®

iService®

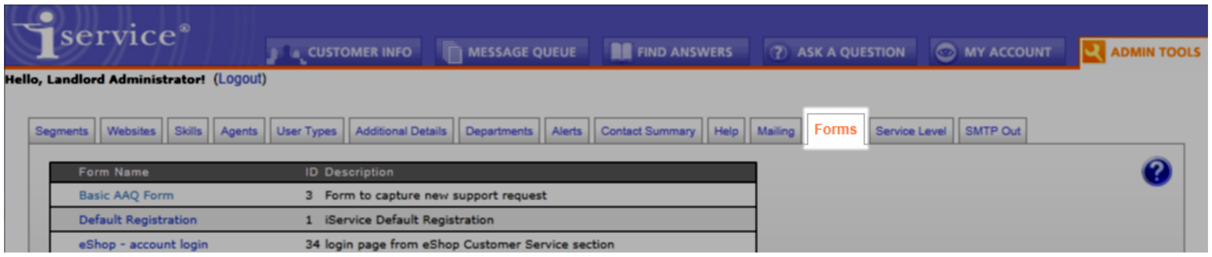
iService Forms User Guide

One-to-One Service.com is a software development company based in Austin, TX.
iService is a product of One-to-One Service.com, Inc.

Overview

1 Overview

iService Forms and the iService Scripting Language (ISL) are a server-side scripting technology that can be used to create dynamic and interactive web applications that leverage the iService web services. An iService Form is an ASP.net page (forms.aspx) that contains server-side scripts that are processed by the iService web server before being sent to the user's browser. You can combine the ISL commands with Hypertext Markup Language (HTML) to create customized interactive Web sites and business processes that leverage all the features of the iService customer interaction system. Forms are created from within the Admin Tools section of iService as shown below.



Form Name	ID	Description
Basic AAQ Form	3	Form to capture new support request
Default Registration	1	iService Default Registration
eShop - account login	34	login page from eShop Customer Service section

The iService Forms Tab

The forms interface allows you to create and manage forms of two types: embedded iService forms, and standalone forms. Embedded forms, such as registering as a customer within iService, are designed to be used as part of an existing iService page. That is, they only function when loaded within one of the standard iService interface pages (e.g., findanswers.aspx). These embedded forms allow you to modify the standard iService user interface. They are not intended to be viewed from the forms.aspx page.

iService forms consist of two elements: the HTML contained within the page (Form Body), and iService actions that the form triggers within iService when the form is submitted (Submit Actions). iService commands can be inserted into your form HTML to interact with iService, such as capturing customer information and saving it as a new contact. These commands, known as the iService Scripting Language, are inserted directly into the HTML of your form and are enclosed within $ signs.

When an iService form is loaded, the forms.aspx page will substitute iService commands with the code necessary to build the final, functional page that interacts with iService to complete its actions.

Intended Users of the Forms Interface

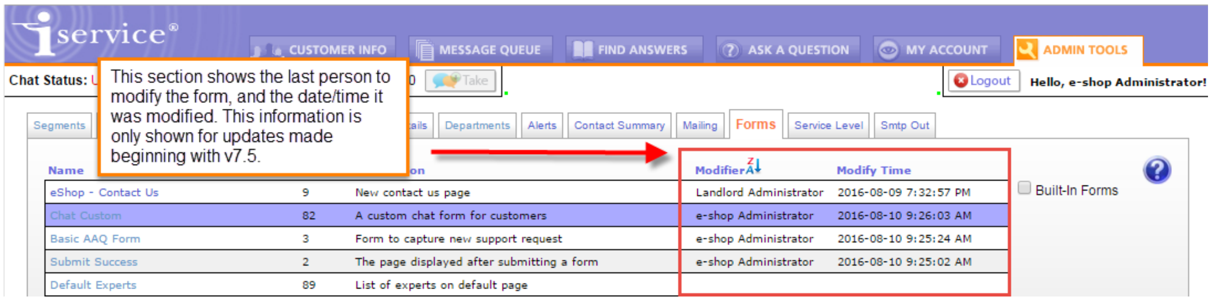
The purpose of the forms interface is to empower business analysts to build various forms that support workflow, customer service, sales, and marketing needs. The user that creates the form does not need to know anything about the underlying web services or .NET code involved. They should have a good understanding of the concepts within iService, and will often be your iService administrator. They build the functionality of the form by simply inserting the commands and configuring the form actions using the graphical interface provided by iService.

Once the form body is created, your web or graphic designer can create HTML for forms without having to know anything about the iService system. This separation of “setup” and “design” makes it easy to build professional looking forms very quickly.

Creating and Editing Forms

2 Creating and Editing Forms

The simplest way to create a form is to use the Forms Wizard, but you can also create forms manually or edit existing forms directly from the Admin Tools - Forms tab. Forms are accessed from within the Admin Tools - Forms page of iService as shown below. Beginning with v7.5, the last modifier and modify date/time are displayed.



This section shows the last person to modify the form, and the date/time it was modified. This information is only shown for updates made beginning with v7.5.

Name			Modifier	Modify Time
eShop - Contact Us	9	New contact us page	Landlord Administrator	2016-08-09 7:32:57 PM
Chat Custom	82	A custom chat form for customers	e-shop Administrator	2016-08-10 9:26:03 AM
Basic AAQ Form	3	Form to capture new support request	e-shop Administrator	2016-08-10 9:25:24 AM
Submit Success	2	The page displayed after submitting a form	e-shop Administrator	2016-08-10 9:25:02 AM
Default Experts	89	List of experts on default page		

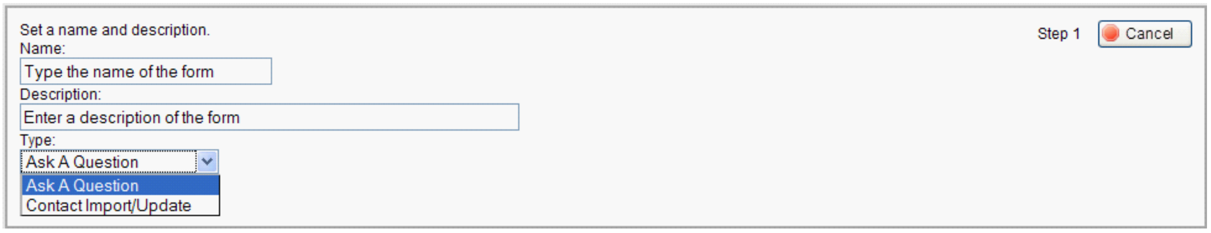
Using the Forms Wizard - Ask a Question Form
Using the Forms Wizard - Contact Import Form
Creating a Form Manually
The Variable Picker

2.1 Using the Forms Wizard - Ask a Question Form

There are four steps within the Ask a Question form wizard that are described below.
The wizard will create both the Form Body and the required Form Actions for an Ask a Question interaction. The setup for Tickets and Notes are very similar, so you can use this wizard as a quick way to get that code. For Tickets you simply check the box on the Create Ticket/Ask a Question action. For a Note, you can add a new Create Note action below the Find/Create contact action and copy the values from the AAQ action that the wizard generates.

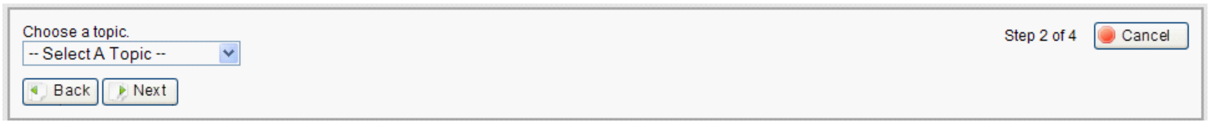
Step 1. Name the form, select the type of form, and click next.

The forms wizard can create two types of forms: ask a question and contact import. Select Ask a Question to create a form that captures a customer question.


Set a name and description.
Name:
Type the name of the form
Description:
Enter a description of the form
Type:
Ask A Question
Ask A Question
Contact Import/Update

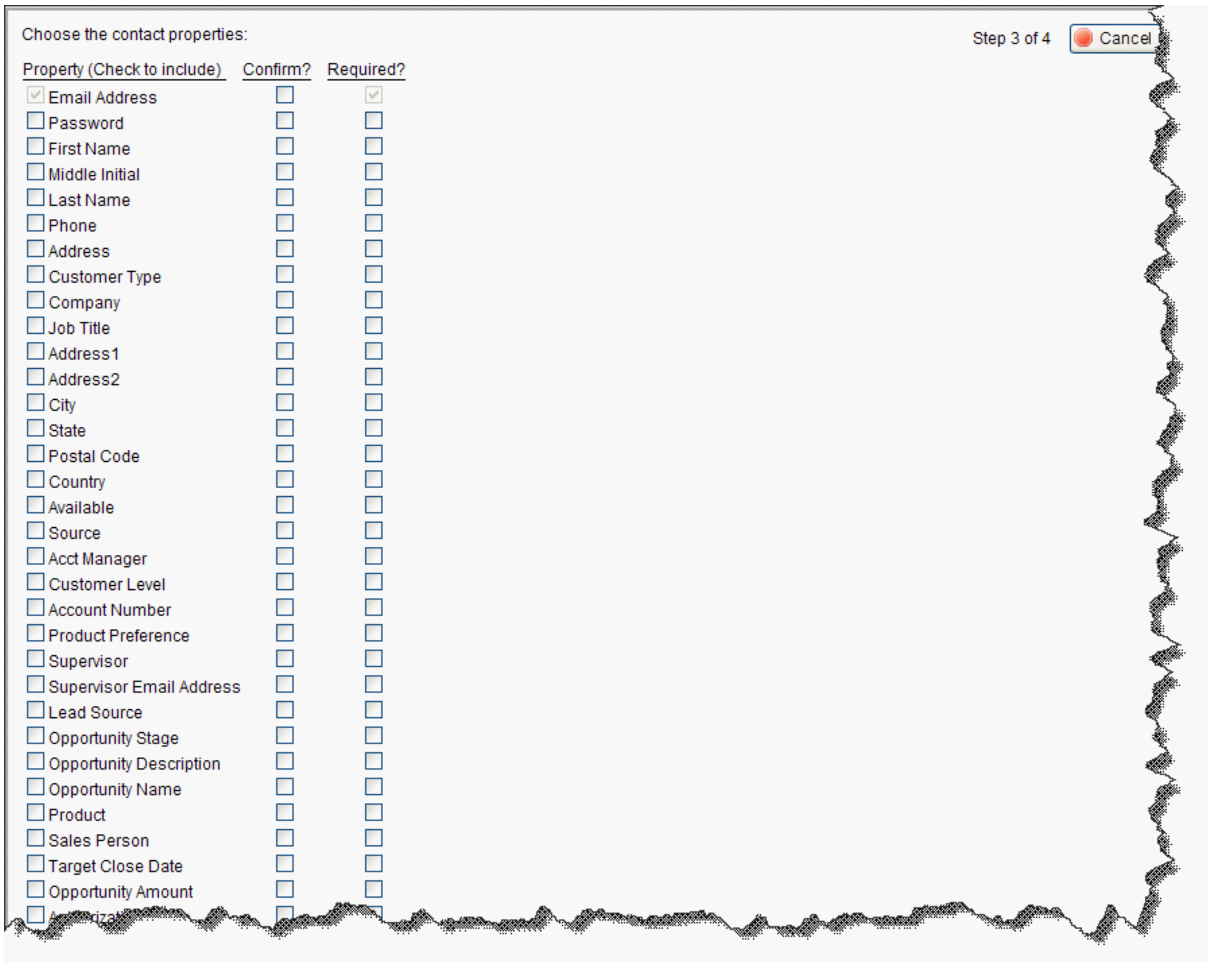
Step 2. Choose the target topic for the question.

The second step of the wizard is to specify the topic for the question or ticket. This is the topic into which the question or ticket will be placed. The topics available will be limited to segments that are associated with the form's website. For more information about iService website, see the iService Configuration Guide.


Choose a topic.
-- Select A Topic --

Step 3. Choose the Contact Properties to include on the form

Email address is always required and is selected by default. If you want the user to confirm their input, check the “Confirm” box. To make the property required, check the “Required” box. All contact properties associated with all segments of the tenant are displayed here.



Choose the contact properties:

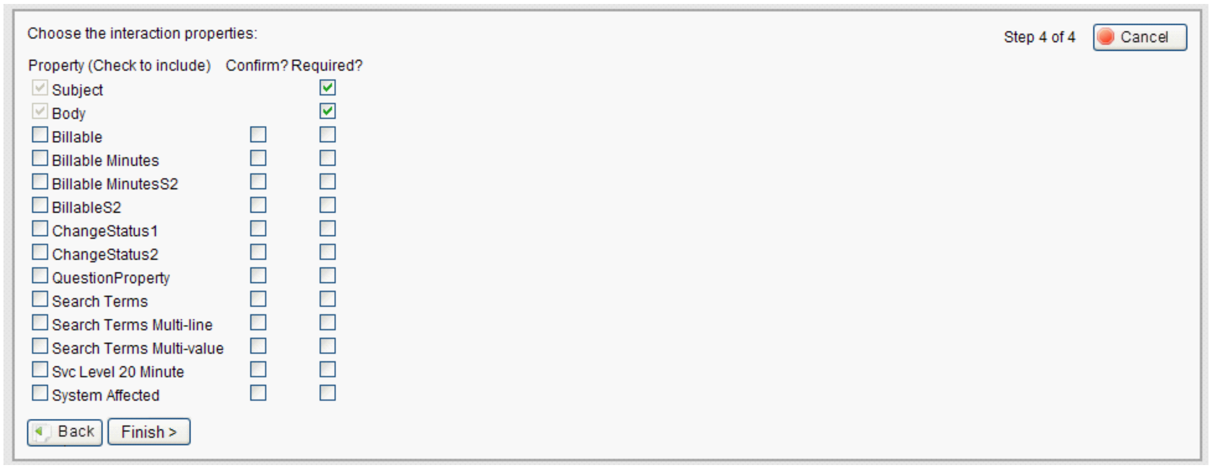
Step 3 of 4

Property (Check to include)	Confirm?	Required?
✓ Email Address	☐	✓
☐ Password	☐	☐
☐ First Name	☐	☐
☐ Middle Initial	☐	☐
☐ Last Name	☐	☐
☐ Phone	☐	☐
☐ Address	☐	☐
☐ Customer Type	☐	☐
☐ Company	☐	☐
☐ Job Title	☐	☐
☐ Address1	☐	☐
☐ Address2	☐	☐
☐ City	☐	☐
☐ State	☐	☐
☐ Postal Code	☐	☐
☐ Country	☐	☐
☐ Available	☐	☐
☐ Source	☐	☐
☐ Acct Manager	☐	☐
☐ Customer Level	☐	☐
☐ Account Number	☐	☐
☐ Product Preference	☐	☐
☐ Supervisor	☐	☐
☐ Supervisor Email Address	☐	☐
☐ Lead Source	☐	☐
☐ Opportunity Stage	☐	☐
☐ Opportunity Description	☐	☐
☐ Opportunity Name	☐	☐
☐ Product	☐	☐
☐ Sales Person	☐	☐
☐ Target Close Date	☐	☐
☐ Opportunity Amount	☐	☐

Choosing Contact Properties

⊟ Step 4. Choose the Interaction Properties to include on the form

The subject and body of the interaction are always required and are selected by default. If you want the user to confirm their input, check the “Confirm” box. To make the property required, check the “Required” box. All interaction properties associated with all segments of the tenant are available for selection.



