API Document

dotMailer®

The Digital Marketing Agency

The dotMailer API uses a SOAP interface and is remarkably easy to use. You can find out details on the methods at http://apiconnector.com. If you have had no experience of using SOAP you can look at some of the following links:

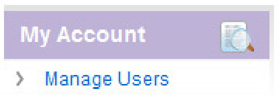
http://en.wikipedia.org/wiki/SOAP

http://www.w3schools.com/soap/default.asp

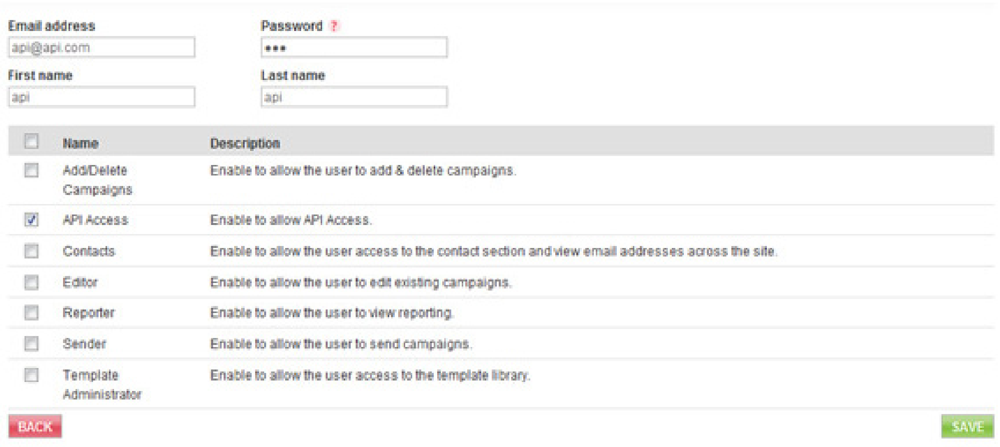
You can use the API however you wish, whether it be synchronising to your CRM, uploading and sending campaigns, advanced data management, etc. This document will show you how to use the methods with C#, PHP, and classic ASP.

The username and password you need to use for the API has to be a managed login; you can create one of these within your dotMailer account. When you login into your account, you can go to the "My Account" section in the top right of your screen

Click on manage users:



And click "add new" and you will see this screen



Email address	Password ?
api@api.com	•••
First name	Last name
api	api

☐	Name	Description
☐	Add/Delete Campaigns	Enable to allow the user to add & delete campaigns.
✓	API Access	Enable to allow API Access.
☐	Contacts	Enable to allow the user access to the contact section and view email addresses across the site.
☐	Editor	Enable to allow the user to edit existing campaigns.
☐	Reporter	Enable to allow the user to view reporting.
☐	Sender	Enable to allow the user to send campaigns.
☐	Template Administrator	Enable to allow the user access to the template library.

You need to make sure the API Access permission is selected, and then save the account. You will then be able to use these details to login to the API.

The API can be used for many different purposes; the main one will be contact management. To learn how to use the functions within the API you will need to know the structure of the objects, most notably the contact object which looks like:

```
<ID>int</ID>
<Email>string</Email>
<AudienceType>Unknown or B2C or B2B or B2M</AudienceType>
<DataFields>
  <Keys>
    <string>string</string>
    <string>string</string>
  </Keys>
  <Values>
    <anyType />
    <anyType />
  </Values>
</DataFields>
<OptInType>Unknown or Single or Double or VerifiedDouble</OptInType>
<EmailType>PlainText or Html</EmailType>
<Notes>string</Notes>
```

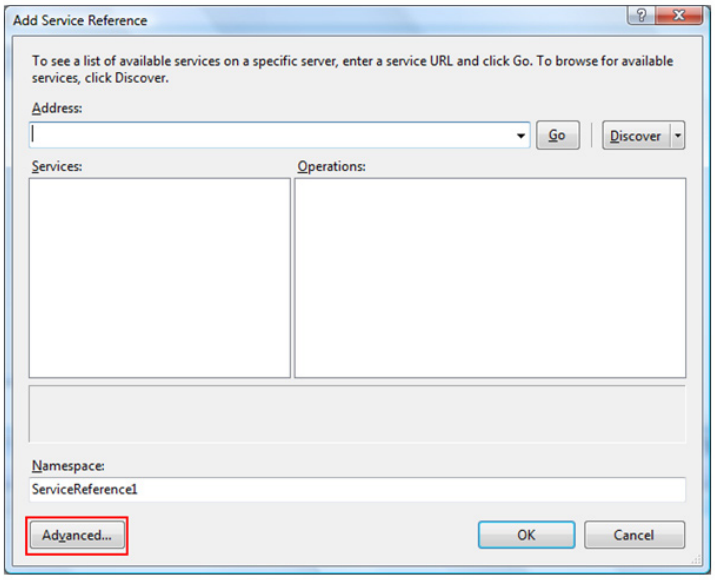
This is a standard type of SOAP object, having normal datatypes within it. The only part of this which may prove complex is the datafields - values fields. This is because they are flexible types. Because of this you will need to specify the type of value you are passing through. You will see how to do this below in the code examples.
You will not be able to used standard matched pairs through HTTP posts to use the API as the data structures are hierarchical/nested.

