Just-in-Time Provisioning

> *Initially created by Shalu Gangwar*

Overview

With Just-in-Time provisioning, you can use a SAML assertion to create regular and portal users on the fly the first time they try to log in. This eliminates the need to create user accounts in advance
For example, if you recently added an employee to your organization, you don't need to manually create the user in Salesforce. When they log in with single sign-on, their account is automatically created for them, eliminating the time and effort with on-boarding the account. Just-in-Time provisioning works with your SAML identity provider to pass the correct user information to Salesforce in a SAML 2.0 assertion. You can both create and modify accounts this way.

Benefits of Just-in-Time Provisioning

- **Reduced Administrative Costs:** Provisioning over SAML allows customers to create accounts on-demand, as part of the single sign-on process. This greatly simplifies the integration work required in scenarios where users need to be dynamically provisioned, by combining the provisioning and single sign-on processes into a single message.
- **Increased User Adoption:** Users only need to memorize a single password to access both their main site and Salesforce. Users are more likely to use your Salesforce application on a regular basis.
- **Increased Security:** Any password policies that you have established for your corporate network are also in effect for Salesforce. In addition, sending an authentication credential that is only valid for a single use can increase security for users who have access to sensitive data.

Field Requirements for the SAML Assertion

To correctly identify which object to create in Salesforce, you must use the User. prefix for all fields passed in the SAML assertion. In this example, the User. prefix has been added to the Username field name.

```
<saml:Attribute
   Name="User.Username"
   NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
      <saml:AttributeValue xsi:type="xs:anyType">testuser@123.org</saml:AttributeValue
</saml:Attribute>
```

The following User fields are required:
Email
LastName
ProfileId
Username (insert only)
The following User fields are supported.
FirstName
CommunityNickname
FederationIdentifier
TimeZoneSidKey

This value is the Time Zone field on the User object.
LanguageLocaleKey
This value is the Language field on the User object.
LocaleSidKey
This value is the Locale field on the User object.
EmailEncodingKey
This value is the Email Encoding field on the User object.
DefaultCurrencyIsoCode
Role
Alias
Title
Phone
CompanyName
This value is the Company field on the User object.
Active
This value is the Active field on the User object and User.isActive in the API.
AboutMe
Street
This value is part of the Address compound field on the User object and User.Street in the API.
State
This value is part of the Address compound field on the User object and User.State in the API.
City
This value is part of the Address compound field on the User object and User.City in the API.
Zip
This value is part of the Address compound field on the User object and User.PostalCode in the API.
Country
This value is part of the Address compound field on the User object and User.Country in the API.
ReceivesAdminInfoEmails
ForecastEnabled
This value is the Allow Forecasting checkbox on the User object.
CallCenter
This value is the User.CallCenterId in the API.
Manager
MobilePhone
DelegatedApproverId
Department
Division
EmployeeNumber
Extension
Fax
ReceivesInfoEmails

Other field requirements:

- Only text type custom fields are supported.
- Only the `insert` and `update` functions are supported for custom fields.
- When using the API for user creation, you can pass the new username into the User.Username field. You can also specify the User.FederationIdentifier if it is present. However, the Username and FederationIdentifier fields can't be updated with API.

Just-in-Time Provisioning and SAML Assertion Fields for Portals

With Just-in-Time (JIT) provisioning for portals, you can use a SAML assertion to create customer and partner portal users on the fly the first time they try to log in. This eliminates the need to create user accounts in advance. Because JIT uses SAML to communicate, your organization must have SAML-based single sign-on enabled.

CREATING PORTAL USERS

The `Portal ID` and `Organization ID` must be specified as part of the SAML assertion. You can find both of these on the company information page for the organization or portal. Because you can also provision regular users, the `Portal ID` is used to distinguish between a regular and portal JIT provisioning request. If no `Portal ID` is specified, then the request is treated as a JIT request for regular platform user. Here are the requirements for a creating a portal user.