Choose the interaction properties:

Step 4 of 4

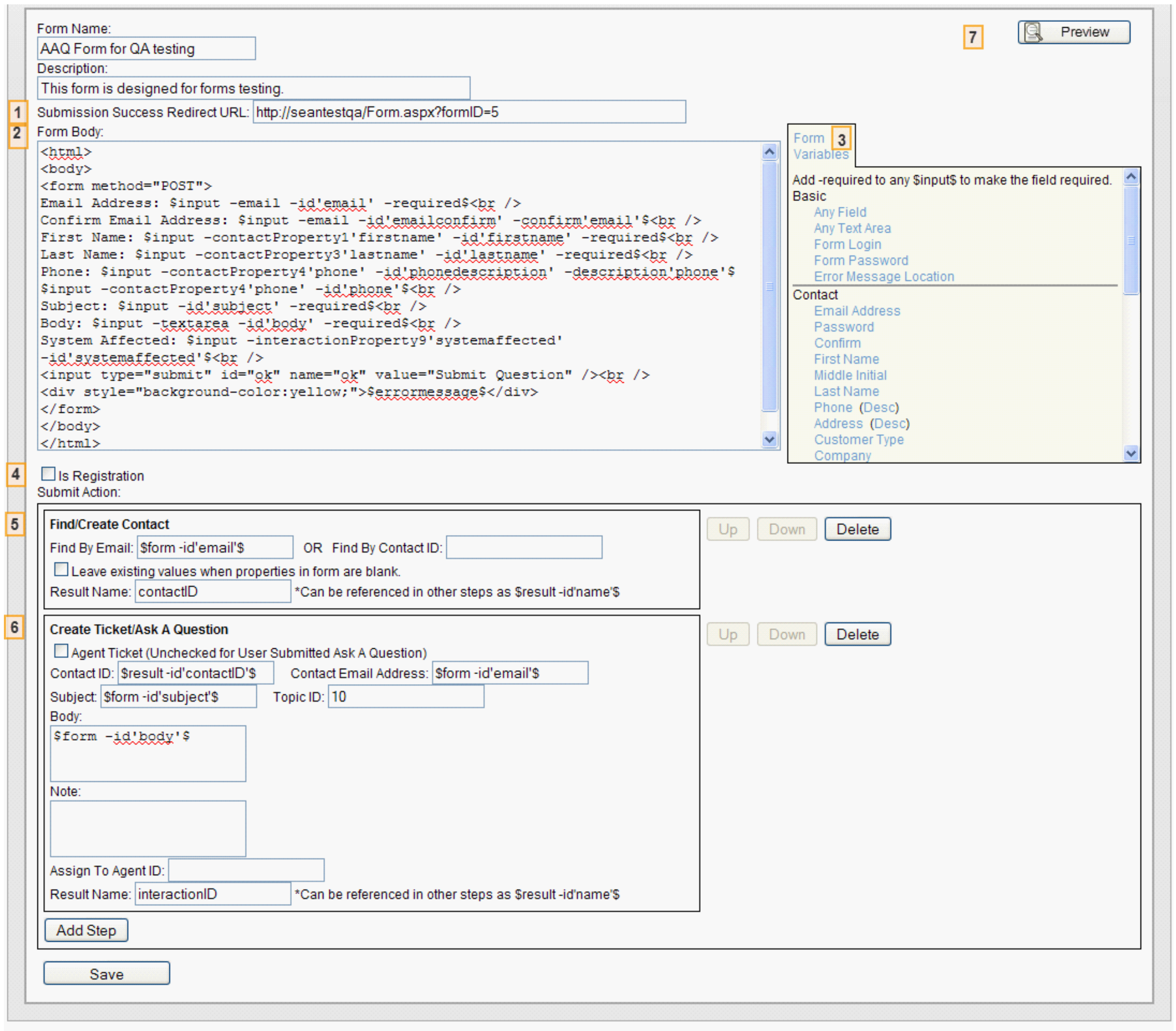
Property (Check to include)	Confirm?	Required?
✓ Subject		✓
✓ Body		✓
☐ Billable	☐	☐
☐ Billable Minutes	☐	☐
☐ Billable MinutesS2	☐	☐
☐ BillableS2	☐	☐
☐ ChangeStatus1	☐	☐
☐ ChangeStatus2	☐	☐
☐ QuestionProperty	☐	☐
☐ Search Terms	☐	☐
☐ Search Terms Multi-line	☐	☐
☐ Search Terms Multi-value	☐	☐
☐ Svc Level 20 Minute	☐	☐
☐ System Affected	☐	☐

Choosing Interaction Properties

The Completed Form

After the form wizard completes, iService displays the completed form ready for further editing and HTML customization. The actions are automatically populated and generally will not require any changes.

The form generated by the example above is shown below.



Form Name: AAQ Form for QA testing

Description: This form is designed for forms testing.

Submission Success Redirect URL: http://seantestqa/Form.aspx?formID=5

Form Body:

```
<html>
<body>
<form method="POST">
Email Address: $input -email -id'email' -required$<br />
Confirm Email Address: $input -email -id'emailconfirm' -confirm'email'$<br />
First Name: $input -contactProperty1'firstname' -id'firstname' -required$<br />
Last Name: $input -contactProperty3'lastname' -id'lastname' -required$<br />
Phone: $input -contactProperty4'phone' -id'phonedescription' -description'phone'$
$input -contactProperty4'phone' -id'phone'$<br />
Subject: $input -id'subject' -required$<br />
Body: $input -textarea -id'body' -required$<br />
System Affected: $input -interactionProperty9'systemaffected'
-id'systemaffected'$<br />
<input type="submit" id="ok" name="ok" value="Submit Question" /><br />
<div style="background-color:yellow;">$errormessage$</div>
</form>
</body>
</html>
```

Form Variables

Add -required to any $input$ to make the field required.

Basic
- Any Field
- Any Text Area
- Form Login
- Form Password
- Error Message Location

Contact
- Email Address
- Password
- Confirm
- First Name
- Middle Initial
- Last Name
- Phone (Desc)
- Address (Desc)
- Customer Type
- Company

☐ Is Registration

Submit Action:

Find/Create Contact

Find By Email: $form -id'email'$ OR Find By Contact ID:

☐ Leave existing values when properties in form are blank.

Result Name: contactID *Can be referenced in other steps as $result -id'name'$

Create Ticket/Ask A Question

☐ Agent Ticket (Unchecked for User Submitted Ask A Question)

Contact ID: $result -id'contactID'$ Contact Email Address: $form -id'email'$

Subject: $form -id'subject'$ Topic ID: 10

Body: $form -id'body'$

Note:

Assign To Agent ID:

Result Name: interactionID *Can be referenced in other steps as $result -id'name'$

Form Example

1 – Submission Success Redirect URL - If the form includes a Submit button, the "Submission Success Redirect URL" listed here will be displayed after the form is successfully submitted. This can be any URL, including other iService forms. You must include the full http or https path with the URL. If no URL is specified, the form will reload itself after it is submitted. Therefore, if the form is accessed directly by users we highly recommend including a redirect page.

2 – Form Body - The body of the form is displayed in this text area, and is editable. For larger or more complex forms, most users will edit the form using an HTML editor such as Visual Studio or Microsoft Visual Web Developer.

3 – Form commands - A command picker is provided to simplify adding new elements to your form. There are three types of commands: Basic, Contact, and Interaction.
4 – IsRegistration - “Is Registration” indicates whether the form will be used as a registration page embedded within the iService application.
The action section of a form defines the actions taken within iService when the form is submitted. In this example there are two actions: Find/Create Contact and Create Ticket/Ask a Question. These are the most common actions when forms are used to capture customer input. The actions from this example are described below.
5 – Find/Create Contact - The first submit action, Find/Create Contact, determines whether an account already exists for the customer. If an account does not exist, it creates a new iService contact record. The lookup process can be based on either email/login or iService ContactID. This is described below.
6 – Create Ticket/Ask a Question - The second action creates the actual Ticket or Ask a Question interaction within iService. The required fields for a new interaction are defined within the action and must be properly defined for the form to be successfully submitted. The required and optional fields are described below.
7 – Preview – Click the “Preview” button to load the form in a new window.

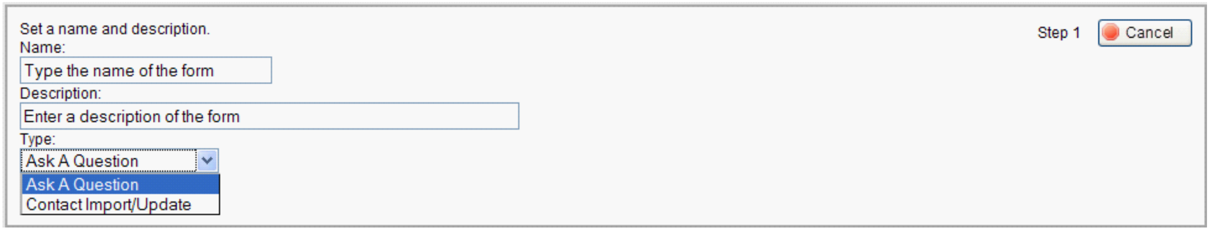
2.2 Using the Forms Wizard - Contact Import Form

Contact Import forms can be used in conjunction with the iService Batch Form Submit utility to import contacts into your iService tenant. This allows you to take any comma separated value file of contact information and process it with the batch utility to create contacts in iService. For information on how to use the batch utility, see the iService Batch Form Submission Utility user guide.

There are two steps within the Contact Import form wizard:

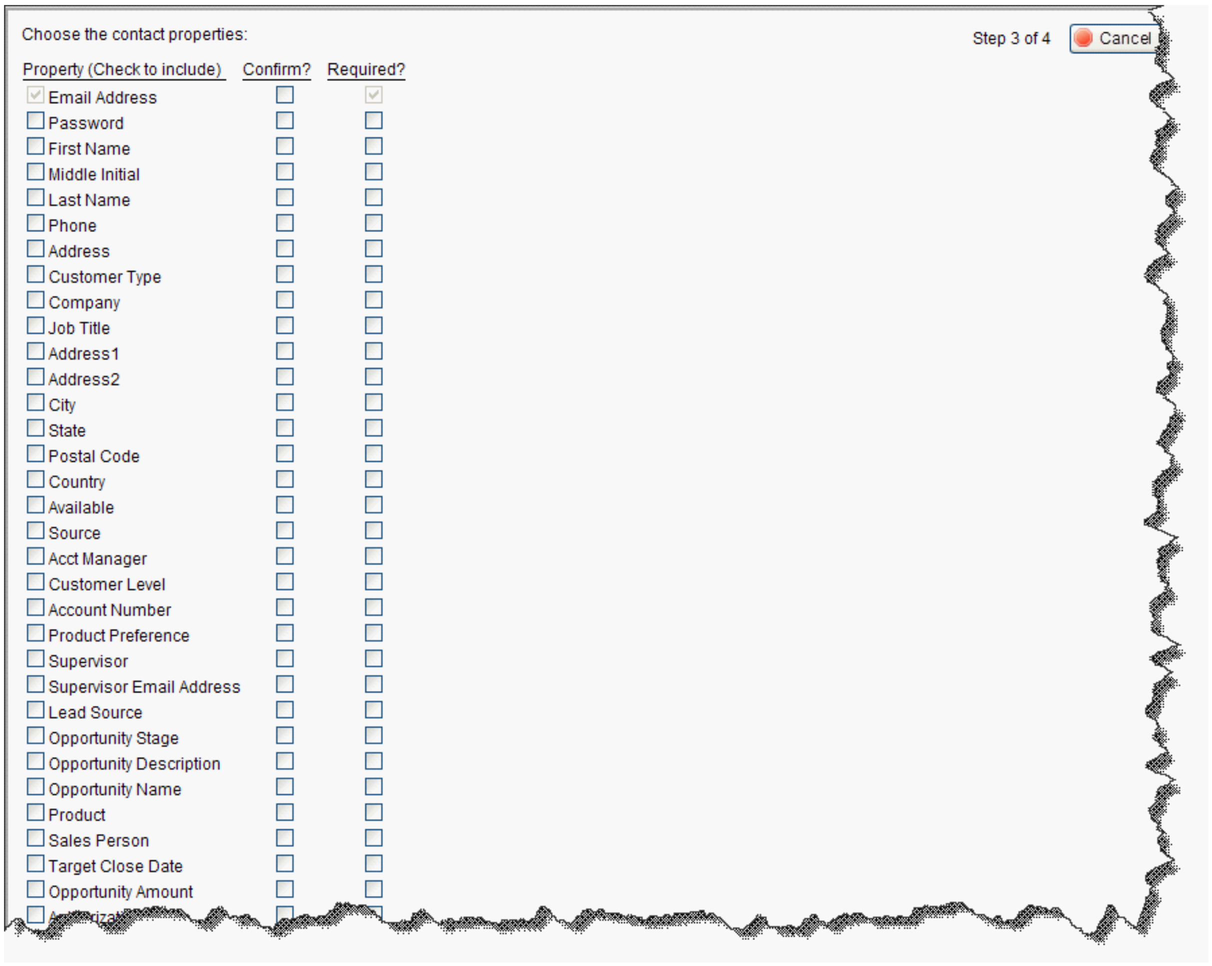
Step 1. Name the form, select the type of form, and click next.

The forms wizard can create two types of forms: ask a question and contact import. Select Contact Import/Update to create a form that creates new contacts in iService.


Set a name and description.
Name:
Type the name of the form
Description:
Enter a description of the form
Type:
Ask A Question

Step 2. Choose the Contact Properties to include on the form

Email address is always required and is selected by default, as it is a unique identifier for contacts within iService . If you want the user to confirm their input, check the “Confirm” box. To make the property required, check the “Required” box. All contact properties associated with all segments of the tenant are displayed here.



Choose the contact properties:

Property (Check to include)	Confirm?	Required?
✓ Email Address	☐	✓
☐ Password	☐	☐
☐ First Name	☐	☐
☐ Middle Initial	☐	☐
☐ Last Name	☐	☐
☐ Phone	☐	☐
☐ Address	☐	☐
☐ Customer Type	☐	☐
☐ Company	☐	☐
☐ Job Title	☐	☐
☐ Address1	☐	☐
☐ Address2	☐	☐
☐ City	☐	☐
☐ State	☐	☐
☐ Postal Code	☐	☐
☐ Country	☐	☐
☐ Available	☐	☐
☐ Source	☐	☐
☐ Acct Manager	☐	☐
☐ Customer Level	☐	☐
☐ Account Number	☐	☐
☐ Product Preference	☐	☐
☐ Supervisor	☐	☐
☐ Supervisor Email Address	☐	☐
☐ Lead Source	☐	☐
☐ Opportunity Stage	☐	☐
☐ Opportunity Description	☐	☐
☐ Opportunity Name	☐	☐
☐ Product	☐	☐
☐ Sales Person	☐	☐
☐ Target Close Date	☐	☐
☐ Opportunity Amount	☐	☐

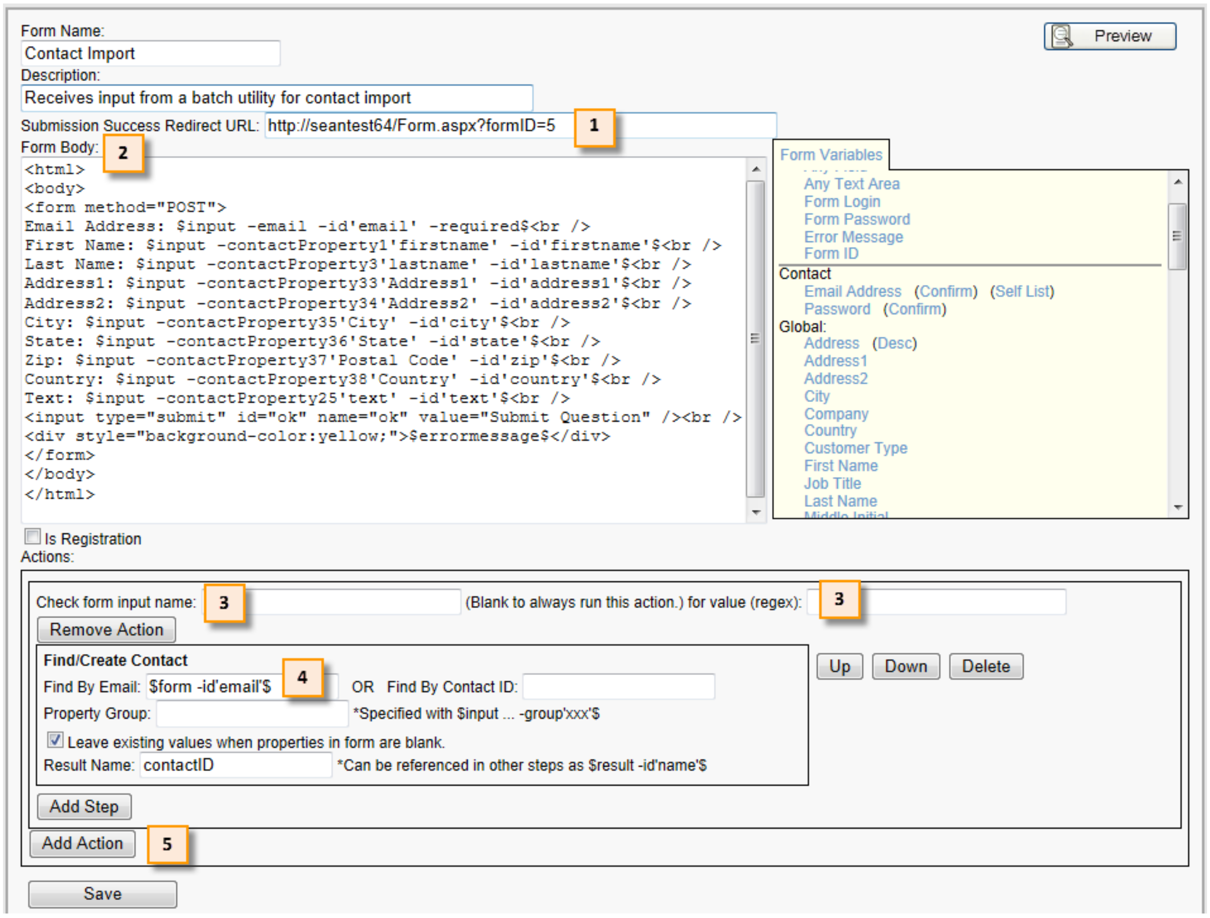
Choosing Contact Properties

After you select the properties for the form, click the Finish button to generate the form.

⊟ The Completed Form

After the form wizard completes, iService displays the completed form ready for further editing and HTML customization. The actions are automatically populated and generally will not require any changes.

The form generated by the example above is shown below.



