FuseIT™

CONNECTED SOLUTIONS

Simple S4S Code Samples

Table of Contents

1. Master-Child Child-to-Parent Query Example from Contact to Account 2
2. Create a New Salesforce Record 2
3. Retrieve an Existing Salesforce Record by ID 2
4. Retrieve an Existing Salesforce record by query 3
5. Update an Existing Account Using only the ID and the New Value 3
6. Delete an Account 3
7. Accessing Custom Objects and Custom Fields 3
8. Handling Custom Picklists 5
9. Retrieving Picklist Values 6
10. Executing Salesforce Partner API methods 7
11. Attaching Files (doc, PDF, xls, etc.) to a Salesforce Contact Record 8
12. Creating Chatter Records 9
13. Creating Chatter Records with an Attachment 9
14. Searching Salesforce Records 10
15. Improving the Performance of S4S 11
16. Using T4 Templates to Create Strongly Typed Objects 13

FuseIT™ CONNECTED SOLUTIONS

1. Master-Child Child-to-Parent Query Example from Contact to Account

```
SalesforceSession salesforceSession = new SalesforceSession(new LoginDetails("usernam
e@example.com", "salesforcePassword"));
ContactDataSource contactDataSource = new ContactDataSource(salesforceSession);

List<Contact> contacts = contactDataSource.QueryEntities<Contact>(
        Contact.Fields.Id,
        Contact.Fields.FirstName,
        Contact.Fields.Account(Account.Fields.Id, Account.Fields.Name)
);

foreach (Contact contact in contacts)
{
        Account account = contact.Account;
        string accountName = account.Name;
}
```

2. Create a New Salesforce Record

```
Account account = new Account();
account.Name = "A new test Account";

SalesforceSession salesforceSession = new SalesforceSession(new LoginDetails("usernam
e@example.com", "salesforcePassword"));
AccountService accountService = new AccountService(salesforceSession);

var saveResult = accountService.Insert(account);
string newAccountId = null;
if (saveResult.success)
{
    newAccountId = account.Id;
}
```

3. Retrieve an Existing Salesforce Record by ID

```
SalesforceSession salesforceSession = new SalesforceSession(new LoginDetails("usernam
e@example.com", "salesforcePassword"));

AccountService accountService = new AccountService(salesforceSession);

Id accountId = new Id("00190000002G9cF");

Account account = accountService.GetByEntityId(accountId);
string accountName = account.Name;
```

4. Retrieve an Existing Salesforce record by query

```
SalesforceSession salesforceSession = new SalesforceSession(new LoginDetails("usernam
e@example.com", "salesforcePassword"));

AccountService accountService = new AccountService(salesforceSession);
string accountNameToFind = "Zoo Creations";
Account matchingAccount = accountService.GetSingleByFieldEquals("Name", accountNameTo
Find);
// If matchingAccount is null then no matching Accounts were found.
string accountName = matchingAccount.Name;
```

5. Update an Existing Account Using only the ID and the New Value

```
SalesforceSession salesforceSession = new SalesforceSession(new LoginDetails("usernam
e@example.com", "salesforcePassword"));
AccountService accountService = new AccountService(salesforceSession);

Id accountId = (Id)"00190000002G9cF";

Account account = new Account(accountId);

// Set the updated value
account.NumberOfEmployees = 8000;

SaveResult result = accountService.UpdateEntity(account);
Assert.IsTrue(result.success);
Assert.AreEqual(accountId, (Id)result.id);
```

6. Delete an Account

```
// Use the Create example above followed by:
DeleteResult deleteResult = accountService.DeleteEntity(account);
```

7. Accessing Custom Objects and Custom Fields

There are a number of options available for accessing object customizations.

1) Use the T4 templates in Visual Studio to generate a Custom object or core object with custom fields to match your Org

With S4S there is the option of using a T4 template to generate classes customized to your Org within a namespace of your choosing. This is a good option if you will only be dealing with Salesforce Orgs that have the same field setup. The process is reasonably simple and repeatable once setup. The advantage of this approach is that you can use IntelliSense to work with the custom object/field.