- You must specify a `Federation ID`. If the ID belongs to an existing user account, the user account is updated. In case of an inactive user account, the user account is updated, but left inactive unless `User.IsActive` in the JIT assertion is set to true. If there is no user account with that `Federation ID`, the system creates a new user.
- If the portal isn't self-registration enabled and a default new user profile and role aren't specified, the `User.ProfileId` field must contain a valid profile name or ID associated with the portal. In addition, the `User.PortalRole` field must contain a valid portal role name or ID. Use Worker for all portal users.

CREATING AND MODIFYING ACCOUNTS

Create or modify an account by specifying a valid `Account ID` or both the `Account.AccountNumber` and `Account.Name`.

- Matching is based on `Account.AccountNumber`. If multiple accounts are found, an error is displayed. Otherwise, the account is updated.
- If no matching account is found, one is created.
- You must specify the `Account.Owner` in the SAML assertion and ensure that the field level security for the `Account.AccountNumber` field is set to visible for this owner's profile.

CREATING AND MODIFYING CONTACTS

Create or modify a contact by specifying the a valid Contact ID in `User.Contact` or both the `Contact.Email` and `Contact.LastName`.

- Matching is based on `Contact.Email`. If multiple contacts are found, an error is displayed. Otherwise, the contact is updated.
- If no matching contact is found, one is created.

SUPPORTED FIELDS FOR THE PORTAL SAML ASSERTION

To correctly identify which object to create in Salesforce, you must use a prefix. In the SAML assertion, use the `Account` prefix for all fields in the Account schema (for example Account.AccountId) and `Contact` prefix for all fields in the Contact schema. In this example, the `Contact` prefix has been added to the `Email` field name.

```
<saml:Attribute
```

```
    Name="Contact.Email"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
       <saml:AttributeValue xsi:type="xs:anyType">testuser@123.org</saml:AttributeValue
 </saml:Attribute>
```

In addition to the standard User attributes supported for regular SAML JIT users, these Account attributes are also supported. For example, specifying an `Account.Phone` attribute in the assertion will update the account's Phone field on the corresponding Account object.

Name
AccountNumber
BillingCity
BillingCountry
BillingPostalCode
BillingState
BillingStreet
Owner
The Owner field on the Account object is Account.OwnerId in the API.
AnnualRevenue
Description
NumberOfEmployees
Fax
Industry
Ownership
Phone
Rating
ShippingAddress
The Shipping Address field is a compound field.
ShippingCity
ShippingCountry
ShippingPostalCode
ShippingState
ShippingStreet
Sic
TickerSymbol
Website
These Contact attributes are supported.
Account
This value is the Account Name field on the Contact object and Account.Name in the API.
Email
FirstName
LastName
Phone
CanAllowPortalSelfReg
AssistantName
AssistantPhone
Birthdate
Owner
This value is the Contact Owner field on the Contact object and Contact.OwnerId in the API.
Department
Description
DoNotCall

HasOptedOutOfEmail
Fax
HasOptedOutOfFax
HomePhone
LastCUUpdatetDate
This value is the Last Modified By field on the Contact object and Contact.LastModifiedDate in the API.
LeadSource
MailingAddress
The Mailing Address field is a compound field.
MailingCity
MailingCountry
MailingPostalCode
MailingState
MailingStreet
MobilePhone
Salutation
OtherAddress
The Other Address field is a compound field.
OtherCity
OtherCountry
OtherPostalCode
OtherState
OtherStreet
OtherPhone
Title
These additional User attributes are supported for portal users.
AccountId
ContactId
PortalRole
Use `Worker` for all portal users.

Guidelines for Just-in-Time (JIT) provisioning

These steps are applicable to standard JIT Provisioning for SAML SSO. For custom configurations where Apex code implements the SamlJitHandler Interface, step 1 applies by default and steps 2-7 can be handled by the Apex code.