Form Name: Contact Import

Description: Receives input from a batch utility for contact import

Submission Success Redirect URL: http://seantest64/Form.aspx?formID=5

Form Body:

```
<html>
<body>
<form method="POST">
Email Address: $input -email -id'email' -required$<br />
First Name: $input -contactProperty1'firstname' -id'firstname'$<br />
Last Name: $input -contactProperty3'lastname' -id'lastname'$<br />
Address1: $input -contactProperty33'Address1' -id'address1'$<br />
Address2: $input -contactProperty34'Address2' -id'address2'$<br />
City: $input -contactProperty35'City' -id'city'$<br />
State: $input -contactProperty36'State' -id'state'$<br />
Zip: $input -contactProperty37'Postal Code' -id'zip'$<br />
Country: $input -contactProperty38'Country' -id'country'$<br />
Text: $input -contactProperty25'text' -id'text'$<br />
<input type="submit" id="ok" name="ok" value="Submit Question" /><br />
<div style="background-color:yellow;">$errormessage$</div>
</form>
</body>
</html>
```

☐ Is Registration

Actions:

Check form input name: (Blank to always run this action.) for value (regex):

Find/Create Contact

Find By Email: $form -id'email'$ OR Find By Contact ID:

Property Group: *Specified with $input ... -group'xxx'$

☑ Leave existing values when properties in form are blank.

Result Name: contactID *Can be referenced in other steps as $result -id'name'$

A Completed Contact Import Form

1 – Submission Success Redirect URL - If the form includes a Submit button, the "Submission Success Redirect URL" listed here will be displayed after the form is successfully submitted. This can be any URL, including other iService forms. You must include the full http or https path with the URL. If no URL is specified, the form will reload itself after it is submitted. Therefore, if the form is accessed directly by users we highly recommend including a redirect page.

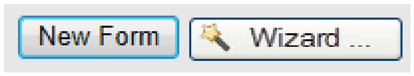
2 – Form Body - The body of the form is displayed in this text area, and is editable. For larger or more complex forms, most users will edit the form using an HTML editor such as Visual Studio or Microsoft Visual Web Developer.

The action section of a form defines the actions taken within iService when the form is submitted. In this example there is only one action: Find/Create Contact.

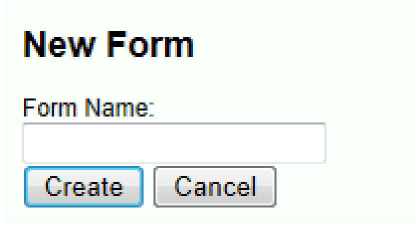
3 - Check Form Input Name / Form Value (regex) - Since forms can contain more than one action (i.e., more than one Submit in the form body), these optional values are used to specify which action is used. To use multiple actions you tag each action with a form parameter which is checked and the value checked for. If the specified form field has the value you choose then that form action is run. So for example you could have a parameter to the form in the URL such as mode=1 or mode=2, and in the form actions you would have both check for "mode" and one action check for value "1" and the other for value "2".
If the form only contains a single action, as in this example, these can be left blank.
4 – Find/Create Contact - This action determines whether an account already exists for the customer. If an account does not exist, it creates a new iService contact record. The lookup process can be based on either email/login or iService ContactID.
5 - You can add additional steps to the selected action, or add additional actions tied to other "Submit" functions within the form body.

2.3 Creating a New Form Manually

To manually create a new form , click the New Form button as shown below.



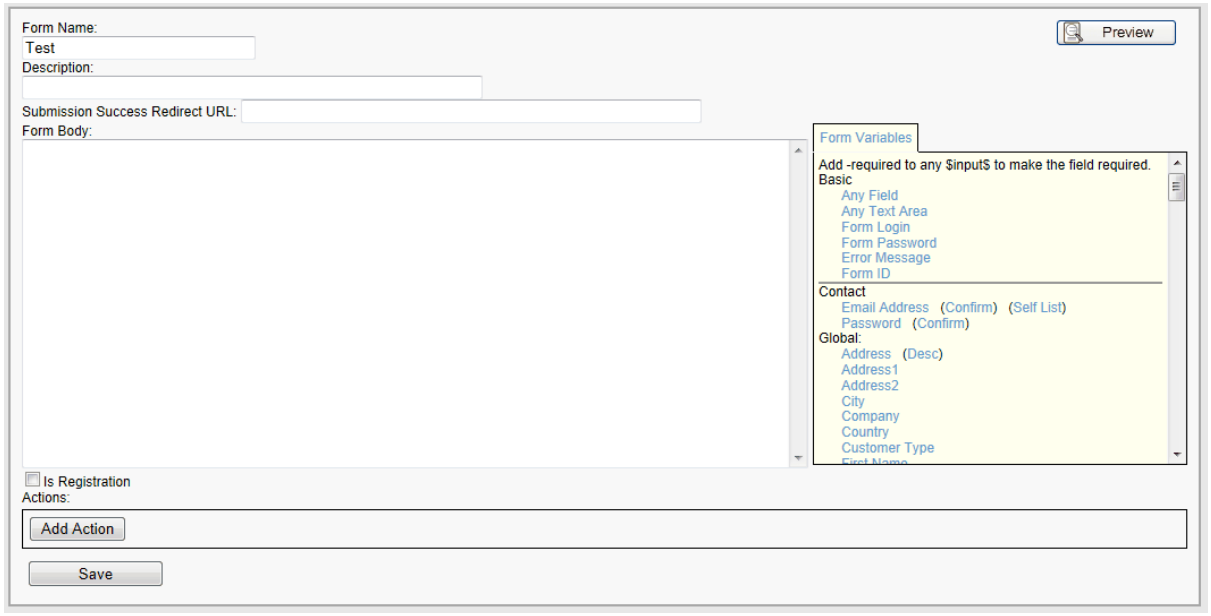
Then, enter a short name for your form in the Form Name box.



After you create the form, you have the option to enter a long description and and save the changes to your form. The form consists of a Form Body, and Form Actions that are triggered when the form is submitted. The form body is an HTML page that uses the iService scripting language to interact with the iService database. The form actions are common functions within iService such as finding or creating contacts, submitting ask a question forms, updating email list subscriptions, etc.

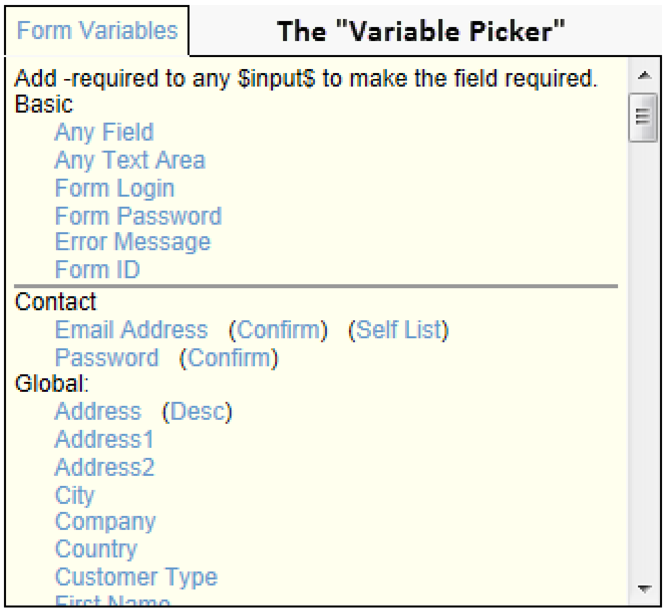
For more details on form commands in the iService scripting language, see the Form commands section.

For more details on form actions, see the Form Actions section.



A Blank Form

2.4 The Variable Picker



Add -required to any $input$ to make the field required.
Basic
Any Field
Any Text Area
Form Login
Form Password
Error Message
Form ID
Contact
Email Address (Confirm) (Self List)
Password (Confirm)
Global:
Address (Desc)
Address1
Address2
City
Company
Country
Customer Type

The variable picker displays a list of commands that are available for your form including the standard properties and all custom properties you created for your tenant. The commands, or variables, are grouped based upon their type. The variable picker performs most of the basic configuration for the command and is the preferred way to insert variables into your form.

The types of variables available are:
Basic: These are general purpose commands not specific to individual properties.
Contact: These are contact properties used for grading the contacts are updating the values for existing contacts in iService. Email address and password have unique characteristics and are listed at the top, followed by the global contact properties and custom properties sorted by iService segment.
Interaction: This section presents a list of all interaction properties within the tenant.
Lists: All of the mailing lists within the tenant are displayed in the lists section.
Campaigns: all of the mailing campaigns within the segment are displayed in the campaigns section.
Conditionals: These are used to evaluate the status of the form, and then modify the display based upon the state. All three of these variables use the IF/THEN/ELSE format.
LoggedIn: These variables are used to display information about the user logged into the form.

MyAccount: The My Account variables are used to create a custom version of the My Account - Subscriptions page within iService.
AskAQuestion: The AskAQuestion section includes the -Topics parameter for the $Input command. This displays a topic tree with all available public topics when inserted into an Ask a Question type of form.
FindAnswers: The Find Answers section includes the commands necessary to build custom Find Answers pages, including lists of topics, articles, and article details.
For details about each of the commands available from the Variable Picker, see the Form Commands section.

Form Commands

3 Form Commands

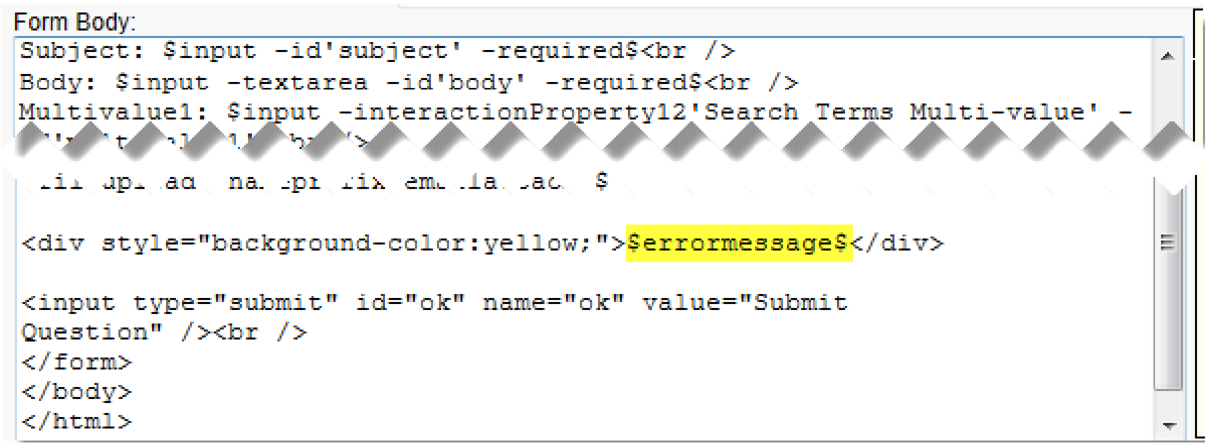
iService forms use commands, the iService Scripting Language, to interact directly with the iService system. These commands are automatically converted into the necessary HTML code for the form. There are two types of commands: those used within the Form Body and those used within submit Actions.

Similar to other commands and variables within iService, form commands are enclosed within a $. For a full list of all available commands , see https://1to1service.iservicecrm.com/f/diag-regex. Many of the more common commands are listed within the forms variable picker in the Admin Tools-Forms page. The forms variable picker will paste the proper syntax into your form for each available property in your tenant.

3.1 $ErrorMessage

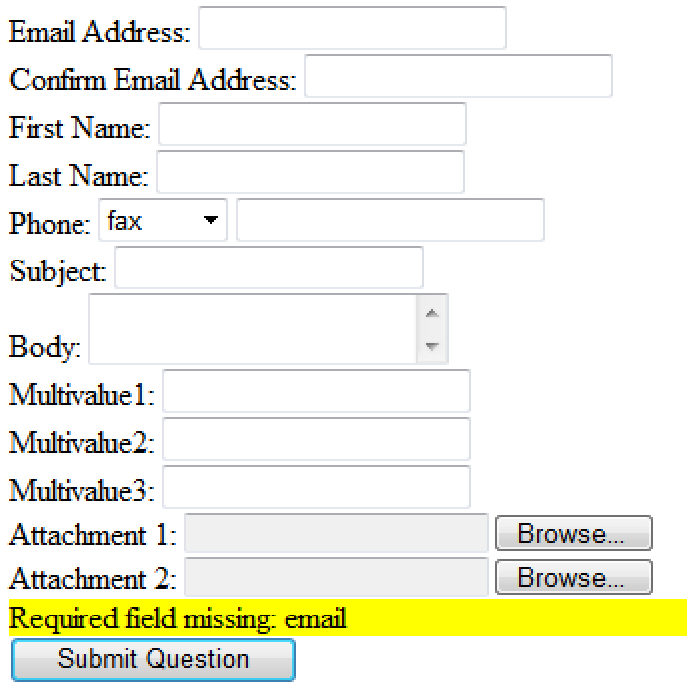
iService forms can perform error checking such as ensuring fields that are marked as required are entered, or confirming the value entered for a field like password matches. when these errors are detected a message can be displayed to the form user with the $Error Message$ command. The error message can be placed anyplace within the form body.

In the example below, the error message is placed just before the submit button and is enclosed within an HTML DIV tag with yellow highlighting. You insert multiple $errormessage$ commands into a form, if desired.


Form Body:

```
Subject: $input -id'subject' -required$<br />
Body: $input -textarea -id'body' -required$<br />
Multivalue1: $input -interactionProperty12'Search Terms Multi-value' -
[illegible]
[illegible] $

<div style="background-color:yellow;">$errormessage$</div>

<input type="submit" id="ok" name="ok" value="Submit
Question" /><br />
</form>
</body>
</html>
```

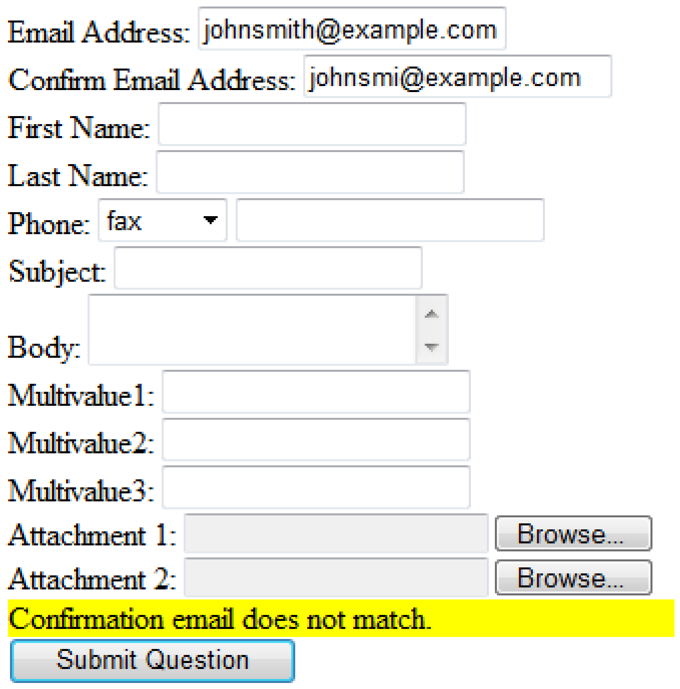
⊟ Example of the error displayed when a required field is not completed.



Email Address:	
Confirm Email Address:	
First Name:	
Last Name:	
Phone:	fax
Subject:	
Body:	
Multivalue1:	
Multivalue2:	
Multivalue3:	
Attachment 1:	Browse...
Attachment 2:	Browse...

Required field missing: email

⊟ Example of the error message when a confirmation field does not match.



Email Address:	johnsmith@example.com
Confirm Email Address:	johnsmi@example.com
First Name:	
Last Name:	
Phone:	fax
Subject:	
Body:	
Multivalue1:	
Multivalue2:	
Multivalue3:	
Attachment 1:	Browse...
Attachment 2:	Browse...

Confirmation email does not match.

3.2 $FileUpload - Adding Attachments

Your form submissions can be modified to accept file attachments by changing the POST command and adding HTML within the form body and using the $FileUpload command.

The Post command for the form should be modified to specify that the form may contain a file attachment. Replace the default post with the following.

```
<form method="POST" enctype="multipart/form-data">
```

This command accepts two parameters: -nameprefix and -group.

Available Parameters for $FileUpload

Parameter Name: -nameprefix

Since every element within an HTML document must have a unique name, you need to add a number or something unique to the name of each file attachment if there is more than one in the form. For example, you might use naming such as file1, file2, file3 where the prefix is the word "file". Or, you could use the naming EmailAttach1, EmailAttach2, etc. where the prefix is EmailAttach. The -nameprefix parameter is used to specify the prefix used.

Example:

Insert the following into your form body wherever you would like the user to select a file. You may include as many file attachment inputs as desired. This example includes two.

```
Attachment 1: <input type="file" name="EmailAttach1" /><br />
Attachment 2: <input type="file" name="EmailAttach2" /><br />
```

3.3 $Header

The $Header command has two use cases: specify the form contains something other than HTML, and allow the form to be loaded in an iFrame.

When you include a form into another form, and the form contains pure CSS, JS, or JSON, you must add the $header variable to the top of the form. This tells the form into which it's included that it has a specific type of content. When including an iService form in an iFrame on another website, we need to tell the browser that it's ok to load our content on their website. These two use cases are explained below.

