



WebEx Connect: User Provisioning and SSO

Developer Technical Note



WebEx Communications Inc.
3979 Freedom Circle, Santa Clara, CA 95054

Corp: +1-408-435-7000 Sales: +1.877.50.WEBEX

www.webex.com

Introduction

The WebEx Connect™ Integration Platform offers a flexible and robust means for integrating customer and partner applications. The central integration mechanism is the construction of WebEx Connect Application Modules that surface application functionality and content into the WebEx Connect Client.

Many Partner applications desire to provide a seamless user experience as Widgets interact with the Partner application or service. This Tech Note provides guidance for automating, or semi-automating, the provisioning of users on the Partner service, and also discusses mechanisms for Single Sign-On (SSO) behavior.

This Tech Note assumes the reader has a reasonable familiarity with the overall WebEx Connect Platform, as can be found in the *WebEx Connect Technical Overview* white paper available on the Connect Developer Community site, under Resources. <http://community.webex.com/connect>.

Delegated Login

This section describes a special approach to logging into the WebEx Connect Platform Service layer that is only available to partners. It forms the underpinning of many of the approaches to provisioning and SSO described later in this Tech Note. It provides a secure way for a partner application to verify and trust access to the application without invoking passwords. Thus, it is a good way to relieve the user having to remember multiple passwords. The only challenge in employing this mechanism is that it requires server-side application logic changes.

Delegated Login operates as follows:

- ❑ The partner, as part of their participation in the WebEx Connect ecosystem and Marketplace, is provisioned with a Connect “Org”. This serves development purposes, and is key to the delegated login mechanism. The Connect platform “knows” this Org is a Partner Org.
- ❑ The partner Org Admin creates a special, “phantom” user account, called something like “server@partner.com”. This account is used to access the Connect Platform Service API (CPS) for server-to-server calls. This account is not required, but is highly recommended because it separates out other user account activities from the server activities. This is especially useful during development and debugging tasks.
- ❑ The partner Application Module, upon start-up, can then contact its back-end application to either provision or SSO the user.
 - The Module obtains the user ID from the Application Framework API, and also obtains a special encrypted session token as follows using `WbxCpsStore()` using the `RequestProxyCred` command as in this example:

```
var cpsStore = mashkit.data.WbxCpsStore.getInstance();

cpsStore.fetch({query:{
  cmd:"execute", task: "RequestProxyCred",
  rootItem:"nullSessionID"},
  scope: this,
  onComplete: function(items, request) {
    if(items.length > 0){
      var item = items[0];
      var proxycred =
        dojox.data.dom.textContent(item.element);
      console.log("proxycred: " + proxycred);
    }
  }
});
```

- The App Module passes this encrypted cred to the application server through `WbxClientAPIStore.httpbind()` or `SoapStore()`, depending on the service protocol being used.
- The application server then calls the CPS API for delegated login. The CPS call uses the “server” user credentials for authentication, and passes the encrypted token in as well, which captures the user ID of the calling module above.

```
https://swapi.webexconnect.com/wbxconnect/op.do?cmd=execute
&task=PartnerLogin&username=xxxx&password=xxxxx&isp=WBX&xml
=<input><tokenID>[proxycréd]</tokenID></input>
```

```
// where:
-- cmd = execute

-- task = PartnerLogin

-- username and password are for the partner “server” user,
not the module user.

-- xml is the xml form that has the <tokenID> contents
passed in by App Module.
```

- CPS verifies the “server” user belongs to a bona-fide partner organization, and decrypts the token. If all is well, CPS returns a success status, and also returns a new (cloned) session token for the user ID passed in. This first provides validation that the user of the module is to be trusted, and second allows the server application to make subsequent calls on behalf of that user if needed. For the purposes of provisioning and SSO, all that is needed is to validate the user/token with the login call.

We will see in the discussions below how this mechanism can be used and trusted for provisioning and SSO operations.

User Provisioning

Typically, the partner application or service is a fee-based offering, and the end customer or organization purchases access to the service. Whether for a fee, or for free, the ideal user experience is to place the partner’s App Module into a Space, and have it work “out of the box”, especially when the access to the application is granted by the user’s parent organization. Thus, the user should only have to perform minimal registration procedures, or none at all.