1. Salesforce attempts to match the Federated ID in the subject of the SAML assertion (e.g. 12345) to the FederationIdentifier field of a existing user record.
2. If a **matching user record is found**, JIT provisioning uses the attributes to **Update** the fields specified in the attributes.
3. If a user with a **matching user record isn't found**, then Salesforce searches the contact for a match based on Contact ID (User.Contact) or email (Contact.Email).
Contact.Email and Contact.LastName are both required properties when User.Contact is not specified. But matching is only based on Contact.Email when both properties exist.
4. If a **matching contact record is found**, JIT provisioning uses the attributes to **Update** the contact fields specified in the attributes and then **Inserts** the new User record
5. If a **matching contact record isn't found**, then Salesforce searches for the Accounts for a match based on Contact.Account or Account.AccountNumber.
Account.AccountNumber and Account.Name are both required properties when Contact.Account is not specified. But matching is only based on Account.AccountNumber when both properties exist.
6. If a **matching account record is found**, JIT provision **Inserts** a new contact record and **Inserts** a new User record based on the attributes provided.
7. If a **matching account record isn't found**, JIT provision **Inserts** a new account record, **Inserts** a new contact record, and

Inserts a new User record based on the attributes provided.
Please refer the below examples as well and their results in different scenarios.

	A	B
1	**Example 1:** *User record exists:* Contact fields (eg LastName, Email) and User fields (eg Email, LastName, etc) are **updated**. *User record does not exist, but contact record does:* SFDC matches contact based on LastName and Email.New user record **inserted**. *User record and contact record do not exist:* New contact and user records are **inserted**.	**Contact.Account**=00130000011Qx7i; **Contact.LastName**=PortalUser; **Contact.Email**=testPortal1@test.com; User.ProfileId=00e30000000wAhX; User.PortalRole=Worker; User.Username=testPortal1@test.com; User.Email=testPortal1@test.com; User.LastName=PortalUser;

In the above example, we can create new portal users and contacts given a hardcoded Account id. So, what if you don't know the id for the Account? Then you can search for it using the Account's Name and Account Number.

	A	B
1	**Example 2:** *User record exists:* Account fields (eg Name, Owner), Contact fields (eg LastName, Email), and User fields (eg Email, LastName, etc) are **updated**. *User record does not exist, but contact record does:* SFDC matches contact based on LastName and Email. Account fields (eg Name, Owner) are**updated**. New user record **inserted**. *User record and contact record do not exist:* SFDC matches account based on AccountNumber and Name. New contact and user records are **inserted**. *User record, contact record, and account record do not exist:* New account, contact, and user records are **inserted**.	**Account.AccountNumber**=9999; **Account.Name**=TestCompany; **Account.Owner**=005J0000000yvS0; Contact.LastName=PortalUser2; Contact.Email=testPortal2@test.com; User.ProfileId=00eU0000000ZLQe; User.PortalRole=Worker; User.Username=testPortal2@test.com; User.Email=testPortal2@test.com; User.LastName=PortalUser2;

Note that in the above example, an owner Id is required to create a new account record.
Now what if you don't want JIT provisioning to create new account records or you don't have Account details in your IDP. Then you can leave out the account attributes altogether.

	A	B
1	**Example 3:** *User record exists:* Contact fields (eg LastName, Email), and User fields (eg Email, LastName, etc) are **updated**. *User record does not exist, but contact record does:* SFDC matches contact based on LastName and Email. New user record **inserted**. *User record and contact record do not exist:* SFDC matches account based on AccountNumber and Name. New contact and user records are **inserted**. *User record, contact record, and account record do not exist:* Process does not insert any new records because an account is needed to create a contact and portal user record.	Contact.LastName=PortalUser3; Contact.Email=testPortal3@test.com; User.ProfileId=00e30000000wAhX; User.PortalRole=Worker; User.Username=testPortal3@test.com; User.Email=testPortal3@test.com; User.LastName=PortalUser3;

The examples above show only the minimum attributes required. However, you can include additional attributes which correspond to standard fields on the User, Contact, and Account records.

SamlJitHandler Interface

Use this interface to control and customize Just-in-Time user provisioning logic during SAML single sign-on.