2) Use the InternalFields property to access a custom field by name

Each Salesforce entity in S4S will have the InternalFields property that can be used to return any field exposed by the Salesforce API. For example, if there was a custom rich text field on Contact called TestRich__c then it could be accessed by S4S as follows:

```
ContactService contactService = new ContactService(salesforceSession);
Contact contact = contactService.GetByEntityId("0037000000TGNKL");
string richText = contact.InternalFields["TestRich__c"];
```

If the field is not string based you can also use `InternalFields.GetField<T>(string fieldName)` to get other data types.

3) Use the InternalFields property to set a custom field value

The InternalFields property is also used to set the value of any field exposed by the API. For example, the TestRich__c field could be set as follows:

```
ContactService contactService = new ContactService(salesforceSession);
Contact contact = contactService.GetByEntityId("0037000000TGNKL");
contact.InternalFields.SetField("TestRich__c ", "New Value");
```

If the field is not string based you can also use `SetField<T>(string fieldName, T fieldValue)` to get other data types.

4) Use a GenericSalesforceEntity to access the custom object and/or field.

This is similar to the previous option but doesn't rely on the existing Contact object.

```
GenericSalesforceService genericSalesforceService = new GenericSalesforceService(sale
sforceSession, "MyCustomObject__c");

GenericSalesforceEntity contact = genericSalesforceService.GetByEntityId("00X7000000T
GNKL");

string richText = contact.InternalFields["TestRich__c"];
```

5) Inherit the default S4S class and add the required properties for a custom field

Inherit from the supplied Contact entity and add the required properties.

```
// Could equally use a ContactService/Contact here

GenericSalesforceService genericSalesforceService = new GenericSalesforceService(sale
sforceSession, "Contact");

GenericSalesforceEntity contact = genericSalesforceService.GetByEntityId("0037000000T
GNKL");

ContactExtension contactExtension = new ContactExtension(contact);

string richText = contactExtension.TestRich;
```

```
public class ContactExtension : Contact
{
	public ContactExtension(EntityBase entityBase) : base(entityBase) { }

/// <summary>
/// Test Rich Text
/// </summary>
public string TestRich
{
	get
	{
		return this.InternalFields.GetField<string>("TestRich__c");
	}
	set
	{
		this.InternalFields.SetField<string>("TestRich__c", value);
	}
}
```

8. Handling Custom Picklists

Values for custom Picklist fields in Salesforce can be set as strings. The following example is for a built in field, but the same approach will work for custom fields.

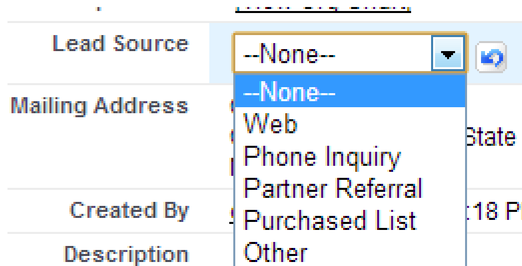
For the Account Rating picklist field the value can be set as follows:

```
/// <summary>
/// Manually created enum to represent known picklist values
/// </summary>
public enum AccountRatings
{
	Cold,
	 Warm,
	 Hot
}

[TestMethod]
public void UpdatePicklistField()
{
```

```
        SalesforceSession salesforceSession = SessionTest.GetActiveSession();

        AccountService accountService = new AccountService(salesforceSession);


        // Retrieve an existing Account. Just get the required fields.
         Id existingAccountId = (Id)"00190000002G9cF";

        Account account = accountService.GetByEntityId(existingAccountId,
              new SObjectField[] { Account.Fields.Id, Account.Fields.Rating});

         string existingRating = account.InternalFields["Rating"];


        // Example using internal fields to set a picklist value.
        // Would equally work for custom picklist fields.
       //account.InternalFields["Rating"] = "Hot";

        account.InternalFields["Rating"] = AccountRatings.Cold.ToString();

        SaveResult updateResult = accountService.UpdateEntity(account);
        Assert.IsTrue(updateResult.success);

        // Restore previous value using the generated property.
        account.Rating = existingRating;
        accountService.UpdateEntity(account);
}
```