Additionally, many partner applications are offered with a free trial to increase adoption. This is a prime case for automated provisioning. There are a couple of approaches to these scenarios that partners may adopt. The level of automation is up to the partner, allowing the offering to be matched with the overall business model of the solution.

Pre-Provisioning

In this scenario, the customer purchases the offering from the partner, and at that time provides user names (at a minimum) to the partner to provision user accounts. The partner might also issue passwords at this time, depending on the desired approach to SSO. Given that user accounts are pre-provisioned, it then comes down to the sign-on process. Any of the mechanisms described below for SSO are possible.

There are obvious disadvantages to pre-provisioning. First, it creates an administrative burden for both the customer and the partner. Second, as additional users are added later to the service, they must each be individually provisioned. The advantage to pre-provisioning is that there is tight control over user accounts and access to the application or service, and no special application logic changes are required such as for invoking delegated login.

Self Provisioning

In this scenario, the App Module (upon initial start-up) prompts or otherwise guides the user through the provisioning process. The Module could obtain the user ID from the framework, or prompt the user for an arbitrary ID. The Module would also ask the user to establish a password as well. Lastly, the Module can obtain contextual information about the user, primarily the user's parent Organization. Using this information, the application or service could decide if that user and Org are entitled to access the service, and provision the account.

For unaffiliated users, there is no parent Org, so the application would have to rely on the user's name and possibly other information in the user's profile to decide if that user should be provisioned.

Auto Provisioning

This scenario is the most desirable from the end user perspective. This is also an ideal method to provide access to free services, or trials. For fee-based services, the issue is how secure the approach is, and what exploits could occur allowing unauthorized users to obtain accounts without paying for them. The use of the Delegated Login mechanism can minimize these concerns.

Free, or Trial Account Provisioning

To automate provisioning where there is no requirement for an initial purchase of a service, the App Module merely obtains user information from the Application framework, and calls the back-end service to provision the user. This information is typically the user ID, Org, and any other information desired from the user's profile.

Fee-based Provisioning

Automatic provisioning for a fee-based service requires not only a secure validation of the user, but also additional information to allow proper billing. This typically can only be employed to auto-provision users who belong to an Org, where that Org has entered into some kind of service relationship with the partner.

To start, the Module should use the Delegated Login mechanism described in the section above to validate the user is a bona-fide Connect user. It should also obtain the Org of that user, and pass this information to the provisioning service. From there, the application logic can inform billing/reporting systems that a new user for the given Org is now active.

The best approach is to provision the user's account using their user ID as the key. Then, SSO can be employed using the delegated login mechanism. Thus, the end result for the user is that they just put the Module in their Space, and everything is automatic.

Single Sign-On

There are multiple approaches to SSO available in the Connect Application framework. The approach chosen depends on what degree of "seamlessness" is desired, and to what degree changes to the application server logic are possible.

Pseudo SSO

This approach involves initially prompting the user for credentials, and after that, SSO is automatic for subsequent accesses to the Module. When the user invokes the partner Widget for the first time, the Module can take a couple of approaches on initial start-up:

- The Module can prompt the user to enter the required user ID and password. These would be declared as sensitive parameters in the Module definition. Once entered, the Module validates the account, and the parameters are then persistently stored locally for subsequent logins. See the Tech Note on Module Parameters for more information about how standard parameter management occurs.
- If the user was provisioned using the user's Connect ID, the Module can obtain the user ID from the container API, The Module then only needs a password, which would be treated as a sensitive parameter as above.

True SSO

To avoid prompting the user for credentials, the Delegated Login mechanism can be used. This approach requires that the user was provisioned using their Connect ID. The password is effectively the encrypted token in the Delegated Login process, and is validated against CPS.

This allows the partner application to trust the user, and given the user ID, can also verify an account for the service exists. This approach is ideal from the perspective of the user experience, but requires specific application logic on the back-end to implement Delegated Login.

Conclusion

The WebEx Connect Application Framework and Platform Service layer enable Partner applications to provide a seamless, automated user experience. By automating provisioning and single sign-on processes, barriers to entry and adoption are minimized. The mechanisms described above not only allow this experience, but do so in a secure fashion.

©2008 WebEx Communications, Inc. WebEx, WebEx MediaTone, and the WebEx logo are registered trademarks of WebEx Communications, Inc. All rights reserved. All other trademarks are the property of their respective owners.