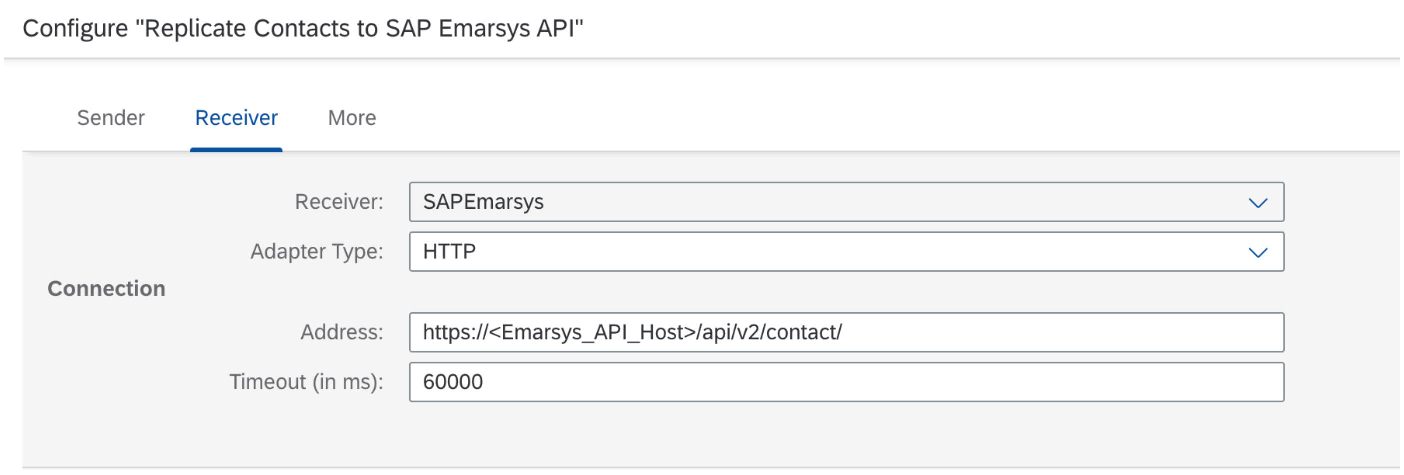
Sender:	Sender
Adapter Type:	HTTPS
Address:	/emarsys/contact/import
Authorization:	User Role
User Role:	ESBMessaging.send
CSRF Protected:	☐

Fig - AdapterConfig.1(Replicate Contacts to SAP Emarsys API)


Configure "Replicate Contacts to SAP Emarsys API"

Receiver:	CI
Adapter Type:	ProcessDirect
Address:	/emarsys/contacts

Fig - AdapterConfig.2(Replicate Contacts to SAP Emarsys API)

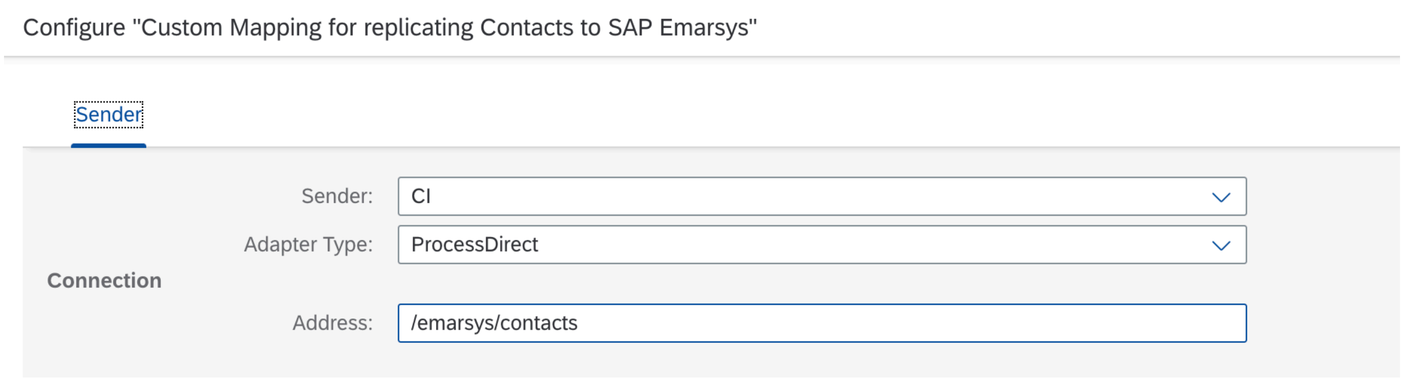

Configure "Replicate Contacts to SAP Emarsys API"

Connection

Receiver:	SAPEmarsys
Adapter Type:	HTTP
Address:	https://<Emarsys_API_Host>/api/v2/contact/
Timeout (in ms):	60000

Fig - AdapterConfig.3(Replicate Contacts to SAP Emarsys API)

Property	Value
Address Fig - AdapterConfig.1(Replicate Contacts to SAP Emarsys API)	URI, based on this the HTTP endpoint for Cloud Integration artifact gets generated and needs to be used by the consumer of this API
Address Fig - AdapterConfig.2(Replicate Contacts to SAP Emarsys API)	Same value as maintained in iFlow - Custom Mapping for replicating Contacts to SAP Emarsys
Address Fig - AdapterConfig.3(Replicate Contacts to SAP Emarsys API)	Emarsys API Endpoint


Configure "Custom Mapping for replicating Contacts to SAP Emarsys"

Connection

Sender:	CI
Adapter Type:	ProcessDirect
Address:	/emarsys/contacts

Fig - AdapterConfig.4(Custom Mapping for replicating Contacts to SAP Emarsys)

Property	Value
Address	Endpoint (same needs to be used in the caller flow)

4 Resources

Authentication guide

Emarsys API Documentation

API Postman Collection