9. Retrieving Picklist Values

Values for Picklist fields in Salesforce can be accessed as strings. The following example is for a built in field, but the same approach will work for custom fields.




```
public void GetLeadSourcePicklistValues()
{
        SalesforceSession salesforceSession = SessionTest.GetActiveSession();

        Field field = FieldService.FieldFromObject("Contact", salesforceSession ,
        "LeadSource");
```

```
        if(field.type == fieldType.picklist)
        {
                PicklistEntry[] picklistValues = field.picklistValues;

                foreach (PicklistEntry picklistValue in picklistValues)
                {
                        bool isDefaultValue = picklistValue.defaultValue;
                        string value = picklistValue.value;
                }
        }
}
```

10. Executing Salesforce Partner API methods

Use the SalesforceSession Binding property to execute API methods. In the following unit test example, the method ConvertLead is used to convert a Salesforce Lead to a Contact (Lead to Account and Opportunity conversions are also possible)

```
public void ConvertLead()
{
    SalesforceSession salesforceSession = SessionTest.GetActiveSession();

     Lead lead = null;
     LeadService leadService = null;
     ContactService contactService = null;
     Contact contact = null;

     try
     {
        lead = new Lead();
        lead.FirstName = "Terry";
        lead.LastName = "Humphris";
        lead.Company = "Humphris Co";
        lead.Status = "Closed - Converted";

        leadService = new LeadService(salesforceSession);
        SaveResult saveResult = leadService.Insert(lead);
          Assert.IsTrue(saveResult.success);

          // Lead will now have an Id
          Assert.IsTrue(leadService.ValidEntityId(lead.Id));

          LeadConvert leadConvert = new LeadConvert();
          leadConvert.leadId = lead.Id;
          leadConvert.overwriteLeadSource = false;
          leadConvert.doNotCreateOpportunity = true;
          leadConvert.convertedStatus = lead.Status;
          leadConvert.sendNotificationEmail = false;
          //leadConvert.contactId = contactId;
```

```
            //leadConvert.accountId = accountId;
            //leadConvert.opportunityName = opportunityName;

            LeadConvertResult[] lcr = salesforceSession.Binding.convertLead(
                 new LeadConvert[] { leadConvert });

            for (int i = 0; i < lcr.Length; i++)
            {
                 if (lcr[i].success)
                 {
                      Console.WriteLine("Conversion succeeded.\n");
                      LeadConvertResult result = lcr[i];

                      contactService = new ContactService(salesforceSession);

                      contact = contactService.GetByEntityId(result.contactId);

                      Assert.IsTrue(contactService.ValidEntityId(result.contactId));
                 }
                 else
                 {
                      Console.WriteLine("Failed: " + lcr[i].errors[0].message);
                      Assert.Fail();
                 }
            }
       }
       finally
       {
            if (contactService != null && contact != null && contact.Id != null)
            {
                 try { contactService.DeleteContact(contact); }
                 catch (Exception) { }
            }
       }
}
```

11. Attaching Files (doc, PDF, xls, etc.) to a Salesforce Contact Record

You need to create an attachment and set its Parent ID to the ID of the SF object that you are linking it to. The main thing to be aware of is that the attachment body needs to be a byte array. Please Google converting different types of documents to a byte array using C#

```
SalesforceSession salesforceSession = SessionTest.GetActiveSession();
ContactService contactService = new ContactService(salesforceSession);
AttachmentService attachmentService
       = new AttachmentService(salesforceSession);

//create a new contact
Contact contact = new Contact();
```

```
contact.FirstName = "John";
contact.LastName = "Doe";

SaveResult saveResult = contactService.SaveContact(contact);

string contactId = saveResult.id;

//create an attachment
Attachment attachment = new Attachment();

attachment.Name = "My attachment name";

//get the document as a byte array
byte[] bytes = System.IO.File.ReadAllBytes("MyPDF.pdf");

attachment.Body = bytes;

//set the Parent ID of the attachment to link it to the contact
attachment.ParentId = contactId;

//save the attachment
saveResult = attachmentService.Save(attachment);
```