You can see the other data structures at http://apiconnector.com

The best way to learn to use the API is through examples;

.NET

To use the API within .NET is very simple. You will need to add the API as a web reference within Visual Studio. This document will show you how to use the .NET 2.0 web reference instead of the newer 3.0 service reference procedure. You can you this by right clicking on the project in the solution explorer and click “add service reference”, then click on the “advanced” button at the bottom


Add Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address:

Services:

Operations:

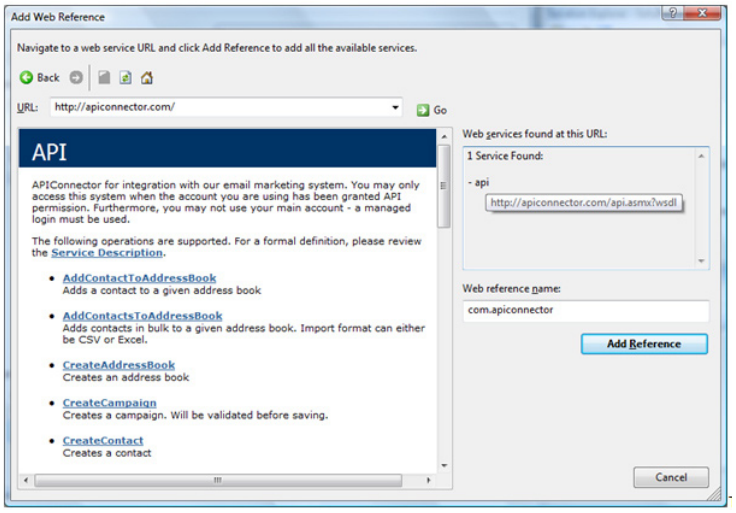
Namespace:

ServiceReference1

Then click the add web reference button

This will bring up the add web reference screen.

The URL to enter is http://apiconnector.com – if this is successful you should see the following details then simply click add reference.


Add Web Reference

Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://apiconnector.com/

API

APIConnector for integration with our email marketing system. You may only access this system when the account you are using has been granted API permission. Furthermore, you may not use your main account - a managed login must be used.

The following operations are supported. For a formal definition, please review the Service Description.

- AddContactToAddressBook
  Adds a contact to a given address book
- AddContactsToAddressBook
  Adds contacts in bulk to a given address book. Import format can either be CSV or Excel.
- CreateAddressBook
  Creates an address book
- CreateCampaign
  Creates a campaign. Will be validated before saving.
- CreateContact
  Creates a contact

Web services found at this URL:

1 Service Found:

- api

http://apiconnector.com/api.asmx?wsdl

Web reference name:

com.apiconnector

The API Web Reference will then have been added to your solution.

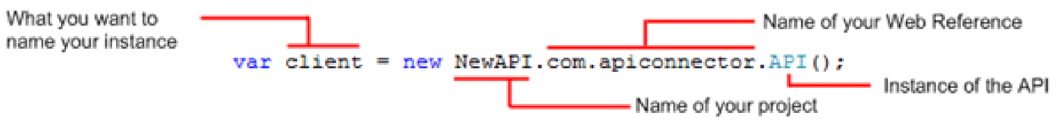
Now you can start creating your integration
The first methods we will go through are returning data as these methods are simpler in structure. The below example is how to retrieve a campaign which is already in dotMailer

```
public getCampaign()
        {
            string username = "example@example.com"; // variable for your username
            string password = "abc123xyz"; // variable for your password
            int campaignid = 1;
            var client = new NewAPI.com.apiconnector.API();//create an instance of the API

            var campaign = client.GetCampaign(username, password, campaignid);
            // call the getCampaign method, pass in the arguments and return the result

        }
```

The below diagram show in more detail the structure of creating the instance of the API



```
var client = new NewAPI.com.apiconnector.API();
```