Specifying the content type

Example:
You have CSS that is used in multiple forms, so you place it into its own form (e.g., Form 2) and reference it in the other forms using the HTML "Link" parameter. You will need to add the $Header command to Form 2 and specify that it is a CSS file.

Inside of Form 1, you would use the link command below to incorporate the CSS from Form 2.
`<link rel="stylesheet" type="text/css" href="/f/2">` **{this brings the CSS from Form 2 into Form 1}**

Inside of Form 2, you would add the following to the beginning of the form.

```
$Header -filetype(CSS)$
```

Available Parameters for $Header - filetype
Parameter Name: -filetype (required parameter)
The -filetype parameter is required to indicate the type of file that is being referenced. The options are CSS (Cascading Style Sheet), JS (JavaScript), and JSON.

Example syntax:
`$Header -filetype(CSS)$` -- this indicates the form is CSS
`$Header -filetype(JS)$` -- this indicates the form is JavaScript
`$Header -filetype(Json)$` -- this indicates the form is Json

Putting forms into iFrames

Example:
You have a contact us form in iService and want to load it using an iFrame on your website.

Inside your iService form, you would use the $header variable as shown below.

```
$Header -allowframeall$
```

It's best practice to include this within a comment to avoid any unintended display in the HTML.

```
<!-- $header -allowframeall$ -->
```

Restricting your form to a specific website
You can prevent your iService form from being loaded in an iFrame from a different domain than iService by using the following version of the header variable. For example, if your iService domain is tenant.iServiceCRM.com, then you can only insert the form into other pages at tenant.iServiceCRM.com when this tag is added.

Example:
```
<!-- $header -allowframesame$ -->
```

3.4 $IF

The if command follows the standard IF/THEN/ELSE logic that is common in most scripting languages. The $IF command is used in conjunction with the $Else$ and $EndIf$ command to form the conditional logic for a page. Multiple $IF commands can be nested together if desired.

The parameters used with the $IF statement (e.g., -loggedin, -myaccountlists, etc.) are evaluated when the iService form loads and specifies the condition that is checked. For example, $IF -loggedin$ will evaluate whether the form was loaded by a user that is logged into iService and has a valid user session. If the user is logged in then the form body after the $IF will be displayed. If the user is not logged into iService, the form body after the $Else$ will be displayed.

The $If command supports various parameters that are useful in different scenarios dependent upon the context of the form. For example, in a customer portal (My Account Form) that presents mailing lists for subscription it can be used to determine whether any mailing lists are available to the logged in contact.

3.4.1 $IF -article

The -article parameter is used to display the details of a Find Answers article. It is used with the Pid (parameter ID) parameter to specify the article for which details will be listed. Since you must supply the form with the ID of the Find Answers article to display, this is usually reference from another form or portion of a form that contains a list of articles with their IDs. Then, when that link is clicked the $IF -article is used to show the details for the selected article.

Basic Example:
```
$IF -article -Pid'articleID'$
     The details for the article specified by articleID would be sho
$Else$
     There is no such article
$EndIf$
```

In the example above, the value for `articleID` would have been passed from another link. However, it is possible to hard code the value for a specific article, if desired, as shown below.

Hard Coded Article Value Example:
```
$IF -article -Pid'95'$
     The details for the article with InteractionID of 95 would be s
$Else$
     There is no such article
$EndIf$
```

To form a fully functional Find Answers page, the $Value -article command is used to present the details for the article selected. See the Find Answers form example using these parameters for more information.

3.4.2 $IF -articlelist

The -articlelist parameter is used to display a list of articles within a Find Answers page. It contains two options that are specified within parenthesis to indicate whether it is a list of articles for a topic (-articlelist(topic)) or for search results (-articlelist(search)). It is used with the -Ptopic (parameter topic) parameter to specify the topic for which articles will be listed.

Since you must supply the form with the ID of the Find Answers topic that is chosen or searched on to display results, this command is usually referenced from another form or portion of a form that contains a list of topics with their IDs. Then, when that link is clicked the $IF -articlelist is used to show the list of articles within the chosen topic.

Basic Example when topic is selected:

```
$IF -articlelist(topic) -Ptopic'topicID'$
     The list of articles specified by Ptopic'topicID' would be show
$Else$
     There are no articles for this topic
$EndIf$
```

In the example above, the value for topicID would have been passed from another link. However, it is possible to hard code the value for a specific topic, if desired, as shown below.

```
$IF -articlelist(topic) -Ptopic'95'$
     The list of articles for the topic with a topic ID 95 would be
$Else$
     There are no articles within topic 95.
$EndIf$
```

To form a fully functional Find Answers page, the $Value -articlelist command is used to present the details for the articles within the selected topics (e.g., article subject, etc.). See the Find Answers form example using these parameters for more information.

3.4.2.1 $IF -Ppagenum

The $If -articlelist command will generate a list of articles, and the -Ppagenum (Page Number Parameter) parameter is used to specify the page of articles that will be displayed. The articles on the page will be dependent upon a) the total number of article created in the list, and b) the number of articles displayed per page as defined by the -Pperpage parameter.

Example:

```
$if -articlelist(topic)  -Ptopic'topicID' -Ppagenum'1' -Pperpage'100
```

The above example would display the first 100 articles, sorted in reverse date order.

3.4.2.2 $IF -Pperpage

The $If -articlelist command will generate a list of articles, and the -Pperpage (Per Page Parameter) parameter is used to specify the number of articles that will be displayed on each page. The -Pperpage parameter should be used with the -Ppagenum parameter to determine which articles are displayed.

Example:

```
$if -articlelist(topic)  -Ptopic'topicID' -Ppagenum'1' -Pperpage'100
```

The above example would display the first 100 articles, sorted in reverse date order.

3.4.2.3 $IF -Psearchid

You can generate a list of find answers articles based on their topic, or based on the results of a keyword search. When generating the results from a keyword search, you use the -PsearchID parameter as shown below.

```
<div>
   <form method="POST" enctype="multipart/form-data">
      $if -fieldregex'topicID'='$^'$
        <input type="hidden" name="topicID" id="topicID" value="1" />
     $endif$
//The section below provides an input box to type a search term//
        <input name="search" id="search" value="$form -id'search'$" /
         <input type="image" id="ok" name="ok" src="http://www.1to1s
        $if -fieldregex'search'='$ *^'$
         <br />
        $else$
        <p>Search Results:</p>
//The $if statement below means "If there are results from the searc
         $if -articlelist(search) -Psearchid'search' -Ptopic'topicID
//The value 'search' within -Psearchid above and below is a referenc
            <table>
               <tr><th>Subject</th><th>Date</th><th>Rating</th><th>V
               $repeat -articlelist(search) -Psearchid'search' -Ptop
               <tr>
                   <td>$value -articlelist(subject)$</td>
                   <td>$value -articlelist(date)$</td>
                   <td>$value -articlelist(rating)$</td>
                   <td>$value -articlelist(viewcount)$</td>
                   <td>$value -articlelist(topicname)$</td></tr>
               $endrepeat$
            </table>
          $else$
            <p>There are no articles.</p>
          $endif$
        $endif$
    </form>
</div>
```

3.4.2.4 $IF -Psort

The `-Psort` parameter is used within a `-articlelist` paramter to specify the sorting for a list of articles. Lists can be sorted in ascending or descending (reverse) order based upon the following fields. Multiple parameters may be used for sorting on more than one column.

SUBJECT,
SUBJECT_REVERSE
DATE
DATE_REVERSE
OPERATOR
OPERATOR_REVERSE
TOPIC
TOPIC_REVERSE
RATING
RATING_REVERSE

COUNT
COUNT_REVERSE

Example: Sort list of articles by subject

```
$if -articlelist(search) -Psearchid'search' -Ptopic'4' -Ppagenum'1'
```

3.4.2.5 $IF -Ptopic

You can generate a list of find answers articles based on their topic, or based on the results of a keyword search. When generating the results from a selected topic, you use the -Ptopic parameter as shown below.

```
<div id="articleList" class="">
//The $IF below says "If there are articles in the reference topic,
   $if -articlelist(topic)  -Ptopic'topicID' -Ppagenum'1' -Pperpage'
      <ul>
//The -Ptopic parameter supplies the ID of the topic for which the l
         $repeat -articlelist(topic) -Ptopic'topicID' -Ppagenum'1' -
           <li><a href="/f/49?articleID=$value -articlelist(id)$">$v
         $endrepeat$
      </ul>
    $else$
       <p>No articles</p>
    $endif$
</div>
```

3.4.3 $If - beforeaction

`$if -beforeaction$` -- True during FormDisplay or before the action during FormSu

3.4.4 $IF -chat

The $If -chat(available) parameter is used to determine whether any iService agents are available to perform Live Chat with customers. The (available) parameter is required with -chat.

The $IF -chat command must be used with a -Ptopic parameter to specify the topic for the chat. This can be hard coded into the chat link, or it can be supplied by the customer when requesting the chat. In order for agents to be available, the following must all be true.

1. The contact requesting the chat has specified a valid topicID for the chat interaction.

2. An agent with segment access to the specified topic is logged into iService AND has set their chat status to available.
3. The available agent has the required Skills for the specified topic.

Example:

```
$If -chat(available) -Ptopic'4'$
  Display your "available" image or chat form
$Else$
  Display your "not available" image or ask a question form
$EndIf$
```

For the case where agents are available, a message (or image) with a link would display offering live chat. When clicked, an iService form can be used to capture contact information before the chat is launched. It's important to capture the user's email address so the interaction history from the chat can be properly archived.

For the case where agents are not available, a message (or image) with a link to an Ask a Question form could be offered to take the customer's message.

Using an iFrame to float an availability image on your website

If you want to float a chat button on top of your page, the code below can be pasted anyplace within the body of your web page. To float the image to the right of the screen change left:0 to right:0.

Example:

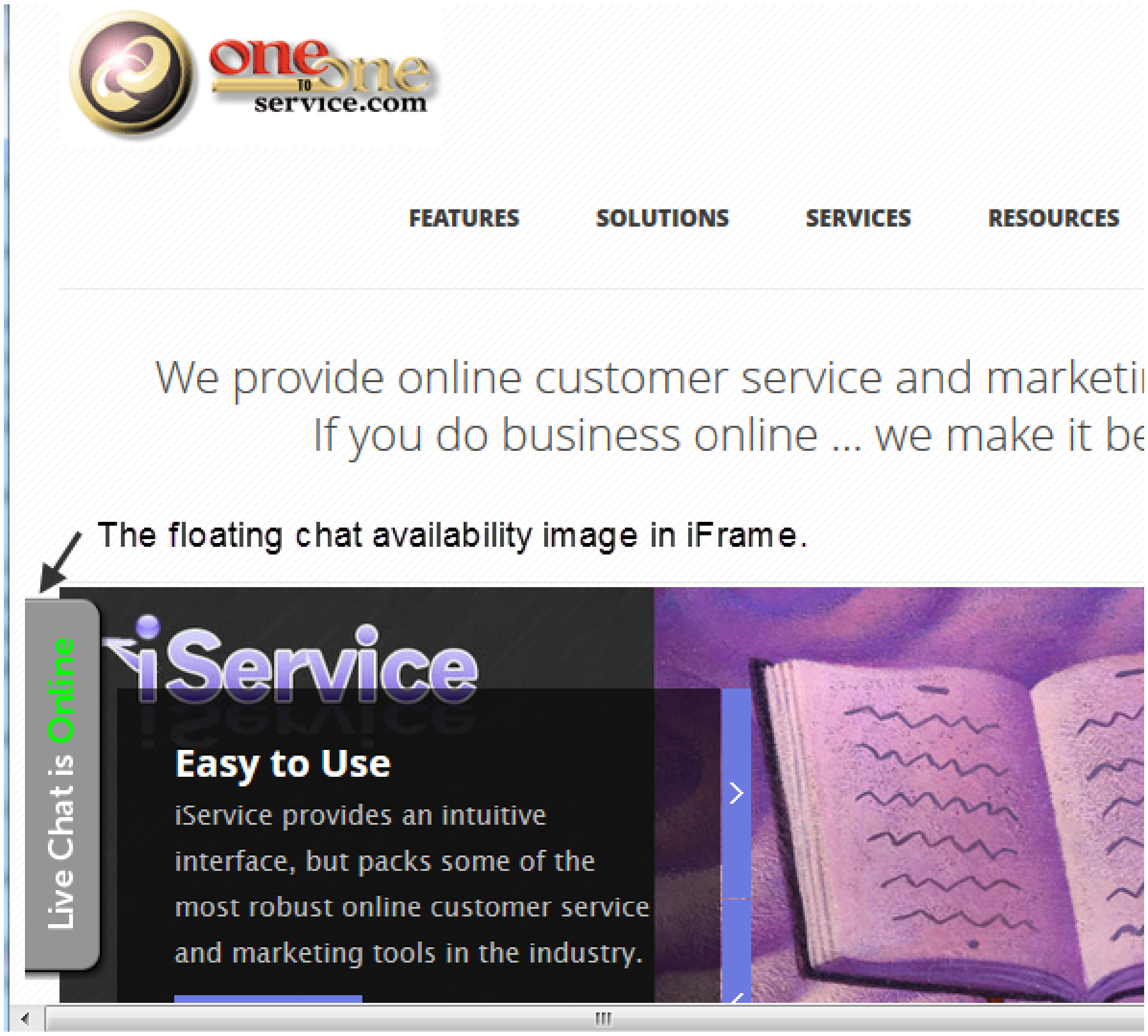
```
<iframe src="https://1to1.iservicecrm.com/f/73" style="posit
```

In the example above, the values are as follows:
src="https... - The source referenced is your iService form that checks for availability (i.e., the $IF -chat(available) code)
style= - The style for the iFrame is what determines where your image will be displayed. The example places the image on the bottom, left corner. The values are:

- position:fixed (the image will not move as you scroll on the page)

- bottom:0 (the image will be fixed to the bottom of the browser screen)
- left:0 (the image will be fixed to the left side of the browser screen)
- z-index:10000 (the image will be on top of other content, assuming it has a z-index of lower than 10000)
- width:55px (the iFrame should be just big enough to display the image and avoid scroll bars)
- height="220" (the height should be slightly larger than your image)
- frameBorder="0" (do not show any border around the image)
- allowTransparency="true" (make the contents transparent so it appears to be part of the page)

The example code shown above would display a chat image as below when agents are available for chat.


oneTOone service.com
FEATURES
SOLUTIONS
SERVICES
RESOURCES
We provide online customer service and marketi
If you do business online ... we make it be
The floating chat availability image in iFrame.
Live Chat is Online
iService
Easy to Use
iService provides an intuitive interface, but packs some of the most robust online customer service and marketing tools in the industry.

⊟ Load Chat Button After Visitor Is On Page For X Seconds

In some cases, you might want to load a chat form if a customer has been sitting on a page for a while. This could indicate they have a problem with the page or question that is causing them to not continue.

The code snippet below can be used with your iService form to accomplish this. In the example below, the 10000 represents milliseconds and will cause the chat button to appear after 10 seconds. You can change this value to whatever you like and insert the link to your own form you want to display.

```
<html> <head> <style type="text/css"> .frame{ display: none; } </style> <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"> </script>
<script type="text/javascript"> $(window).load(function() { setTimeout(function()
{ $('.frame').css('display','block'); }, 10000); }); </script> </head> <body>Hello...
If you're reading this, you're main page containing form has already loaded. You
should see a button in about 10seconds. <div class="frame"> <iframe
src="https://1to1.iservicecrm.com/f/83" style="position:fixed; left:0; z-index: 10000;
width:55px; bottom:0; padding:0; margin:0;" height="220" frameBorder="0"
allowtransparency="true"></iframe> </div> </body> </html>
```

Working Form Examples

Examples of working customer chat forms can be downloaded from the our website at http://1to1service.com/Resources/iServiceAddOns.aspx.

3.4.5 $IF -checked

Used within a repeat to determine if a value is checked. This is required when custom HTML is used because the HTML input tag will not know if the iService value is checked. This currently only works within a $repeat for an input checkbox on subscriptions (Mailing Lists, Campaigns, and Find Answers Articles). This can be used when there is special styling required for the checkbox next to a mailing list, for example, when using JQuery Mobile or other styling that requires HTML.

Example:

```
$repeat -myaccountarticles$
 <tr>
   <td>$value -myaccountarticle(topic)$</td>
   <td>$value -myaccountarticle(subject)$</td>
   <td>$input -id'article' -myaccount(article) -customHTML$</td>
       <input name="article" $If -checked$checked="checked"$EndIf$>
 </tr>
$endrepeat$
```

3.4.6 $IF -fieldregex

The $If -fieldregex parameter is used to evaluate the status of fields within an iService form using regular expression. When the form loads, the enclosed text is either inserted or not depending upon whether the value in the named field is a match for the specified regular expression. This parameter is useful within forms that tend to be reloaded for various reasons, whether because of an error or user input. The expression checking is limited to comparing form field contents to a hard-coded regular expression.