12. Creating Chatter Records

Creating a Chatter record currently requires using the GenericEntity class.

```
GenericSalesforceEntity feedItem = new GenericSalesforceEntity("FeedItem");

//The ParentId is the ID of the sObject that the Feed Item is attached to.
// In this case it is a Contact ID.
contactFeed.InternalFields.SetField("ParentId", "0039000000Msoqj");

contactFeed.InternalFields.SetField("Body", "Some chatter text");

GenericSalesforceService chatterContactService = new
GenericSalesforceService(GetSalesforceSession, "FeedItem");

SaveResult saveResult = chatterContactService.Insert(feedItem);
```

13. Creating Chatter Records with an Attachment

Creating a Chatter record with an attachment add the attachment to the Chatter record rather than the using the conventional Attachment record as outlined in the “Attaching Files (doc, PDF, xls, etc.) to a Salesforce Contact Record”. Also, the byte array for the attachment must be converted to a base 64 string for the ContentData field of the FeedItem.

```
GenericSalesforceEntity feedItem = new GenericSalesforceEntity("FeedItem");
```

```
            //The ParentId is the ID of the sObject that the Feed Item is attached to. In
this case it is a Contact ID.
            contactFeed.InternalFields.SetField("ParentId", "0039000000Msoqj");
            contactFeed.InternalFields.SetField("Body", "Some chatter text");

            //Attach a file
            byte[] bytes = System.IO.File.ReadAllBytes("MyPDF.pdf");

            contactFeed.InternalFields.SetField("ContentData",
Convert.ToBase64String(fileChatter.FileBytes));
            contactFeed.InternalFields.SetField("MyPDF.pdf ", fileChatter.FileName);

            GenericSalesforceService chatterContactService = new
GenericSalesforceService(GetSalesforceSession, "FeedItem");

            SaveResult saveResult = chatterContactService.Insert(feedItem);
```

14. Searching Salesforce Records

You should always consider performance when searching Salesforce. The following code searches for Salesforce Contacts that meet specific criteria. The important points are:

- Use a **AddDataSourceFilter** method to modify the query that is sent to Salesforce. Only records that match this filter criterion will be pulled down.
- Within the `foreach` loop don't pull the Contact down again as this will send another query to Salesforce. You should avoid calling:
  `Contact person = contactService.GetByEntityId(contact.Id);`
- When calling QueryEntities only pass the fields that you require are parameters. These will be pulled down and populated into the Contact for use in the foreach loop. The fewer fields that are pulled the faster the query will go. If you don't need to pull the details of the corresponding Account this will also improve query performance.
- Consider how many Contact records will match the search criteria? If there are, say 2000 or 4000 records, the following approach will be fine as only two partner API calls will be required. If there are 10,000 or more matches we should consider an approach that supports paging.
- Consider how frequently the data accessed and how often does it change. It may be appropriate to store the query result in an `EntityCache` for a duration to reduce the number of API calls.

In the following example you can change the customBooleanFieldName to be "`Find_a_person__c`" and the AddDataSourceFilter to true.

```
[TestMethod]
public void GetByCustomFieldAndMailingCountry()
{
    SalesforceSession salesforceSession = SessionTest.GetActiveSession();
    ContactDataSource contactDataSource = new ContactDataSource(salesforceSession);

    // this could be a customer field. Uses built in field for the test case.
```

```
    string customBooleanFieldName = "IsDeleted";
    string filterCountry = "USA";

    // these will be converted into query filters in Salesforce so that only the
       matching rows will be returned.
    // TODO: Change the false to true when using Find_a_person__c
    contactDataSource.AddDataSourceFilter(
         customBooleanFieldName, ComparisonOperator.Equals, false);

    contactDataSource.AddDataSourceFilter(
         Contact.Fields.MailingCountry, ComparisonOperator.Equals, filterCountry);

    // specify the exact fields you need from the Contact here and they will be
       returned in the query. You may or may not need to return the field that you
       were filtering on.
     List<Contact> matchingContacts =
         contactDataSource.QueryEntities<Contact>(
         Contact.Fields.Id, Contact.Fields.MailingCountry,
         Contact.Fields.Account(
         Account.Fields.Id, Account.Fields.Name, Account.Fields.Website),
         new SObjectField() { Name = customBooleanFieldName });

      Assert.AreNotEqual(0, matchingContacts.Count);

      foreach (Contact contact in matchingContacts)
      {
         // confirm that the customFieldName filter was applied
         Assert.IsFalse(
             contact.InternalFields.GetField<bool>(customBooleanFieldName));

         // confirm that the mailing country filter was applied
         Assert.AreEqual(filterCountry, contact.MailingCountry);

         Account accountForContact = contact.Account;

         Assert.IsNotNull(accountForContact);

         string accountName = accountForContact.Name;
         Assert.IsNotNull(accountName);
      }
}
```