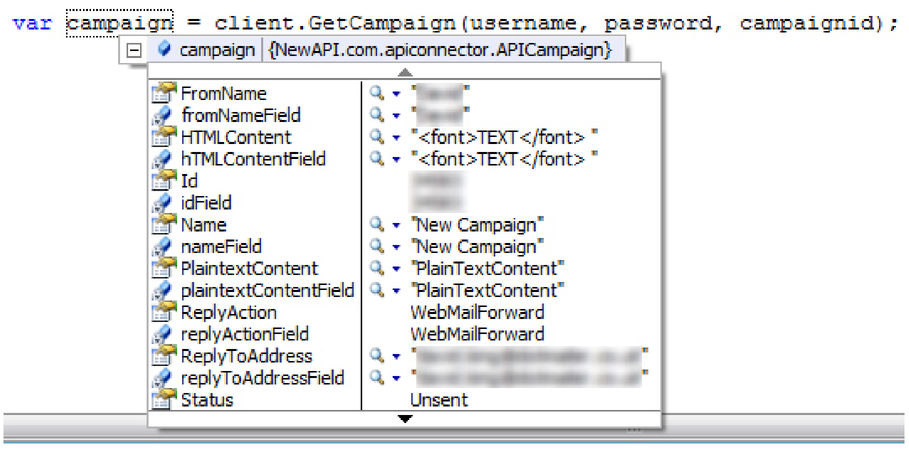
What you want to name your instance
Name of your Web Reference
Instance of the API
Name of your project

This class is quite self explanatory, it creates an instance of the API and the calls the getCampaign method with the required arguments. It will return the campaign as a campaign object called campaign. So you could the get the information about the campaign like

string html = campaign.HTMLContent;

This will create a string call html with the HTML of the campaign in, you will be able to do this for all of the contents of the campaign object (to see the structure of the any of the objects please look at the apiconnector.com site)

Or you can look at the campaign object e.g.



```
var campaign = client.GetCampaign(username, password, campaignid);
```

campaign {NewAPI.com.apiconnector.APICampaign}

FromName	"[redacted]"
fromNameField	"[redacted]"
HTMLContent	"<font>TEXT</font> "
hTMLContentField	"<font>TEXT</font> "
Id	[redacted]
idField	[redacted]
Name	"New Campaign"
nameField	"New Campaign"
PlaintextContent	"PlainTextContent"
plaintextContentField	"PlainTextContent"
ReplyAction	WebMailForward
replyActionField	WebMailForward
ReplyToAddress	"[redacted]"
replyToAddressField	"[redacted]"
Status	Unsent

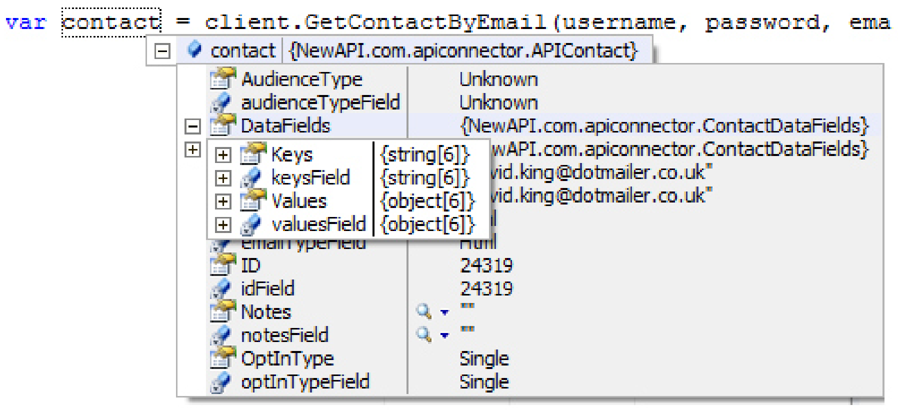
The same structure of class is used to retrieve data for other objects e.g. a contact

```
public GetContact()
{
    string username = "";
    string password = "";
    string email = "test@test.com";

    var client = new NewAPI.com.apiconnector.API();

    var contact = client.GetContactByEmail(username, password, email);
}
```

With the contact object you will notice that the datafields are an object within the contact object e.g.



```
var contact = client.GetContactByEmail(username, password, ema
  contact  {NewAPI.com.apiconnector.APIContact}
    AudienceType          Unknown
    audienceTypeField     Unknown
    DataFields            {NewAPI.com.apiconnector.ContactDataFields}
                          wAPI.com.apiconnector.ContactDataFields}
      Keys        {string[6]}  vid.king@dotmailer.co.uk"
      keysField   {string[6]}  vid.king@dotmailer.co.uk"
      Values      {object[6]}
      valuesField {object[6]}
    ID                    24319
    idField               24319
    Notes                 ""
    notesField            ""
    OptInType             Single
    optInTypeField        Single
```

The datafields object contacts two arrays, one called keys which is a array of strings which contains the names of the dataFields, and an 2nd array called values which is a array of objects. The values objects contains the values of the dataFields, it is an array of objects as it is capable of containing strings, ints, Booleans and dates.