Example 1: Checking values of a submitted form.
If custom HTML is used and a form is submitted with an error because of a missing field, -fieldregex can be used to preserve the values originally entered into a form. This is illustrated below.

```
<label>I have a question about:</label>
<div>$input -id'requesttype' -required -customhtml$</div>
 <select name="requesttype">
   <option value="">Please Choose One</option>
   <option $if -fieldregex'requesttype'='Product'$selected$endif$ va
   <option $if -fieldregex'requesttype'='Service'$selected$endif$ va
   <option $if -fieldregex'requesttype'='Other'$selected$endif$ valu
 </select>
</div>
```

In the example above, the form has an input named requesttype. The -fieldregex parameter determines whether the option was selected when the form was submitted, and sets the value when the page is reloaded by setting the option to "selected" if true. In this example, the $else$ command is not used.

Example 2: Determining whether a topic was selected in a Find Answers form.
Find Answers FAQ forms allow the user to search within a topic, or across all topics. The $IF -fieldregex below is used to determine whether a topic was selected when the search was submitted and return the proper results.

```
$if -fieldregex'topicID'='$^'$
    If the topicID is blank, then search on the entire topic tree.
$else$
    If the topicID is not blank, then limit to search to the specifi
$endif$
```

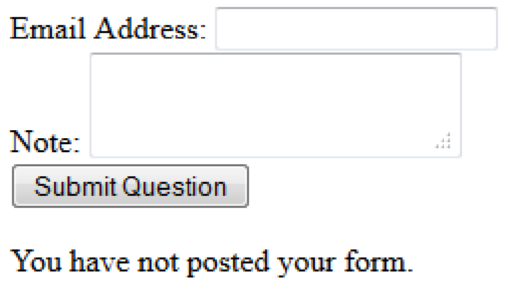
3.4.7 $IF -ispostback

`$IF -isPostBack$` checks whether the user has submitted the current form. When the form is submitted, the page is re-rendered and isPostBack becomes true when it reloads.

This is useful when you want to display a different form body or additional information after a form is posted. A simple example is shown below.

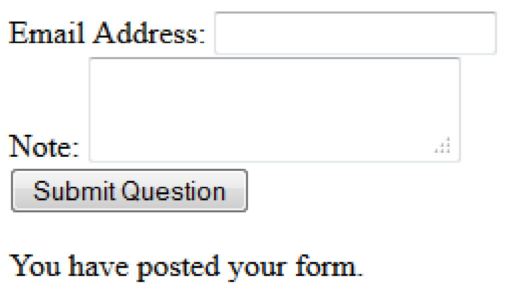
```
<html>
<body>
<form method="POST">
Email Address: $input -email -id'email'$<br />
Note: $input -textarea -id'body'$<br />
<input type="submit" id="ok" name="ok" value="Submit Question" /><br
</form>

$If -isPostBack$
You have posted your form.
$Else$
You have not posted your form.
$EndIf$

</body>
</html>
```

In this example, the form would look like the following before the note was submitted.


Email Address:
Note:

You have not posted your form.

The form would look like the following after the note was submitted.


Email Address:

Note:

Submit Question

You have posted your form.

3.4.8 $If - lasteval

`$if -lasteval$` -- True during FormDisplay, or after action during FormSubmit.

3.4.9 $IF -loggedin

`$IF -loggedin$` checks whether or not the user is logged into iService when loading the form. This is most often used when constructing a login panel for forms that require an agent login.

A simple example of $IF -loggedin$ is shown below.

```
$if -loggedin$      <- Checks to see if user is logged in
   Welcome $value -loggedin(name)$! <- If user is logged in, we display their name
from iService

$else$ <- If user is not logged in we ask them to enter their login details
   Please login:<br />
   LoginName: $loginname -id'loginname'$<br /> <- Ask user to enter their login
   Login Password: $loginpassword -id'loginpassword'$<br /> <- Ask for user’s password
$endif$ <- Marks the end of the IF statement
```

Related Topics:
$LoginName
$LoginPassword
$Value -loggedin

3.4.10 $IF -more

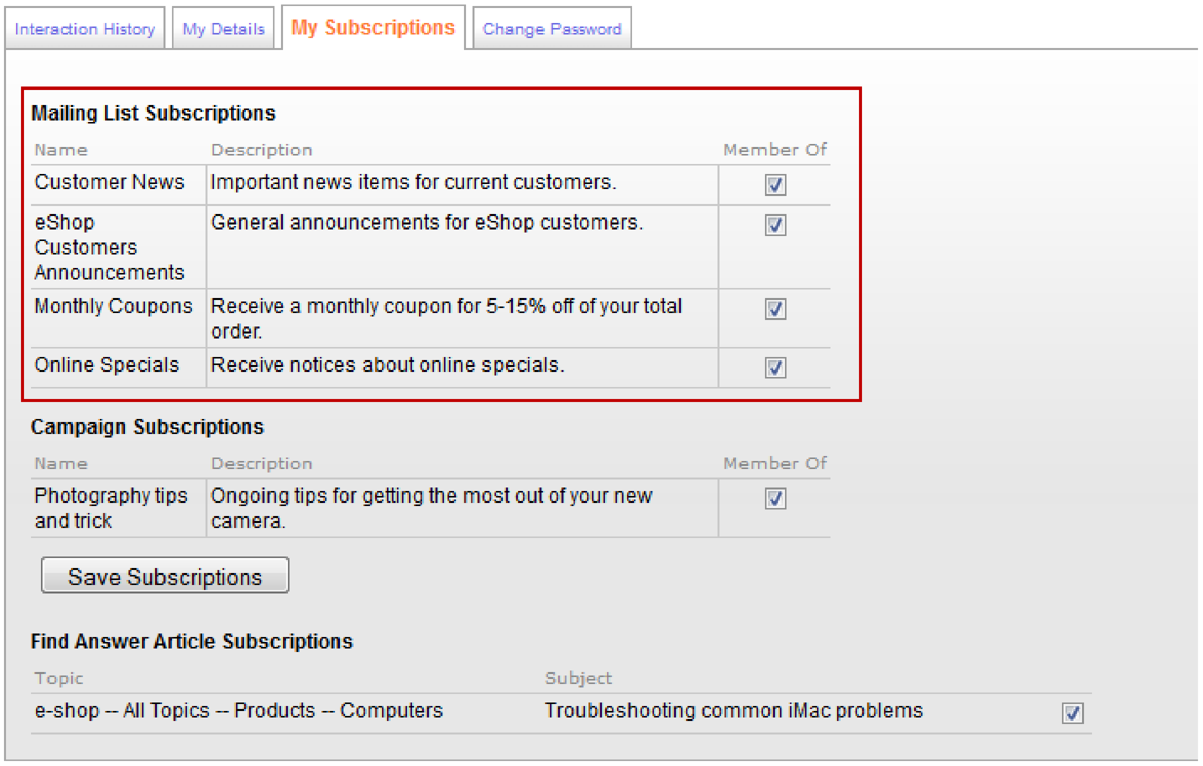
Use with $repeat to indicate whether there are more elements to repeat. The value for this paramter is true for each iteration until the last item repeated, at which point it becomes false.

Example: Apply different formatting to the last item in a list that is generated by $Repeat.

```
$repeat$
   $If -more$
      Place the rows with standard formatting here.
  $Else$
     Place the final row with underlines here.
  $EndIf$
$EndRepeat$
```

3.4.11 $IF -myaccountlists

The -myaccountlists parameter is used with the $IF command to determine whether any mailing lists are available to the logged in user. This presents the same list of mailing lists that the user would see from the /MyAccount.aspx?mode=subscriptions page within iService as shown below.



Mailing List Subscriptions

Name	Description	Member Of
Customer News	Important news items for current customers.	✓
eShop Customers Announcements	General announcements for eShop customers.	✓
Monthly Coupons	Receive a monthly coupon for 5-15% off of your total order.	✓
Online Specials	Receive notices about online specials.	✓

Campaign Subscriptions

Name	Description	Member Of
Photography tips and trick	Ongoing tips for getting the most out of your new camera.	✓

Find Answer Article Subscriptions

Topic	Subject	
e-shop -- All Topics -- Products -- Computers	Troubleshooting common iMac problems	✓

Example:

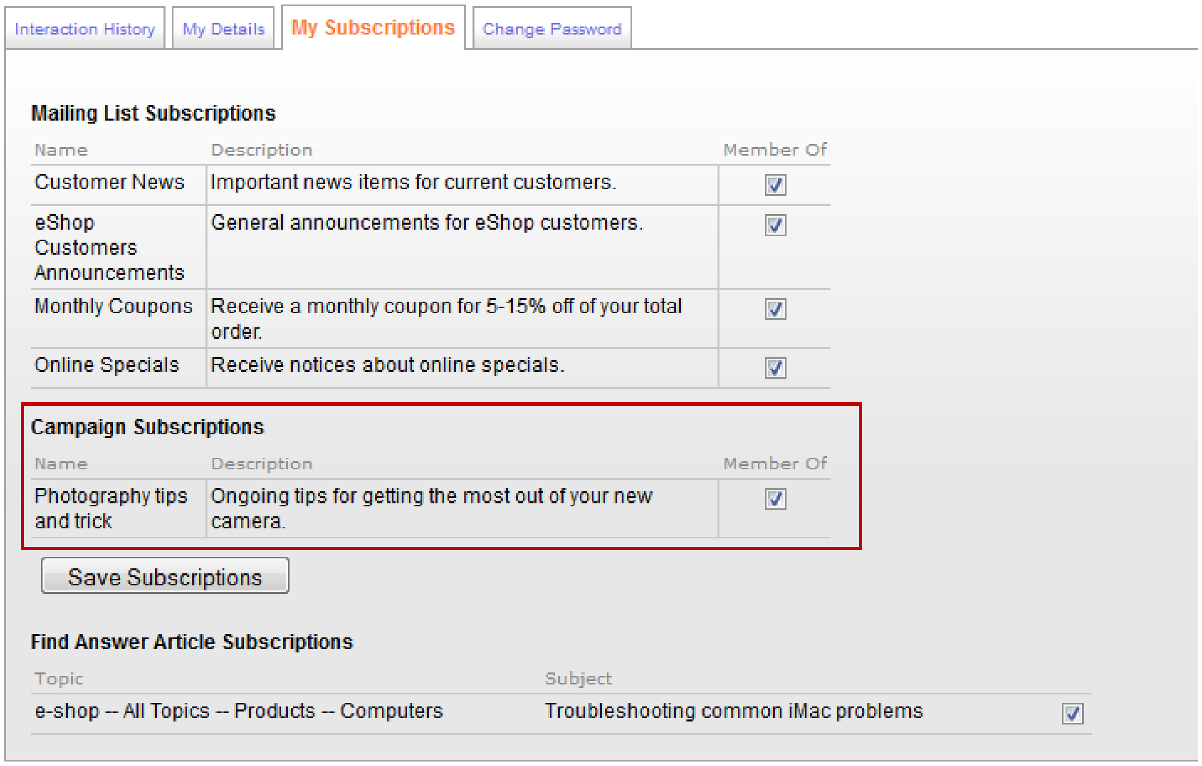
```
$if -myaccountlists$

     Yes, there are lists available to you.
```

```
$else$

     No, there are no lists for you.

$endif$
```

For an example of using -myaccountlists to generate a table of available lists, see the My Account Subscription example.

3.4.12 $IF -myaccountcampaigns

The -myaccountcampaigns parameter is used with the $IF command to determine whether any mailing campaigns are available to the logged in user. This presents the same list of mailing campaigns that the user would see from the /MyAccount.aspx?mode=subscriptions page within iService as shown below.



Mailing List Subscriptions

Name	Description	Member Of
Customer News	Important news items for current customers.	✓
eShop Customers Announcements	General announcements for eShop customers.	✓
Monthly Coupons	Receive a monthly coupon for 5-15% off of your total order.	✓
Online Specials	Receive notices about online specials.	✓

Campaign Subscriptions

Name	Description	Member Of
Photography tips and trick	Ongoing tips for getting the most out of your new camera.	✓

Find Answer Article Subscriptions

Topic	Subject	
e-shop -- All Topics -- Products -- Computers	Troubleshooting common iMac problems	✓

Example:

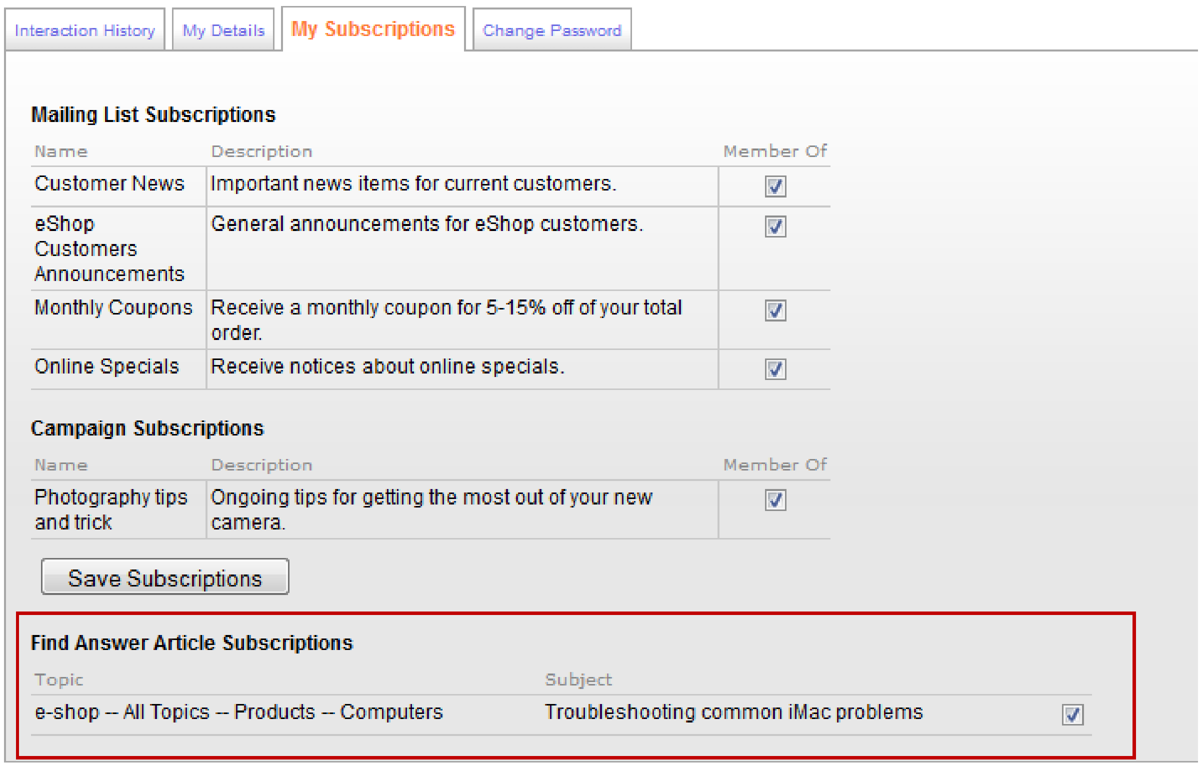
```
$if -myaccountcampaigns$

     Yes, there are campaigns available to you.

$else$
```

```
     No, there are no campaigns for you.

$endif$
```

For an example of using -myaccountcampaigns to generate a table of available lists, see the My Account Subscription example.

3.4.13 $IF -myaccountarticles

The -myaccountarticles parameter is used with the $IF command to determine whether the logged in user is subscribed to any Find Answers articles. This presents the same list of Find Answers articles that the user would see from the /MyAccount.aspx?mode=subscriptions page within iService as shown below.



Mailing List Subscriptions

Name	Description	Member Of
Customer News	Important news items for current customers.	✓
eShop Customers Announcements	General announcements for eShop customers.	✓
Monthly Coupons	Receive a monthly coupon for 5-15% off of your total order.	✓
Online Specials	Receive notices about online specials.	✓

Campaign Subscriptions

Name	Description	Member Of
Photography tips and trick	Ongoing tips for getting the most out of your new camera.	✓

Find Answer Article Subscriptions

Topic	Subject	
e-shop -- All Topics -- Products -- Computers	Troubleshooting common iMac problems	✓

Example:

```
$if -myaccountarticles$

     Yes, you are subscribed to Find Answers articles.

$else$
```

```
     No, you are not subscribed to any Find Answers articles.

$endif$
```

For an example of using -myaccountarticles to generate a table of available lists, see the My Account Subscription example.

3.4.14 $If - Question|Answer|Interaction(attachments)

`$if -q/a/int(attachments)$` -- True if the interaction has attachments. Can be use

3.4.15 $IF -submitsuccess

`$IF -SubmitSuccess$` checks whether a form has been submitted successfully. In order for submitSuccess to be true, all of the following must be true.

1. Pass Input validation - All iService validations (confirm input, required fields, etc.) must pass validation when the form is submitted.
2. All actions must run without error - If there are actions specified on the form, they must run without error.
3. The form must be a post - The form must include an HTML post.

Example 1: Thank you notice
This is useful when you want to present different content on a form after a form is posted. For example, you could present a thank you message after an ask a question form is successfully posted.

```
$IF -SubmitSuccess$
   Thanks for contacting us.
$Else$
   Please submit your request.
$EndIf$
```

Example 2: Multi-step form
A form with multiple steps can be constructed to load a second section after the first section posts successfully.

```
$IF -SubmitSuccess$
   Continue on with the 2nd part of your process.
```

```
$Else$
   Submit the first part of your process here.
$EndIf$
```

3.4.16 $IF -topictree

The $If -topictree parameter is used to generate a list of Find Answers topics in a tree format only if additional topics exist. It is embedded within a $Repeat -topictree(findanswers)$ command to produce a tree structure of topics, and if no sub-level topics exist it will avoid blank lines.