NAMESPACE

Auth

Usage

To use custom logic for user provisioning during SAML single sign-on, you must create a class that implements Auth.SamlJitHandler. This allows you to incorporate organization-specific logic (such as populating custom fields) when users log in to Salesforce with single sign-on. Keep in mind that your class must perform the logic of creating and updating user data as appropriate, including any associated account and contact records.
In Salesforce, you specify your class that implements this interface in the `SAML JIT Handler` field in SAML Single Sign-On Settings. Make sure that the user you specify to run the class has “Manage Users” permission.

SAMLJITHANDLER METHODS

THE FOLLOWING ARE METHODS FOR SAMLJITHANDLER.

- **createUser(samlSsoProviderId, communityId, portalId, federationId, attributes, assertion)**
  Returns a User object using the specified Federation ID. The User object corresponds to the user information and may be a new user that hasn’t t been inserted in the database or may represent an existing user record in the database.
- **updateUser(userId, samlSsoProviderId, communityId, portalId, federationId, attributes, assertion)**
  Updates the specified user’s information. This method is called if the user has logged in before with SAML single sign-on and then logs in again, or if your application is using the `Existing User Linking URL`.

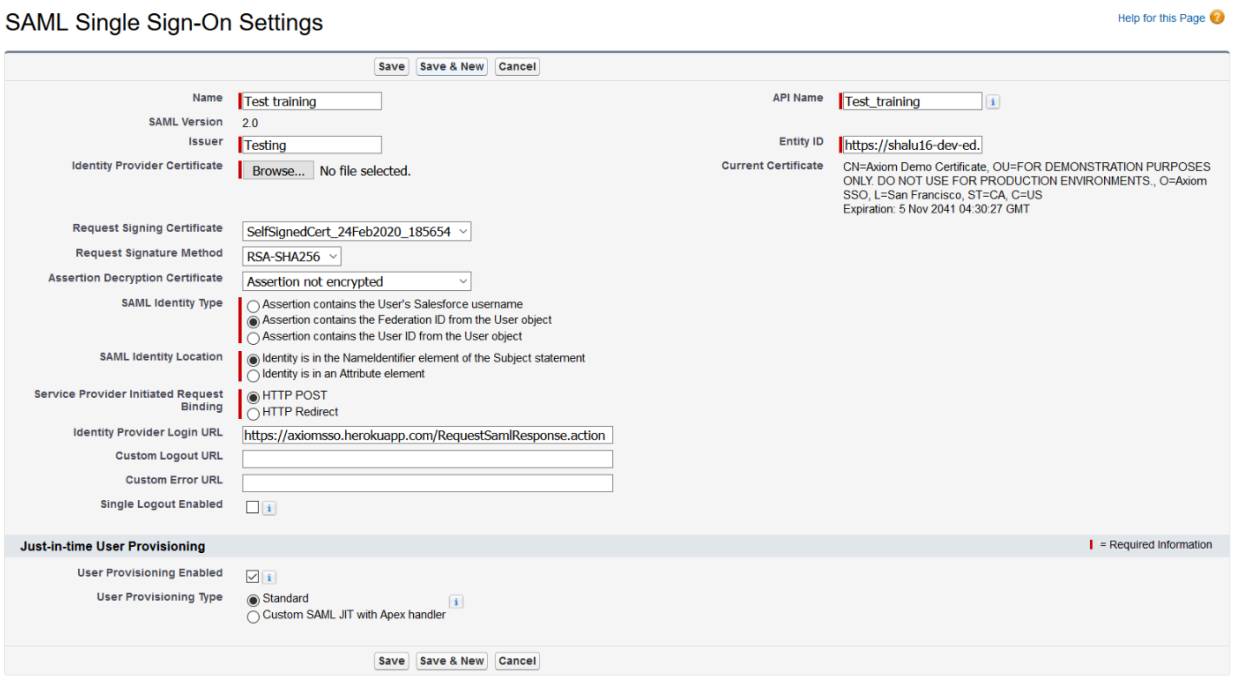
SamlJitHandler Example Implementation

```
global class StandardUserHandler implements Auth.SamlJitHandler {
    private class JitException extends Exception{}
    private void handleUser(boolean create, User u, Map<String, String> attributes,
        String federationIdentifier, boolean isStandard) {
        if(create && attributes.containsKey('User.Username')) {
            u.Username = attributes.get('User.Username');
        }
        if(create) {
            if(attributes.containsKey('User.FederationIdentifier')) {
                u.FederationIdentifier = attributes.get('User.FederationIdentifier');
            } else {
                u.FederationIdentifier = federationIdentifier;
            }
        }
        if(attributes.containsKey('User.ProfileId')) {
            String profileId = attributes.get('User.ProfileId');
            Profile p = [SELECT Id FROM Profile WHERE Id=:profileId];
            u.ProfileId = p.Id;
        }
        if(attributes.containsKey('User.UserRoleId')) {
            String userRole = attributes.get('User.UserRoleId');
            UserRole r = [SELECT Id FROM UserRole WHERE Id=:userRole];
            u.UserRoleId = r.Id;
        }
        if(attributes.containsKey('User.Phone')) {
```