15. Improving the Performance of S4S

Typically you will get a session from Salesforce as follows:

```
SalesforceSession salesforceSession = new SalesforceSession("connectionStringName");
```

There may be times when a rapid real time response from Salesforce is critical, like when logging into Sitecore using credentials stored in a Salesforce contact record (this is called Security Connector functionality).

In these situations it is best to keep the Salesforce session alive as long as possible. You could do this either via the ASP.NET Cache, or with the SalesforceSessionSingleton base class that is part of S4S. It has a number of constructors.

```
/// <summary>
/// Implements the Singleton Pattern to return a SalesforceSession instance created u
sing the S4SConnStringUI connection string details
/// </summary>
public class SitecoreSalesforceSessionSingleton : SalesforceSessionSingleton
{
     private SitecoreSalesforceSessionSingleton(): base(null, "connectionStringName")
     {
     }

     /// <summary>
     /// Get the singleton instance of the Salesforce Session.
     /// </summary>
     public static SalesforceSession SessionInstance
     {
           get
           {
                 SitecoreSalesforceSessionSingleton singleton
                      = new SitecoreSalesforceSessionSingleton();
                 return singleton.Instance;
           }
           set
           {
                 SitecoreSalesforceSessionSingleton singleton
                      = new SitecoreSalesforceSessionSingleton();
                 singleton.SetNewInstance(value);
            }
      }
}
```

Then, whenever you need a SalesforceSession you make the following call instead:

```
SalesforceSession salesforceSession =
      SitecoreSalesforceSessionSingleton.SessionInstance;
```

Even then, because the default application pool session time-out is 20 minutes, sessions will expire meaning the next visitor will take a performance hit. To mitigate this you could increase the timeout in web.config and set the application pool’s worker process default idle timeout as described here - http://asp-net.vexedlogic.com/2012/05/23/aspasp-net-session-timeout-how-do-i-change-it/. This will, at least let you extend the refresh duration to what you want. You may also need to change the Session Storage Mode described in the same article.

Another approach is the KeepAlive page (http://sitecorebasics.wordpress.com/2011/06/21/basics-of-keepalive/ from Sitecore). With the Sitecore UrlAgent you can keep the SalesForceSession alive in the keepalive.aspx. The KeepAlive agent is set to 1 hour by default.

A cleaner solution would be to create a new ASPX page just for maintaining the SalesForceSession instead of modifying the Sitecore KeepAlive.aspx page. Then add a new URLAgent that calls that new web page on some frequency.