Example:

```
$repeat -topictree(findanswer)$
  $value -topic(name)$
  $value -topic(messagecount)$

  $if -topictree(findanswer)$
    $repeat -topictree(findanswer)$
      $value -topic(name)$
      $value -topic(messagecount)$
    $endrepeat$
  $endif$

$endrepeat$
```

3.5 $Include

The $Include command works similar the .asp include function. It allows you to include one form into another placing the contents of the referenced form directly into the calling form. This allows you to reuse forms that contain common information, or to break large forms into smaller and more workable components. The include command will only insert the form body for the referenced form and will ignore any actions that might be specified on that form.

Available Parameters for $Include

Parameter Name: -formID (required parameter)

The format of the command is $Include -formID'#'$ where # is the ID of the form to be inserted.

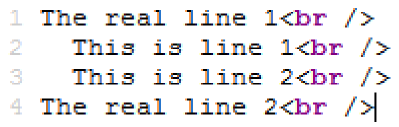
Example:
```
$Include -formID'5'$ -- this will case the form to substitute the bo
```

<u>Parameter Name: -indent (optional parameter)</u>
The -indent parameter generates spacing for the included form HTML when the actual form.aspx page is generated. This can be useful if you are trying to debug or review the code that is generated by the $Include function.

Example 1:
```
$Include -formID'5' -indent'  '$

This example will include two spaces before each line of HTML includ
```

Example 2:
```
The real line 1<br />
$include -formID'5' -indent'  '$
The real line 2<br />
```

If form 5 included two rows of HTML (This is line 1, This is line 2), the form generated would have the following HTML.



```
The real line 1<br />
  This is line 1<br />
  This is line 2<br />
The real line 2<br />
```

Essentially, whatever is entered as the value for the indent parameter will be a prefix for every line within the form that is included.

3.6 $Input

The most common way to capture input from a user is with the $Input command. This command generates HTML INPUT tags to capture input, and makes this input available to the various iService Actions that you specify for the form.

The basic syntax for the $Input command as as follows.

```
$input [-id'description']$
```

The $Input command creates an HTML input box, and the -id description is converted into the HTML ID tag. This ID tag is used to identify the input element, control styling, etc. Each command within an iService form MUST have a unique ID. With a few exceptions, all parameters used in the $Input are structured by beginning with a dash (-) and ending with a description enclosed within singled quotes (').

The syntax for parameters used with $Input is generally -PARAMETER'Description'. The description can be any text desired without spaces, and should be descriptive enough so that it's useful when referencing the command elsewhere within the form. The available parameters are as follows.

Example $Input command that captures values for a contact property using -group and -required

```
$input -contactProperty1'firstname' -id'firstname' -group']
```

- -contactProperty1'firstname' -- the first parameter in this example is a contact property. The – is used to indicate a new parameter, and the term contactProperty1 is used to indicate the input is a contact property. ‘firstname’ is an arbitrary name used to make the command more readable. The reference to 1 in this example is the ID of the property within iService. You can use the variable picker to generate the base $Input command, and it will automatically populate the propertyID number.
- -id’firstname’ -- this second parameter is used to identify the input within the form actions section, and as a reference within the forms cascading stylesheet (CSS) for web designers. It is a unique identifier for the input command and can use any name that you’d like. It should be descriptive of the input (e.g., “firstname”).
- -group – the “group” parameter is used on forms that have multiple submit actions. It allows you to designate the action to which the input is related. When you have a form that creates multiple interactions (e.g.,

two tickets) and do not use the group parameter, the properties will be added to all of the interactions.

- -required – the “required” parameter is optional, and indicates that the user must complete this field before submitting the form. If the form is submitted without a value for this input item, an error message will be displayed.

3.6.1 $Input -addtocampaign

Below is an example of adding a contact to a mailing campaign.

```
$input -addtocampaign1'Test Campaign' -id'campaign'$
```

- -id’campaign’ -- this parameter is used within the form actions section, and as a reference within the forms cascading style sheet (CSS) for the web designer. It is the ID for the HTML input and can be whatever you want.

- `addtocampaign1'Test Campaign'` –the addtocampaign parameter is used to add the contact to the specified mailing campaign. To add the contact to a mass mailing list, use –addtolist. The number (e.g., 1 in this example) is the campaign ID found on the Mailing tab.

The addtolist and addtocampaign commands generate a check box for the user to click indicating they want to be added to this list.

Hiding the Check box

If you want the check box to be automatically checked and not displayed to the user, you can hide the input and use the following modification to your form to automatically check the box.

1. Add the following CSS entry into the HEAD of the form body to hide the paragraph in which you will place the list check box.

```
   p.hide { display: none; }
```

2. Modify the form command to indicate the box is checked.

```
   <form method="POST" name="myform" onSubmit="document.myform.list.c
```

3. Enclose the actual check box for the list subscription within a hidden paragraph.

```
   <p class="hide">$input -addtocampaign1'Test Campaign' -id'campaign'$</p>
```

For an example form used to add a contact to a campaign, see the Form Examples section of this user guide.

3.6.2 $Input -addtolist

Below is an example of adding a contact to a mailing list.

```
$Input -id'List12' -addtolist12'List12'
```

- -id’List12’ -- this parameter is used within the form actions section, and as a reference within the forms cascading style sheet (CSS) for web designer.
- -addtolist12 – the addtolist parameter is used to add the contact to the specified mass mailing list. To add the contact to a mass mailing campaign, use –addtocampaign. The number (e.g., 12 in this example) is the list or campaign ID found on the Mailing tab.

The addtolist and addtocampaign commands generate a check box for the user to click indicating they want to be added to this list.

Hiding the Check box

If you want the check box to be automatically checked and not displayed to the user, you can hide the input and use the following modification to your form to automatically check the box.

1. Add the following CSS entry into the HEAD of the form body to hide the paragraph in which you will place the list check box.

```
   p.hide { display: none; }
```

2. Modify the form command to indicate the box is checked.

```
   <form method="POST" name="myform" onSubmit="document.myform.list.c
```

3. Enclose the actual check box for the list subscription within a hidden paragraph.

```
   <p class="hide">$input -id'list' -addtolist1'CustomerList'$</p>
```

For an example form used to add a contact to a mailing list, see the Form Examples section of this user guide.

3.6.3 $Input -confirm

`[-confirm]` - You can require the user to enter the value for an input box twice to confirm that it is entered correctly. The – confirm parameter is used to capture the second input and match it against the first input. A common example of this is to confirm that the user has entered the e-mail address correctly. the description entered for the – confirm parameter should exactly match the ID of the input that you are confirming. For example, if the -ID of the input for e-mail address is `-id'email'` then the value for this parameter would be `-confirm'email'`.

An example of confirmation for email address is shown below.

```
Email Address: $input -email -id'email' -required$<br />
Confirm Email Address: $input -email -id'emailconfirm' -confirm'emai
```

The second input, which is used to confirm the first entry, has its own unique -id value (-id'emailconfirm'). It uses the -confirm attribute to specify the input that it is confirming (-confirm'email').

3.6.4 $Input -contactProperty

`[-contactProperty#'Description']` - If you want to save the input from your form directly into an existing contact property, such as company name, then add the contact property parameter to your $Input command. The ID of the property is placed after the -contactProperty text (e.g., -contactProperty7 ... where 7 is the ID of the contact property desired).

Example: A Contact's First Name

```
Enter Your First Name: $input -contactProperty1'First Name' -id'Firs
```

In the example above, the first name of the contact is captured and saved into the iService database for the contact.

3.6.5 $Input -customhtml

iService forms will automatically create the HTML necessary for the form based on the form body commands and parameters used. You can customize any of the input commands using the `-customhtml` parameter. The `-customhtml` can be inserted anyplace inside the $Input command. Various examples of using the -customhtml parameter are shown below. When using -customhtml, the custom HTML markup should reference the $Input ID of the element being modified using the "name" parameter.

Converting an Input into a Drop Down List

iService forms will automatically generate an HTML drop down list (select statement) for any contact or interaction properties that are based upon lists (i.e., an Additional Detail). However, you can convert any $Input field into a drop down list and specify the selections using the -customhtml parameter.

```
Year: $input -id'year' -required -customhtml$
<select name="year">
  <option value="">--select a type--</option>
  <option value="1999">1999</option>
  <option value="2000">2000</option>
  <option value="2001">2001</option>
</select>
```

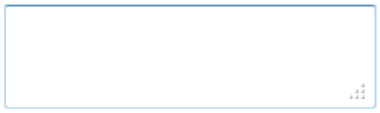
In this example, the ID of the input field is "year" and the HTML select command is used generate a drop down list. Within the HTML code use the "name" parameter to specify the iService $Input command that is being modified. In this example, we used <select name="year"> within the custom HTML to indicate the input is going into the iService $Input command with the id of year.

Customizing a Text Area

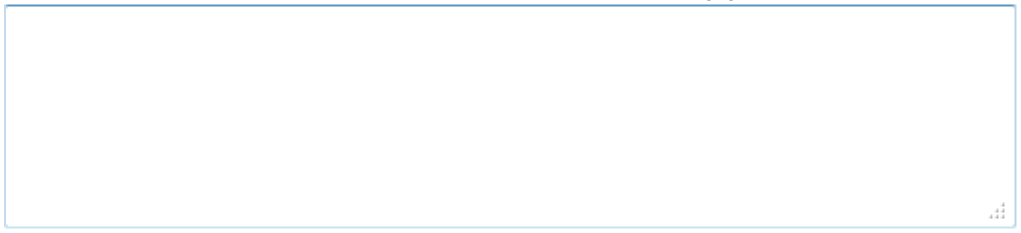
You can create a text area with your $Input command by including the -textarea parameter within the command. This will generate the standard HTML text area without styling. To customize the size and attributes of the text area, use the -customhtml parameter and the HTML textarea command as shown below.

```
Change Details: $input -textarea -id'details' -required -cus
<textarea name="details" cols="60" rows="6"></textarea>
```

Prior to the custom HTML, the text box looks like this.



After this custom HTML, the text box would appear as shown below.



NOTE: If you use a textarea with a contact or interaction property, be sure to limit the amount of input accepted. Interaction Properties within the iService database are limited to 1028 characters of data. Any addition data captured by your text area will be truncated.

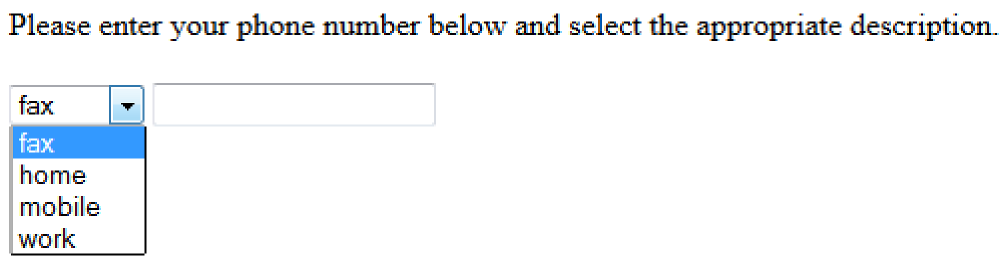
3.6.6 $Input -description

Multi-value properties within iService often contain a description. For example, phone is contact property that can have multiple values. Since there are different types of phone numbers, a description is used with this field to designate the number as home, work, mobile, fax, etc. To set the description for an input, you an either hard code it in the form or allow the user to select it from a list. The -description parameter is only used with an existing $input to specify the description for that input.

Example: Enter a phone number

```
Please enter your phone number below and select the appropriate
description.<br />
<br />
$input -contactProperty4'Phone' -id'phonedescription' -
description'phone'$
$input -contactProperty4'Phone' -id'phone'$
```

The above form body will generate the form shown below.

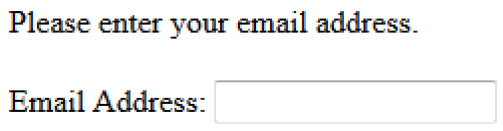

Please enter your phone number below and select the appropriate description.

3.6.7 $Input -email

[-email] - this parameter will save the input as a login for the contact within the iService database. Login / email address is a special field that has its own parameter, but could also be saved using the -contactProperty parameter.

Example: Capture a user's email address
Please enter your email address. <br />
<br />
Email Address: $input -email -id'email'$

The above form body will generate the form shown below.


Please enter your email address.

Email Address:

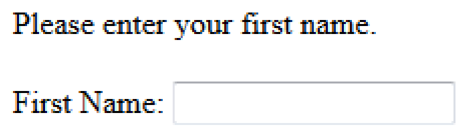
3.6.8 $Input -firstname

[-firstname] - this parameter will save the input as the first name for the contact within the iService database. First Name is a standard field that has its own parameter (-firstname), but could also be saved using the -contactProperty parameter. Since every iService database has this field using the -firstname parameter can be more intuitive than using the -contactProperty parameter.

Example: Capture a user's first name
Please enter your first name. <br />

<br />
Email Address: $input -firstname -id'firstname'$

The above form body will generate the form shown below.



Please enter your first name.

First Name:

3.6.9 $Input -group

`[-group]` - If your form has multiple action steps, such as creating two separate tickets or creating multiple contacts from one submission, use the group parameter with your input. This allows you to specify to which action the input belongs. For instance, you might capture two first names in your form because you are creating two contacts. The -group parameter allows you to tell each CreateContact action which name is associated with it.

3.6.10 $Input -id

Every HTML input on a form must have a unique ID. The -ID parameter is used with the $Input command to specify the ID that will be generated for that input. This ID can be used for styling via CSS. This ID is also referenced within form actions using the syntax $Form -id'xx'$, which the ID from the input is specified in the form action.

Example: Specify an ID for first name

Form Body Text: `$input -firstname -id'first_name'$`

Rendered HTML: `<input id="first_name" name="first_name" type="text" value="" class="text" />`

3.6.11 $Input -interactionProperty

`[-interactionProperty#'Description']` - If you want to save the input from your form directly into an existing interaction property, such as Priority, then add the contact property parameter to your $Input command. The ID of the property is placed after the -interactionProperty text (e.g., -interactionProperty7 ... where 7 is the ID of the interaction property desired).

Example: The relative priority of a question.

```
$input -interactionProperty1'Priority' -id'Priority'$
```

In the example above, the Priority of the question is captured and saved into the iService database for the interaction. The ID of the interaction property in the iService database in this example is 1.

3.6.12 $Input -lastname

[-lastname] - this parameter will save the input as the last name for the contact within the iService database. Last Name is a standard field that has its own parameter (-lastname), but could also be saved using the -contactProperty parameter. Since every iService database has this field using the -lastname parameter can be more intuitive than using the -contactProperty parameter.

Example: Capture a user's last name
Please enter your last name. <br />
<br />
Email Address: $input -lastname -id'lastname'$

3.6.13 $Input -myaccount

[-myaccount] - The -myaccount parameter is used within a My Account Subscriptions form to allows users to subscribe to mailing lists, campaigns, and articles. It generates an input checkbox indicating the user is subscribed to a list.

For an example of usage, see the sample My Account Subscriptions form.

3.6.14 $Input -password

[-password] - The -password parameter instructs the form to mask the users input by adding the HTML type="password" tag. Use this parameter when you want the values typed into the input to be unreadable to users.

3.6.15 $Input -required

`[-required]` - The "required" parameter makes the input field mandatory on the form. If it is left blank, an error will be generated and can be displayed using the $ErrorMessage$ command.

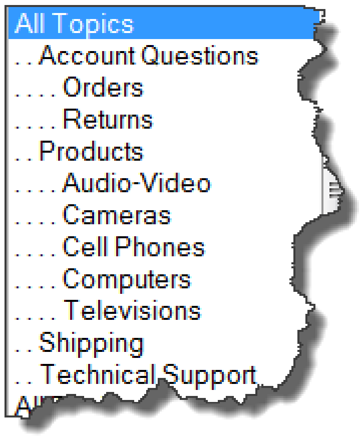
3.6.16 $Input -textarea

[-textarea] - The textarea parameter will convert a $Input into an HTML textarea rather than an input box. Text areas contain scroll bars so multiple lines of text can be entered.

3.6.17 $Input -topics (Topic Tree Parameter)

The command "Public Topics" is used to present a drop down menu of the Topic Tree for the iService website in which the form is located. It is actually a parameter used withinthe $INPUT command.

`$Input -topics  -id'uniquename'$` – This will display a drop down menu where the user can pick a topic. An example is shown below.



The topics displayed are dependent upon the website on which the form is displayed, and the access rights of the user viewing the page. The topics displayed with this command are the same as the user would see on the FindAnswers.aspx page (the standard iService Find Answers tab) on the same website.

3.6.18 Attaching Files

Your form submissions can be modified to accept file attachments by changing the POST command and adding HTML within the form body and using the type="file" input parameter.