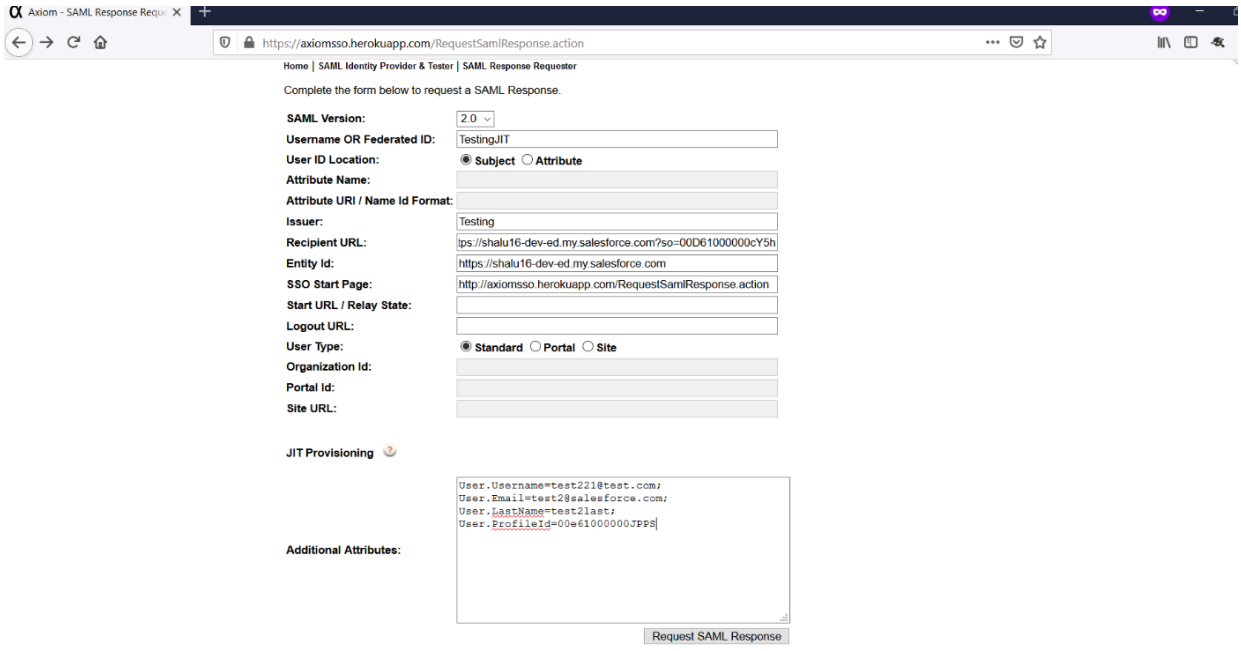
```
            u.Phone = attributes.get('User.Phone');
        }
        if(attributes.containsKey('User.Email')) {
            u.Email = attributes.get('User.Email');
        }

    //More attributes here - removed for length

        //Handle custom fields here

        if(!create) {
            update(u);
        }
    }

    private void handleJit(boolean create, User u, Id samlSsoProviderId, Id communityI
        String federationIdentifier, Map<String, String> attributes, String assertion)
        if(communityId != null || portalId != null) {
            String account = handleAccount(create, u, attributes);
            handleContact(create, account, u, attributes);
            handleUser(create, u, attributes, federationIdentifier, false);
        } else {
            handleUser(create, u, attributes, federationIdentifier, true);
        }
    }

    global User createUser(Id samlSsoProviderId, Id communityId, Id portalId,
        String federationIdentifier, Map<String, String> attributes, String assertion)
        User u = new User();
        handleJit(true, u, samlSsoProviderId, communityId, portalId,
            federationIdentifier, attributes, assertion);
        return u;
    }

    global void updateUser(Id userId, Id samlSsoProviderId, Id communityId, Id portalI
        String federationIdentifier, Map<String, String> attributes, String assertion)
        User u = [SELECT Id FROM User WHERE Id=:userId];
        handleJit(false, u, samlSsoProviderId, communityId, portalId,
            federationIdentifier, attributes, assertion);
    }
}
```