You can get the values of these by:

```
NewAPI.com.apiconnector.ContactDataFields datafields = contact.DataFields;

string[] keys = datafields.Keys;
object[] values = datafields.Values;
```

Then you can iterate through the arrays to get the values out

Inserting data will require more code as you need to populate the fields before you call the method, the below method is inserting a campaign

```
public InsertCampaign()
{
    string username = "";
    string password = "";

    var client = new NewAPI.com.apiconnector.API();
    var campaign = new NewAPI.com.apiconnector.APICampaign();
    //create an instance of a campaign

    campaign.Name = "New Campaign";
    //populate the name of the campaign
    campaign.ReplyAction = NewAPI.com.apiconnector.ReplyActions.WebMailForward;
    //set the reply type this is set to forward mail to a email address
    campaign.ReplyToAddress = "reply@test.com";
    //the email address to forward replies to
    campaign.Status = NewAPI.com.apiconnector.CampaignStatus.Unsent;
    // the status of the campaign, this should normally be unsent
    campaign.Subject = "subject"; // the subject line
    campaign.HTMLContent = "<html>Content<a href='http=$UNSUB$'>Unsubscribe</a></html>'";
    // the html content, this would normally be retrieved from a different source
    campaign.PlaintextContent = "The Plain Text Content http://$UNSUB$";
    // the plain text content, all content will require an unsubscribe link

    var result = client.CreateCampaign(username, password, campaign);
    // call the CreateCampaign Method and pass in arguments

}
```

Inserting a contact is done in the same way, except you have the datafields object to insert as well, the code for this looks like

```
public AddContact()
{
    string username = "";
    string password = "";

    var client = new NewAPI.com.apiconnector.API(); //create an instance of the API

    var contact = new NewAPI.com.apiconnector.APIContact(); //create an instance of a contact

    contact.Email = "test@test.com";
    //populate the contact with an email address (this is the only mandatory field)
    contact.AudienceType = NewAPI.com.apiconnector.ContactAudienceTypes.B2B;
    // populate the contact with and audience type this example is Business to Business
    contact.EmailType = NewAPI.com.apiconnector.ContactEmailTypes.Html;
    // populate the contatc email type this will decided whether it will send a plain text
    //email or HTML to this contact
    contact.ID = -1; // this does not need to be specified here as we are creating the contact
    contact.Notes = "String of notes"; //Notes on the contact
    contact.OptInType = NewAPI.com.apiconnector.ContactOptInTypes.Double;
    // the manner the contact was subscribed

    contact.DataFields = new NewAPI.com.apiconnector.ContactDataFields();
    // the Datafields of the contact is a SOAP object and need to be created


    string[] keys = new String[2];
    // This is an array of strings for the names of the datafields
    keys[0] = "FirstName";          // These have to match up with the dataFields in dotMailer
    keys[1] = "LastName";
    // you will be able to get these field names through the ListContactDataLabels method
    keys[2] = "DateOfBirth";

    object[] values = new object[3];
    // as these can be of any types it has be created as a array of objects
    values[0] = "Joe";
    values[1] = "Bloggs";
    values[2] = new DateTime(1982, 12, 01); // 01/12/1982

    contact.DataFields.Keys = keys; //assign the keys array to the contact
    contact.DataFields.Values = values; // assign the values array to the contact

    var result = client.CreateContact(username, password, contact);
    // makes the call to the api and returns the result
```

dotMailer®

These are the basic methods of the API in .NET, if you find you have any problems with these then please contact support@dotmailer.co.uk

PHP

PHP doesn't have an inbuilt SOAP wrapper, so in the example we are using PHP_SOAP.dll which will come with most LAMP installations.