The Post command for the form should be modified to specify that the form may contain a file attachment. Replace the default post with the following.

```
<form method="POST" enctype="multipart/form-data">
```

Since every element within an HTML document must have a unique name, you need to add a unique value to the name of each file attachment if there is more than one in the form. For example, you might use naming such as file1, file2, file3 where the prefix is the word "file".

Example:
Insert the following into your form body wherever you would like the user to select a file. You may include as many file attachment inputs as desired. This example includes two.

```
Attachment 1: <input type="file" name="EmailAttach1" /><br />
Attachment 2: <input type="file" name="EmailAttach2" /><br />
```

NOTE: The iService $input command is not used to accept files. Standard HTML input tags are used.

3.7 $JSON

`$json -loginloggedin$` -- Serialize all data returned by LoginLoggedIn web service

3.8 $LoginName

The $LoginName generates an input box for entering your iService login. This is used on forms where an agent login is required to submit the form, such as when creating a ticket. This can be used in connection with the $IF LoggedIn command to optionally present a login box when the user is not already logged into iService.

Available Parameters for $LoginName

Parameter Name: -id

Since the $loginname command generates an input box that accepts user input, it must include an ID to be in a valid HTML format. Use the -id parameter to specify an id for the HTML that is generated.

Example:

```
$loginname -id'loginname'$
```

Parameter Name: -customhtml

The $loginname command generates an input box into which the user will enter their iService login name. This box can be styled using custom HTML if the -customhtml parameter is used.

Example:
```
$loginname -id'loginname' -customhtml$
     Login Name: <input type="text" name="loginname" size="45">
```

Within the custom HTML you must use the name tag (i.e., `name="loginname")` to map back to the ID of the $Loginname command. This example will generate a text box for entering the login name with a width of 45.

Learn more about the -CustomHTML parameter here.

3.9 $LoginPassword

The $LoginPassword generates an input box for entering your iService password. This is used on forms where an agent login is required to submit the form, such as when creating a ticket. This can be used in connection with the $IF LoggedIn command to optionally present a login box when the user is not already logged into iService.

Available Parameters for $LoginPassword

Parameter Name: -id

Since the $loginpassword command generates an input box that accepts user input, it must include an ID. Use the -id parameter to specify an id for the HTML that is generated.

Example:

$loginpassword -id'loginpassword'$

Parameter Name: -customhtml

The $loginpassword command generates an input box into which the user will enter their iService login password. This box can be styled using custom html if the -customhtml parameter is used.
Example:

```
$loginpassword -id'loginpassword' -customhtml$
     Login Name: <input type="text" name="loginpassword" size="45">
```

Within the custom HTML you must use the name tag (i.e., `name="loginpassword")` to map back to the ID of the $Loginname command. This example will generate a text box for entering the login password with a width of 45.

Learn more about the -CustomHTML parameter here.

3.10 $Repeat

The $Repeat command creates a repeating list of values for the type of parameter specified. The $Repeat command is always used with the $Value command to specify what is repeated. It is also often used with the $IF command to display alternate text if there are no $Values to repeat. The syntax for this command is as follows.

```
$Repeat -parameter$
     $value -parameter(name)$
$EndRepeat
```

The types of values repeated will be dependent on the type of form. For example, a find answers page will display a repeating list of topics or articles. A My Account - Subscriptions page will display a repeating list of mailing lists, campaigns, or articles that are available for subscription. An alert auto response will display a repeating list of interaction included within the alert.

Supported parameters for $Repeat

- `$repeat -agents(admin)$` -- Lists all agents in the tenant. Use $value -agent$ inside.$
- `$repeat -alertitems$` -- Lists items in an alert autoresponse. Use $value -interaction$ inside.

- `$repeat -autoresponses(admin)$` -- Lists all auto responses in the tenant. Use $value -autoresponse$ inside.
- `$repeat -contactproperties(admin)$` -- Lists all contact properties in the tenant. Use $value -conactproperty$ inside.
- `$repeat -diag(builtinforms)$` -- Lists the built in forms. Use $value -form$ inside. Example: f/diag-builtinforms-body?formID=diag-builtinforms
- `$repeat -diag(regex)$` -- Lists the regexes for the valid form variables. Use $value -diagregex$ inside. Example: f/diag-builtinforms-body?formID=diag-regex
- `$repeat -forms(admin)$` -- Lists all forms in the tenant. Use $value -form$ inside.
- `$repeat -prefix'...'$` -- -prefix is used with the other options. It's used to help compute the ids of the html generated for $inputs inside the block.
- `$repeat -history(children)$` -- Lists interactions inside a thread. Use inside context that has an interaction, or use -Pid to specify an interaction ID.
- `$repeat -history(threads)$` -- Lists interaction thread roots. Use inside context that has a customer, or use -Pid to specify a contact ID.
- `$repeat -inbox(...)$` -- Lists interactions in the calling agent's inbox.
- `$repeat -intproperties(admin)$` -- Lists all interaction properties in the tenant. Use $value -intproperty$ inside.
- `$repeat -mailboxes(admin)$` -- Lists all mailboxes in the tenant. Use $value -mailbox$ inside.
- `$repeat -messagesearch$` -- Interaction search variable. Used same as find answer $repeat -articlelist$.
- `$repeat -messagesearchfields(properties|statuses|types)$` -- Lists items that go in the dropdowns in the message search page. Use $value -messagesearchfield$ inside. Example: f/diag-builtinforms-body?formID=ngappbuiltin-messagequeue-search
- `$repeat -question/answer/interaction(attachments)$` -- Lists the attachments of an interaction. Use inside context that has an interaction, or use -Pid to specify an interaction ID. Use $value -attachment$ inside.
- `$repeat -segments(admin)$` -- Lists all segments in the tenant. Use $value -segment$ inside.

- `$repeat -stockresponses(admin)$` -- Lists all stock responses in the tenant. Use $value -stockresponse$ inside. Example for all these $repeat -...(admin)$ vars: f/diag-builtinforms-body?formID=ngappbuiltin-admin-js
- `$repeat -stockresponses(agent)$` -- Lists all stock responses for the agent. Use -Pid to specify agent ID.
- `$repeat -stockresponses(segment)$` -- Lists all stock responses for the segment. Use -Pid to specify segment ID.
- `$repeat -userchat(lines)$` -- Lists all lines in a chat. Use -Pid to specify interaction ID for agent form, -Pguid to specify chat guid for user form. Use $value -userchat(line...)$ inside.

3.10.1 Using Repeat Within Find Answers

The Find Answers knowledge base contains frequently asked questions organized by iService topic. The process for retrieving articles is usually by browsing lists of articles or performing a search.

Building a Topics List
When browsing articles, the first step is to select a topic to display all of its articles. The Repeat command can be used to generate this list of topics. Once the article is identified, it can be opened to view the article's details and attachments.

`$repeat -topics(findanswer)$` - Creates a flat list of all topics.
`$repeat -topictree(findanswer)$` - Creates a hierarchical topic tree that can be nested to show parent child relationships. To display multiple levels of the tree, combine as many $repeat commands as needed.

```
$repeat -topictree(findanswer)$
  $value -topic(name)$
     $repeat -topictree(findanswer)$
       $value -topic(name)$
     $endrepeat$
  $endif$
$endrepeat$
```

Building an Article List
Based upon the topic selected, a list of available articles can be presented. Use the -articlelist parameter to display the list of articles.

```
$repeat -articlelist(topic) -Ptopic'nameofinputwithtopicID' -Ppagenu
  $value -articlelist(subject)$
$endrepeat$
```

For a complete list of values that can be displayed, see the $value command.

Composing the Article

The article details are composed using the $IF command, rather than repeat (it is not a repeating list).

```
$if -article -Pid'nameofinputwitharticleID'$
  $value -article(subject) -Pid'nameofinputwitharticleID'$
  $value -article(date) -Pid'nameofinputwitharticleID'$
  $value -article(question) -Pid'nameofinputwitharticleID'$
  $value -article(answer) -Pid'nameofinputwitharticleID'$
$else$
No Article Found
$endif$
```

For a complete list of values that can be displayed, see the $value command.

3.10.2 Using Repeat Within My Account

The subscriptions page uses a combination of $IF and $repeat. The if statement is used in the event there are no mailing lists to display, and the repeat command generates a repeating list.

```
$if -myaccountlists$
  $repeat -myaccountlists$
    $value -myaccountlist(name)$
    $value -myaccountlist(description)$
    $input -id'enteruniquenamehere' -myaccount(list)$ <-- Creates a
  $endrepeat$
$else$
No Mailing Lists
$endif$
```

3.11 $Stock

The $Stock command is used to insert a stock response into auto responses, forms, etc. It can be used in the following places:

- Agent Email
- Note
- Ticket
- Agent Response (My Queue or Supervise)
- Auto Response
- Form
- Mass Mailing Message
- Another Stock Response

Format for $Stock

```
$Stock -id'#' -name'Name of the Response'$
```

The stock response can be identified with either the -id, -name, or both parameters.

Parameter Name: -id

Example:

```
$Stock -id'7'$
```

Parameter Name: -name

Example:

```
$Stock -name'My Footer'$
```

Using both parameters

Example:

```
$Stock -id'7' -name'My Footer'$
```

Using both parameters is typically recommended as it makes the command easier to read.

3.12 $Value

The value command is used for defining text to display within the form body. The options available depend upon the type of information being displayed. If there are no values to display then nothing will be displayed when the command is interpreted.

Values For The User Logged Into iService

If a user is logged into iService when using a form, the following commands can be used to display values about their identify.

```
  $value -loggedin(name)$ <- Displays the name of the user if logged
  $value -loggedin(contactid)$ <- Displays the contact ID of the use
  $value -loggedin(sessionid)$ <- Displays the session ID of the use
```

Values For The Form Itself

The -formID parameter is used to insert the ID of the current form into the form body. This is especially useful when you create a form or form component that you want to use in several places. Rather than hard code the ID of the form, simply insert $value -formID$ into your form body and it will be converted into the actual form ID number when the page is rendered.

```
  $value -formid$ <- Displays the ID of the form that is being displ
```

Find Answers Values

Within the customized find answers page, the following are available.

<u>Topics (requires a $repeat command)</u>

```
  $value -topic(id)$ <- Displays the ID of the selected topic
  $value -topic(parentid)$ <- Displays the ID of the selected topic'
  $value -topic(name)$ <- Displays the name of the selected topic
  $value -topic(messagecount)$ <- Displays the number of unanswered
  $value -topic(segmentid)$ <- Displays the ID of the segment of the
  $value -topic(segmentname)$ <- Displays the name of the segment th
  $value -topic(visibility)$ <- Displays whether the selected topic
```

<u>Article Lists (requires a $repeat command)</u>

```
  $value -articlelist(id)$ <- Displays the ID of the selected articl
  $value -articlelist(subject)$ <- Displays the subject of the selec
  $value -articlelist(date)$ <- Displays the date created of the sel
  $value -articlelist(topicid)$ <- Displays the ID of topic of the s
  $value -articlelist(topicname)$ <- Displays the name of the topic
  $value -articlelist(rating)$ <- Displays the average rating for th
  $value -articlelist(viewcount)$ <- Displays the number of views of
  $value -articlelist(public)$ <- Displays whether the selected arti
```

<u>Article Details (note: you can select any name desired for the Pid of the article detail because it is an input from the form, or hard coded ID)</u>

```
  $value -article(id) -Pid'xx'$ <- Displays the ID of the selected a
  $value -article(subject) -Pid'xx'$ <- Displays the subject of the
  $value -article(date) -Pid'xx'$ <- Displays the date created of th
  $value -article(question) -Pid'xx'$ <- Displays the question of th
  $value -article(answer) -Pid'xx'$ <- Displays the answer of the se
  $value -article(public) -Pid'xx'$ <- Displays whether the selected
  $value -article(segmentid) -Pid'xx'$ <- Displays the ID of the sel
  $value -article(segmentname) -Pid'xx'$ <- Displays the name of the
  $value -article(topicid) -Pid'xx'$ <- Displays the ID of the topic
  $value -article(topicname) -Pid'xx'$ <- Displays the name of the t
  $value -article(viewcount) -Pid'xx'$ <- Displays the number of vie
  $value -article(rating) -Pid'xx'$ <- Displays the average rating f
  $value -article(myrating) -Pid'xx'$ <- Displays the rating of the
  $value -article(creatorname) -Pid'xx'$ <- Displays the name of the
```

The following $value commands are used when displaying article details, but require their own $repeat parameter ($repeat -attachments -Pid'xx'$).

```
  $value -attachment(id)$ <- Displays the ID of the attachment(s)
  $value -attachment(name)$ <- Displays the name of the attachment(s
  $value -attachment(ownerid)$ <- Displays the ID of the article tha
```

My Account Subscriptions Values

Within the customized subscriptions page, the following are available.

<u>List of Mailing Lists (requires a $repeat command)</u>

```
  $value -myaccountlist(id)$ <- Displays the ID of the selected mail
  $value -myaccountlist(name)$ <- Displays the name of the selected
  $value -myaccountlist(description)$ <- Displays the description of
```

<u>List of Campaigns (requires a $repeat command)</u>

```
  $value -myaccountcampaign(id)$ <- Displays the ID of the selected
  $value -myaccountcampaign(name)$ <- Displays the name of the selec
  $value -myaccountcampaign(description)$ <- Displays the descriptio
```

<u>List of Find Answers Articles To Which The User Is Subscribed (requires a $repeat command)</u>

```
  $value -myaccountarticle(id)$ <- Displays the ID of the selected f
  $value -myaccountarticle(subject)$ <- Displays the subject of the
  $value -myaccountarticle(topic)$ <- Displays the topic of the sele
```

3.12.1 $Value -alert

The -alert parameter is used to display the name of an alert and the count of its messages within an Alert Auto Response.

Available parameters for the $value -alert()$ command are:
<u>Name</u> - Display the name of the alert within an alert auto response using `$value -alert(name)$`.
<u>Count</u> - Display the number of messages included within an alert auto response using `$value -alert(count)$`.

An example of how this might be used within an alert auto response is shown below.

```
This is an automated alert from iService which was generated to in
Name of alert: $value -alert(name)$
Number of items in this alert: $value -alert(count)$
```

3.12.2 $Value -alertitem

Use the -alertitem parameter to display the details of an interaction within an alert auto response.

The following elements of an interaction can be displayed within the auto response.
<u>ID</u> - To display the interaction ID for the message, use `$value -alertitem(id)$`.
Date - To display the date the interaction was created in iService, use `$value -alertitem(date)$`.
<u>Time</u> - To display the time the interaction was created in iService, use `$value -alertitem(id)$`.
<u>Agent</u> - To display the iService agent currently assigned to the interaction, use `$value -alertitem(agent)$`.

Name - To display the name of the contact, use `$value -alertitem(name)$`.
Email - To display the email address of the cotnact, use `$value -alertitem(email)$`.
Subject - To display the subject line of the interaction, use `$value -alertitem(subject)$`.

The -alertitem parameter must be used within a $repeat command to generate the repeating list of values. A simple example of this command is as follows.

```
$repeat -alertitems$
     Date of Message: $value -alertitem(date)$
     Subject of Message: $value -alertitem(subject)$
     From: $value -alertitem(name)$
$endrepeat$
```

NOTE: When using this command within and HTML table, the $repeat command must be marked as a comment. Since it is embedded within an HTML table, your browser will remove the tag if it is not commented because placing it between a table parameter and row parameter is an invalid location for HTML code.

```
<table border="0" cellpadding="3" cellspacing="1" align="left" width
<thead>
       <tr>
         <th>Date of Original Msg</th>
         <th>Interaction ID</th>
         <th>Contact Name</th>
         <th>Contact Email</th>
         <th>Subject</th>
         <th>Assigned Agent:</th>
       </tr>
 </thead>

<tbody>
       <!-- $repeat -alertitems$ -->
      <tr>
         <td>$value -alertitem(date)$ $value -alertitem(time)$</td>
<!--Edit the line below for your iService URL-->
         <td><a href="https://1to1service.iservicecrm.com/MessageQue
         <td>$value -alertitem(name)$<br></td>
         <td>$value -alertitem(email)$</td>
         <td>$value -alertitem(subject)$</td>
         <td>$value -alertitem(agent)$</td>
      </tr>
      <!-- $endrepeat$  -->
 </tbody>
</table>
```

3.12.3 $Value -article

The -article parameter is used to display the various parts of a find answers article. The syntax for the command is:

`$Value -article(part)$` where part is the aspect of the article that you want to display.

Example: Display the subject of an article
`$value -article(subject)$`

The options include:
`(id|subject|date|question|answer|public|segmentid|segmentname|topici`

3.12.4 $Value -articlelist

The -articlelist parameter is used to display the various parts of a find answers article within an article list. This list is generated from a search or from a selected topic in a list of find answers topics.
The syntax for the command is:

`$Value -articlelist(part)$` where part is the aspect of the article that you want to display within the list.