Steps to configure JIT

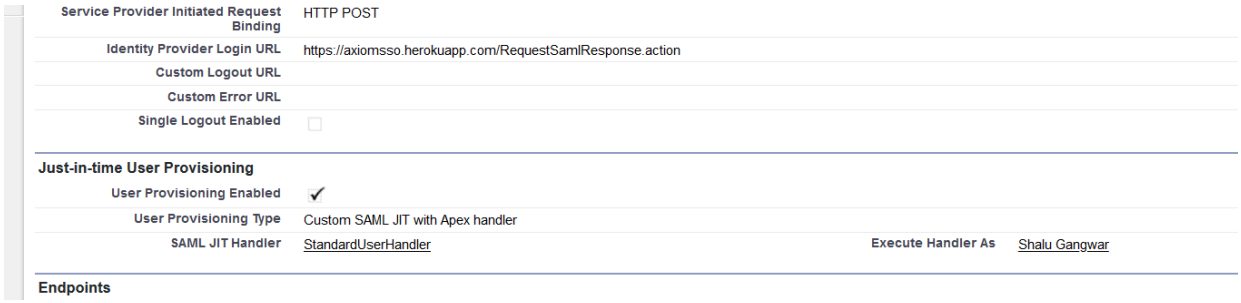
1. Enable JIT provisioning from Single sign on Settings.



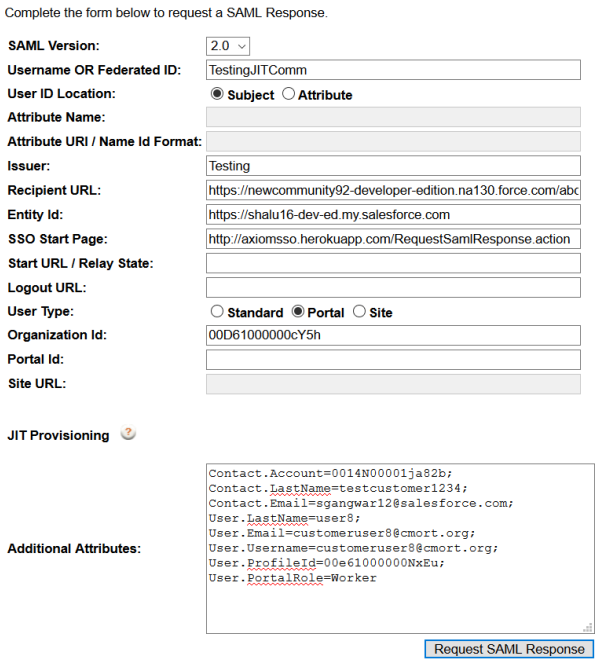
2. You have to select “User Provisioning Type” as either “standard” or “Custom SAML JIT with Apex handler“ as per your business requirement.
3. If you select “Standard” then you can create the users by passing the field values from IDP which are required for user creation.



4. Once you Click on “Request SAMl Response” you and then “Login” you will see the user created.
5. However if you select custom JIT then you need to select the apex class in “SAML JIT Handler” field and the person on which context the class will run will be selected in “Execute Handler As” field.