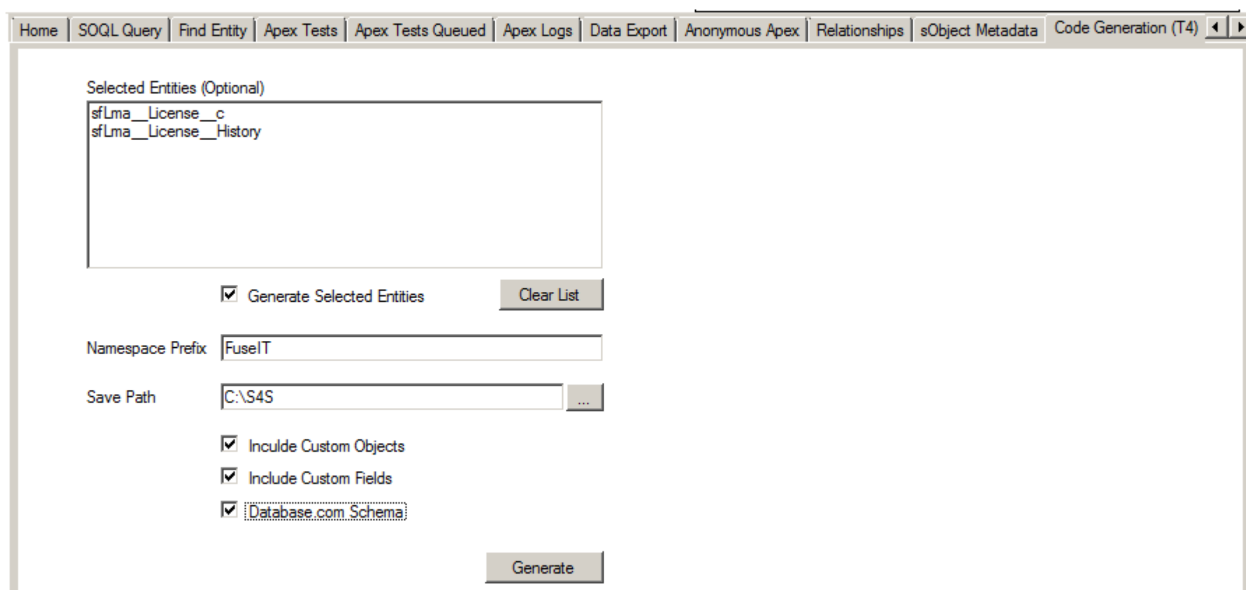
16. Using T4 Templates to Create Strongly Typed Objects

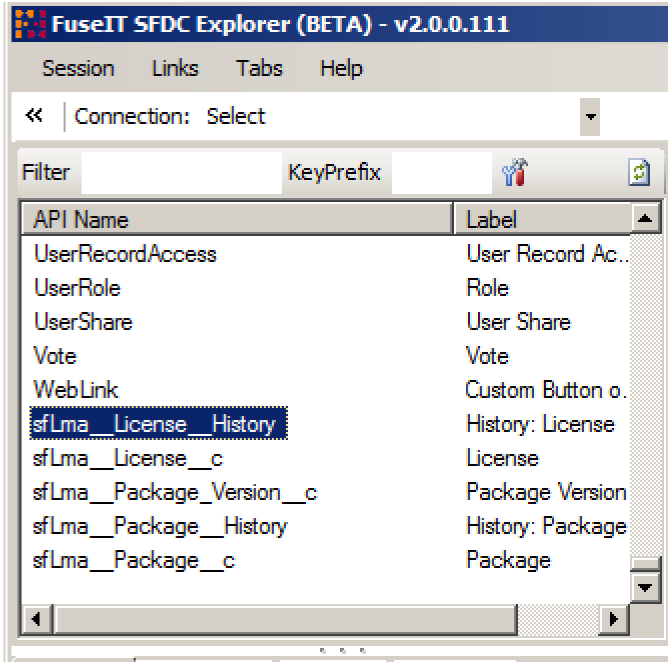
You can generate strongly typed classes for all, or some, of your Org within a namespace of your choice with T4 templates. This is a good option if you deal with Salesforce Orgs that have static fields that never change. The process is simple and repeatable once setup. The advantages of this approach are type safety and that you can use IntelliSense to work with the custom object/field.

The main disadvantage of having strongly typed custom objects and fields types is that when you change the schema in Salesforce, you need to regenerate the strongly typed objects and recompile your project.

The fastest way to create strongly typed objects is to use the FuseIT SFDC Explorer (download here). This is a free windows application that lets you explore Salesforce using the same technology S4S uses. We recommend you download and use it to validate your calls to Salesforce, logins, generate T4 templates and much more – as can be seen in the tabs in the following image.

After logging into the FuseIT SFDC Explorer with your Salesforce credentials and security token, select the Code Generation (T4) tab.





Drag each Salesforce object from the left panel (see left image) onto the entity canvas then select Generate Selected Entities.

Enter a Namespace and select the remaining options.

Clicking Generate will create a folder at the nominated location containing the strongly typed C# classes that you can add to your .NET project.