To enable this SOAP wrapper you have to edit you php.ini file to include extension=php_soap.dll

You will also need to make sure that the php_Soap.dll file is in your ext directory

We will start with a simple retrieval of a campaign the code to do this is

```
<?php

$username = ""; //API username
$password = ""; //API Password
$campaignid = 1;
// campaignid(you can get this from the listcampaigns method


$client = new SoapClient("http://apiconnector.com/api.asmx?WSDL");
//make an instance of the API


$params = array("username" => $username, "password" => $password, "campaignId" => $campaignid);
//Builds all the arguments into an array

$result = $client->GetCampaign($params);
// calls the API method with aruments and returns result as $result

$campaign = $result->GetCampaignResult;
//get the campaign object from the result

$htmlresult = $campaign->HTMLContent;
//gets the HTMLcontent from the campaingn object
?>
```

This code is firstly defining the username, password and campaignId, the three arguments needed by the GetCampaign method. Then it builds the arguments into an array and passes this array into the method which returns the result as an object called result. The result object is then broken down into it constituent parts

The below code is to return a contact by email address

The result variable which is returned is a contact object which you need to handle like an array of objects, to get the data out of this you need to

```
<?php

$username = ""; //your api username
$password ="";  //your api password
$email = "";    //the email you wish to find

$client = new SoapClient("http://apiconnector.com/api.asmx?WSDL");
 //creating a new instance of the API

$params = array("username" => $username, "password" => $password, "email" => $email);
// Building the arguments for the SOAP query

$result = $client->GetContactByEmail($params);
// calling the API method
var_dump($result);
// this will display the result into a webpage

?>
```

Then you can iterate through the two arrays to get all the data out.

```
$result = $client->GetContactByEmail($params); //returns the result as a variable

$Contact = $result->GetContactByEmailResult;
//Gets the contact object from the result variable
$email = $Contact->Email; // gets the email from the contact object
$emailid = $Contact->ID; // Gets the emailid from the contact object
$DataFields = $Contact->DataFields; // gets the datafield object from the contact object
$keys = $DataFields->Keys; // gets the keys object from the datafields object
$string = $keys->string; // gets the array of key names from the keys object
$key1 = $string[0]; //FirstName


$values = $DataFields->Values; // gets the values object from the datafields object
$AnyType = $values->anyType; // gets the anyType object from the Values object
$value1 = $AnyType[0]; //Value for FirstName
```

To insert data into dotMailer through the API is again similar; the code below is how you would insert a contact into dotMailer

```php
<?php


$username = ""; //apiusername
$password = ""; //api password
$addressbookid  = 1; //addressbook id for where you want to insert the contact
$email = ""; //email address you want to insert
$AudienceType = "Unknown"; // Audience type, this can be B2B, B2c etc
$OptInType = "Unknown"; // OptInType, this can be single, double etc
$EmailType ="Html";
//Email Type, this is whta kind of email the contact will recieve e.g. html or plaintext
$FirstName = "FirstName"; // firstname datafield
$LastName = "LastName"; // lastname datafield
$dob =  mktime(0,0,0,12,1,1982);
//this datafields will have to be already created in dotMailer

$client = new SoapClient("http://apiconnector.com/api.asmx?WSDL");
//create new instance of the api

$keys = array("FIRSTNAME", "LASTNAME", "DOB"); // creates an array of the datafield names

$typedVar1 = new SoapVar($FirstName, XSD_STRING, "string", "http://www.w3.org/2001/XMLSchema"); //creates the values
as a set datatype
$typedVar2 = new SoapVar($LastName, XSD_STRING, "string", "http://www.w3.org/2001/XMLSchema");  //see other notes
$typedVar3 = new SoapVar($dob, XSD_DATE, "date", "http://www.w3.org/2001/XMLSchema");
$Values = array($typedVar1, $typedVar2, $typedVar3); //makes an mixed array of the values


$Datafields = array("Keys" => $keys, "Values" => $Values); // makes an array of the datafield

$contact =array("Email" => $email,"AudienceType" => $AudienceType, "OptInType" => $OptInType, "EmailType" =>
$EmailType ,"ID"=> -1, "DataFields" => $Datafields);
// creates an array which mimiks the structure of the contact object
$params = array("username" => $username, "password" => $password, "contact" => $contact , "addressbookId" =>
$addressbookid);
// builds an array of the parameter to pass into the API method
$result = $client->AddContactToAddressBook($params);
// calls the API method with the arguments and returns the result
?>
```

What you may notice about the code above is the use of the variable builder i.e

```php
$typedVar1 = new SoapVar($FirstName, XSD_STRING, "string", "http://www.w3.org/2001/XMLSchema");
```

What this does is sets the type of the dataField, this needs to be done else the API will reject the value as it won’t know what type it is.

These are the basic methods of using the API within PHP, if you have any difficulties using this then please contact support@dotmailer.co.uk