6. To create community user you just need to pass some extra attributes like contact and account fields :-

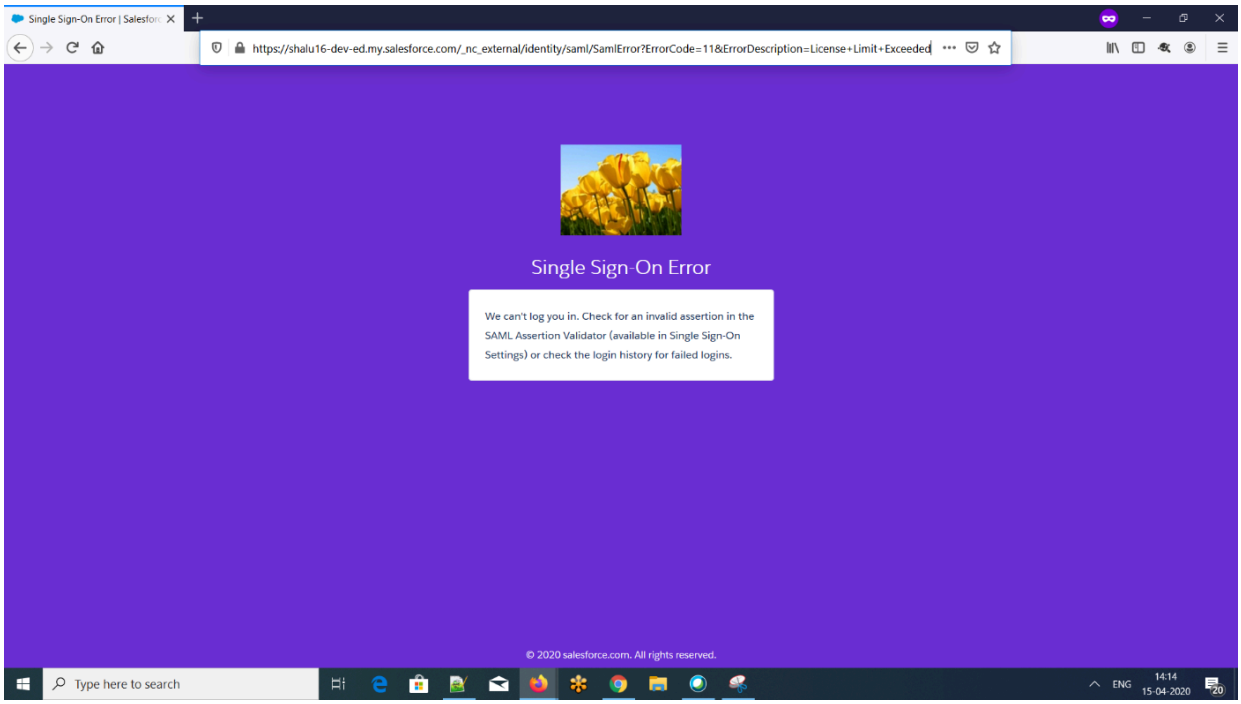


Just-in-Time Provisioning Errors

SAML errors are returned in the URL parameter, for example:

```
http://login.salesforce.com/identity/jit/saml-error.jsp?
ErrorCode=5&ErrorDescription=Unable+to+create+user&ErrorDetails=
INVALID_OR_NULL_FOR_RESTRICTED_PICKLIST+TimeZoneSidKey
```

NOTE
Salesforce redirects the user to a custom error URL if one is specified in your SAML configuration



Error Messages

	CODE	DESCRIPTION	ERROR DETAILS
1	1	Missing Federation Identifier	MISSING_FEDERATION_ID
2	2	Mis-matched Federation Identifier	MISMATCH_FEDERATION_ID
3	3	Invalid organization ID	INVALID_ORG_ID
4	4	Unable to acquire lock	USER_CREATION_FAILED_ON_UROG
5	5	Unable to create user	USER_CREATION_API_ERROR
6	6	Unable to establish admin context	ADMIN_CONTEXT_NOT_ESTABLISHED
7	8	Unrecognized custom field	UNRECOGNIZED_CUSTOM_FIELD
8	9	Unrecognized standard field	UNRECOGNIZED_STANDARD_FIELD
9	11	License limit exceeded	LICENSE_LIMIT_EXCEEDED
10	12	Federation ID and username do not match	MISMATCH_FEDERATION_ID_AND_USERNAME_ATTRS
11	13	Unsupported provision API version	UNSUPPORTED_VERSION
12	14	Username change isn't allowed	USER_NAME_CHANGE_NOT_ALLOWED
13	15	Custom field type isn't supported	UNSUPPORTED_CUSTOM_FIELD_TYPE
14	16	Unable to map a unique profile ID for the given profile name	PROFILE_NAME_LOOKUP_ERROR
15	17	Unable to map a unique role ID for the given role name	ROLE_NAME_LOOKUP_ERROR
16	18	Invalid account	INVALID_ACCOUNT_ID
17	19	Missing account name	MISSING_ACCOUNT_NAME
18	20	Missing account number	MISSING_ACCOUNT_NUMBER
19	22	Unable to create account	ACCOUNT_CREATION_API_ERROR
20	23	Invalid contact	INVALID_CONTACT
21	24	Missing contact email	MISSING_CONTACT_EMAIL
22	25	Missing contact last name	MISSING_CONTACT_LAST_NAME
23	26	Unable to create contact	CONTACT_CREATION_API_ERROR
24	27	Multiple matching contacts found	MULTIPLE_CONTACTS_FOUND
25	28	Multiple matching accounts found	MULTIPLE_ACCOUNTS_FOUND
26	30	Invalid account owner	INVALID_ACCOUNT_OWNER
27	31	Invalid portal profile	INVALID_PORTAL_PROFILE
28	32	Account change is not allowed	ACCOUNT_CHANGE_NOT_ALLOWED
29	33	Unable to update account	ACCOUNT_UPDATE_FAILED
30	34	Unable to update contact	CONTACT_UPDATE_FAILED
31	35	Invalid standard account field value	INVALID_STANDARD_ACCOUNT_FIELD_VALUE
32	36	Contact change not allowed	CONTACT_CHANGE_NOT_ALLOWED
33	37	Invalid portal role	INVALID_PORTAL_ROLE
34	38	Unable to update portal role	CANNOT_UPDATE_PORTAL_ROLE
35	39	Invalid SAML JIT Handler class	INVALID_JIT_HANDLER
36	40	Invalid execution user	INVALID_EXECUTION_USER
37	41	Execution error	APEX_EXECUTION_ERROR
38	42	Updating a contact with Person Account isn’t supported	UNSUPPORTED_CONTACT_PERSONACCT_UPDATE

OTHER WAYS TO TROUBLESHOOT THE ISSUE:-

Debug logs
Saml Response

SAMPLE VALUES

```
Insert User
```

```
=========================
User.Username=test221@test.com;
User.Email=test2@salesforce.com;
User.LastName=test2last;
User.ProfileId=00e61000000JPPI


Update User(Federation ID :- TestingJIT) :-
============
User.Username=test123ww67@gmail.com;
User.Email=test123ww67@gmail.com;
User.LastName=test17;
User.ProfileId=00e61000000JPPS;
User.Title=test

Portal user Creation :-
==============
Organisaion ID ;- 00D61000000cY5h
========================
Contact.Account=0014N00001ja82b;
Contact.LastName=testcustomer1234;
Contact.Email=sgangwar12@salesforce.com;
User.LastName=user8;
User.Email=customeruser8@cmort.org;
User.Username=customeruser8@cmort.org;
User.ProfileId=00e61000000NxEu;
User.PortalRole=Worker
```

RELATED CASES :-

26152399
19037137
17629254
18819914
13626682

Portal User creation and Creation of portal JIT User:
Portal User SSO.

REFERENCES:

Just-in-Time Provisioning for SAML
Understanding JIT Provisioning
SamlJitHandler Interface