Example: Display the subject of an article within a list of articles
`$value -articlelist(subject)$`

The options include:
(id|subject|date|topicid|topicname|rating|viewcount|public)

A more complete example of usage is shown below.

```
<div id="searchArea">
   <form method="POST" enctype="multipart/form-data">
      $if -fieldregex'topicID'='$^'$
        <input type="hidden" name="topicID" id="topicID" value="1" />
     $endif$
        <input name="search" id="search" value="$form -id'search'$" /
     $if -fieldregex'search'='$ *^'$
         <br />
        $else$
          <p>Search Results:</p>
              $if -articlelist(search) -Psearchid'search' -Ptopic'to
                 <table>
                  <tr><th>Subject</th><th>Date</th><th>Rating</th><t
                   $repeat -articlelist(search) -Psearchid'search' -
                   <tr>
                      <td>$value -articlelist(subject)$</td>
                      <td>$value -articlelist(date)$</td>
                      <td>$value -articlelist(rating)$</td>
                      <td>$value -articlelist(viewcount)$</td>
                      <td>$value -articlelist(topicname)$</td>
                    </tr>
                    $endrepeat$
                  </table>
                $else$
                  <p>There are no articles.</p>
                $endif$
      $endif$
   </form>
</div>
```

The above find answers search generates a list of articles that includes columns for the article subject, date, rating, viewcount, and topic name.

3.12.5 $Value -attachment

The -attachment parameter is used to display attachments with find answers articles. Articles may have files attached (Excel sheets, PDF documents, etc.), and these attachments can be displayed using the -attachments parameter.

The syntax for the command is:
`$Value -attachment(part)$` where part is the aspect of the file attachment that you want to display within the find answers article.

Example: Display the name of a file attached to a find answers article
`$value -attachment(name)$`

The options include:
(id|name|ownerid)

The `$value -attachments` command requires the use of the `$IF -attachments` and the `$Repeat -attachments` to generate the desired list of files attached to an article. A simple example of this is shown below.

```
<div>
   $if -attachments -Pid'articleID'$
      $repeat -attachments -Pid'articleID'$
         <div data-role="content"><!-- Attachment List Repeat -->
            <div>
               //The anchor tag below generates the clickable link fo
               <a href="File.aspx?interactionID=$value -article(id) -
            </div>
         </div><!-- /Attachment List Repeat -->
      $endrepeat$
   $else$
      There are no file attachments for this article.<br />
   $endif$
</div>
```

3.12.6 $Value -diagregex

The DiagRegex parameter is a special value used to display the regular expression list supported by iService. It is used within a built in form that exists within every iService installation, and can be found at the following url:

https://1to1service.iservicecrm.com/f/diag-regex

This form can be used as a short cut to all of the form variables and is updated with each release of iService.

3.12.7 $Value -customer|-question|-answer

This command is used to display customer information from the iService database. The command is used when constructing auto responses, making stock responses for agents, and other scenarios where you want to automatically insert information from iService. It can also be used within the Form Body of an iService form to customize content for your support portals and workflow applications.

Reserved Names vs. Property IDs
There are a few common values for contacts and interactions that can be specified by referencing their reserved name. Other values can be accessed using their property ID or property name.

Reserved Names
The format for specifying the value of a reserved name is

```
$value -customer(ReservedName)|-question(ReservedName)|-
answer(ReservedName)$
Example: $value -customer(company)$
```

The reserved property names and their usage are:
[-Customer names]

- `$value -customer(name)$` - displays the First Name and Last Name of the contact. If the contact doesn't have a name specified it will show their email address.
- `$value -customer(company)$` - displays the contact's Company.
- `$value -customer(firstname)$` - displays the contact's First Name only.
- `$value -customer(lastname)$` - displays the contact's Last Name only.
- `$value -customer(email)$` - displays the contact's email/login. If they have more than one specified, it will only display the first.
- `$value -customer(password) $` - displays the contact's password in clear text. This only functions within the Contact Creation auto response and is used to provide a new registrant with their auto generated password.

[-Question | -Answer | -Interaction names]

- `$value -question|-answer|-interaction(`see below`)$`

The full list of values available is shown below.

```
id
ref
type
status
mailbox
```

```
fromemail
toaddress
cc
bcc
date
datetime
time
customerID
customername
topicID
topicname
segmentID
segmentname
subject
body
bodyhtml
bodyplain
quotedbody
note
assignedToID
operatorname
attachments
statusaudit
answerproperties
questionproperties
```

Using the Property Parameter

All other properties can be referenced by the property ID or property name using the format -customer(property#) or -customer(property'propertyname'). The syntax for -question and -answer are identical to -customer.

Examples of displaying other properties are below.

`$value -customer(property7)$` - where the ID of the contact property is 7.

`$value -customer(property'special sauce')$` - where the name of the contact property is special sauce.

`$value -question(property7)$` - where the ID of the question interaction property is 7.

The -customer|-question|-answer parameters can be used in two different scenarios: when iService knows the contact, and when you specify the ID of the customer/question/answer using the -Pid parameter.

<u>Scenarios Where iService Knows the Contact ID</u>

In the following scenarios, iService knows the contact based on the context of the page that is being submitted. In these cases you do not need to specify the contact ID using the -Pid parameter because iService already knows who they are. These include:

- Auto Responses
- Agent Responses
- Notes
- Agent Emails
- Tickets
- Mass Mailing Messages

<u>Scenarios Where iService Does Not Know the Contact ID</u>

In most forms, iService will not know the context of the contact unless the user is logged into iService, or the contact ID is specified. To specify the ID of the contact, use the -Pid parameter. When displaying multiple properties for the same contact you only need to include the -Pid for the first property.

Example: Display contact properties in a form for contact ID 10.

```
Name: $value -customer(name) -Pid'10'$<br />
Company: $value -customer(company)$<br />
First Name: $value -customer(firstname)$<br />
Last Name: $value -customer(lastname)$<br />
Email Address: $value -customer(email)$<br />
```

Example: Display properties in a form with ID passed from URL

In most cases, the ID of the contact will be passed to form body rather than hard coded. One way to do this is to pass the ID in the URL string and use the $form -id$ parameter to incorporate it into the form body.

URL Passed: `http://iservicecrm.com/f/5?contactID=10`

In the above URL, form number 5 is loaded and the value 10 is passed for field named contactID.

Syntax For Using The Value Passed In The URL: `$value -customer(name) -Pid'$form -id'contactID$')`

In the example above, the value of 10 would be substituted in the form body for contactID, resulting in a $value command of `$value -customer(name) -Pid'$10')`

3.12.8 $Value -formID

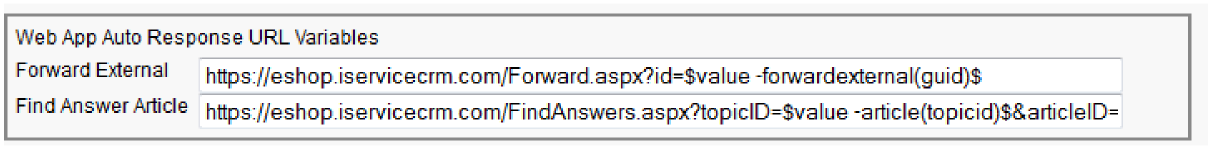
The -formID parameter displays the ID of the current form.

Example:
`$value -formID$` - This $value -formID$ command will display the ID of the form in which it is inserted. For example, when inserted into form 3 this command will return the value "3".

3.12.9 $Value -forwardexternal

The -forwardexternal parameter is used within an Agent Notification Auto Response. It supplies the details for the auto response, such as the agent that forwarded the message, their comments, an external answer page URL, etc. The following options are supported.

url - generates a URL that can be clicked to open the external answer page.
id - the reference number of the interaction that was forwarded
guid - This parameter is used within the Admin Tools - Website page to specify the Forward External URL. When variables are inserted into your forward external auto response template, a URL must be generated (e.g., the Forward External variable that creates a link for answering a question). You must specify the URL to be used in this section, which is typically the domain followed by the URL as shown below.



Web App Auto Response URL Variables

Forward External	https://eshop.iservicecrm.com/Forward.aspx?id=$value -forwardexternal(guid)$
Find Answer Article	https://eshop.iservicecrm.com/FindAnswers.aspx?topicID=$value -article(topicid)$&articleID=

agentname - inserts the name of the agent that submitted the Forward External message.
agentemail - inserts the email address of the agent that submitted the Forward External message
agentcomments - inserts the comments from the agent that submitted the Forward External message

An example of these parameters within an auto response is shown below.

> A new message has been submitted through our support website and you have been asked to provide the response. Details about the person that assigned this to you and their notes are shown below.
>
> Details From Assigning Agent
> **Agent that assigned this to you:** `$value -forwardexternal(agentname)$`
> **Email address of agent that assigned this to you:** `$value -forwardexternal(agentemail)$`
> **Notes from agent (if any):** `$value -forwardexternal(agentcomments)$`
>
> You can answer this by clicking the link below, or copying it into your browser:
> `$value -forwardexternal(url)$`

3.12.10 $Value -interaction

The -interaction parameter functions similar to the -question and -answer parameter, but is used in cases where iService does not know the context. For example, within a form to list the subject and body of an interaction from the iService database. Within auto responses, the -question and -answer parameters would be used.

See -question|-answer for details

3.12.11 $Value -loggedin

The -loggedin parameter displays values for the user that is logged into iService when they load the form. This parameter can display the users name, contactID, and sessionID.

Syntax:
```
$value -loggedin(name|contactID|sessionID)$
```

Examples:

`$value -loggedin(name)$` - Displays the full name of the user as recorded in iService.

`$value -loggedin(contactID)$` - Displays the user's contact ID from iService.

`$value -loggedin(sessionID)$` - Displays the session ID for the user from iService.

3.12.12 $Value -myaccountarticle

This parameter is used to display values for a list of Find Answers articles to which a contact has subscribed. It is used within a custom My Account - Subscriptions page.

See the My Account - Subscriptions Form example for details on usage.

3.12.13 $Value -myaccountcampaign

This parameter is used to display values for a list of mailing campaigns to which a contact has subscribed. It is used within a custom My Account - Subscriptions page.

See the My Account - Subscriptions Form example for details on usage.

3.12.14 $Value -myaccountlist

This parameter is used to display values for a list of mass mailing lists to which a contact has subscribed. It is used within a custom My Account - Subscriptions page.

See the My Account - Subscriptions Form example for details on usage.

3.12.15 $Value -notifyresponse

The -notifyresponse parameter is used to generate an ID and response body when sending questions to external users for help answering a question. Within the iService Message Queue, agents can use the Forward to External agent option to send a question to an outside person for help. The agent selects an auto response template which must include an ID and a response body. This auto response must include the ID parameter and one of the body parameters.

Syntax:
`$value -notifyresponse(id|body|bodyatstart)$`

Examples:
`$value -notifyresponse(id)$` - Displays the ID that is used to tie the response back to the original question. The notify id is displayed within the message using this format:
[Notify#: 199311]

`$value -notifyresponse(body)$` - Displays the original question within a Begin and End marker. The external agent will then type their response between these markers and send the response back to iService.
This response body would look similar to the following:

```
==== Begin Answer ==== d1c4cab8-8657-43f3-b8e4-a309f3644786

On 2/20/2013 9:42:35 PM, Peter Major <pmj@domain.com> wrote:

Hi -

Thank you again for your help.

I was able to log in using your instructions. However, when I try to download the
attachment, I am taken to a blank page

- Peter

==== Answer End ==== d1c4cab8-8657-43f3-b8e4-a309f3644786
```

`$value -notifyresponse(bodyatstart)$` - Displays the original question without a Begin marker. The external agent may then type their response immediately after clicking reply and will not need to type between the begin and end markers, as they do with the body parameter. When using this parameter, the `$value -notifyresponse(bodyatstart)$` command must be the first entry within the auto response. It will generate content similar to the example above but without the ===Begin Answer === label.

3.12.16 $Value -now

The parameter -now returns the current date and time.

Example: `$value -now$` might return 3/7/2014 1:19:20 PM

3.12.17 $Value -passwordreset

The -passwordreset parameter is used to generate a password reset link that a contact can use to reset their password. The contact clicks on the generated link and will be taken to a page where they can set a new password.

Syntax:
```
$value -passwordreset(url|ipaddress)$
```

Examples:

`$value -passwordreset(url)$` - Displays the URL that will allow the contact to change their password. This URL will include a GUID that is specifically tied to the contact, and is valid for 10 minutes.

`$value -passwordreset(ipaddress)$` - Displays the IP address of the user that requested the password reset within iService. The contact must be at this IP address when they visit the password reset URL. This prevents someone from requesting a password reset and someone else from completing the request. It also provides evidence about the machine from which the request was made.

3.12.18 $Value -Pid

-Pid is used to specify the ID for a parameter (i.e., the Parameter ID). It allows you to specify the ID for a contact or interaction property and load its parameters.

Syntax: -Pid'x' where x is the desired ID
Example: Use -Pid to load the subject line of an interaction.

```
$value -interaction(subject) -Pid'ref'$
```

In the example above, a user would input the value for ref within an input field, and it would be passed to this $value command to display the subject.

Examples: Use -Pid to load the first name of a contact.

```
$value -customer(firstname) -Pid'25044'$
```

The example above would display the first name for the contact whose ContactID in iService is 25044.

3.12.19 $Value -repeat

The `$Value -repeat` command is used to generate a repeating list of values.

Options: `$value -repeat(prefix|index|count)`
Prefix - To place a prefix at the start of any $Input HTML name value, use this parameter.
Index - The count parameter will specify the number of times to repeat.
Count - Provides the current iteration number of the repeat, from 0 to count -1.

3.12.20 $Value -tenant

The -tenant parameter displays values for the tenant that is hosting the iService form. This parameter can display the tenant name and the tenant ID.

Syntax:

```
$value -tenant(name|ID)$
```

Examples:

`$value -tenant(name)$` - Displays the name of the tenant as recorded in iService in the Management Console.

`$value -tenant(ID)$` - Displays the tenant's ID from iService.

3.12.21 $Value -today

The -today parameter displays the current date at the time the form is loaded. The date is displayed in MM/DD/YYYY format.

Example:

`$value -today$` - This would display a value similar to 6/3/2013.

3.12.22 $Value -topic

The $value -topic parameter is used with the $repeat command to display a list of topics from your iService database.

Options: id|parentid|name|messagecount|segmentid|segmentname|visibility

Example: Display a list of topics

```
$repeat -topics(findanswer)$
  $value -topic(id)$
  $value -topic(parentid)$
  $value -topic(name)$
  $value -topic(messagecount)$
  $value -topic(segmentid)$
  $value -topic(segmentname)$
  $value -topic(visibility)$<br />
$endrepeat$
```

The result of the above without formatting might look like the following.

```
4  All Topics   18  3   Segment1   public
6  4 _Spam        0  3 Segment1    private
5  4 _Undeliverable  0  3  Segment1   private
```

3.12.23 $Value -version

The -version parameter displays the iService version number. The version is displayed as it is within the footer of the standard iService web interface.

Example:

`$value -version$` - This would display a value similar to 6.2 - 2013.05.15 - 14:26:14.

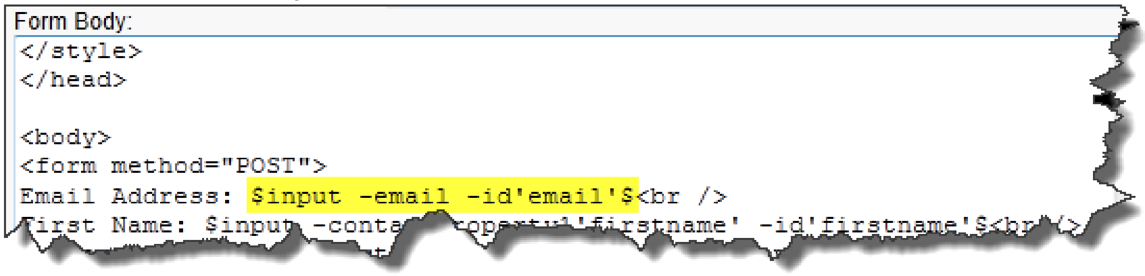
3.13 $Form

When you create iService forms, the form body can be customized to capture any type of input using the $Input command. This allows the user of the form to input their information, and make it available for use in the action commands. Within the action command, you use the $Form command to specify the values that have been input to the form.

The syntax for the $Form command is as follows.

```
$form -id'ID of the Input'$
```

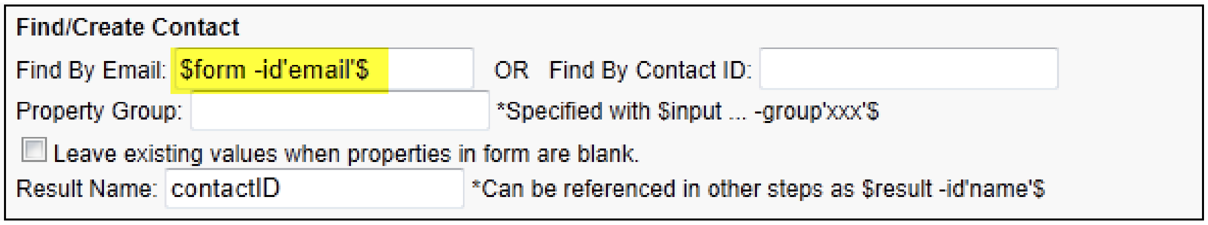
For example, supposed you have a form body that captures a user's email address with a $Input command named 'email' as shown below.



Form Body:

```
</style>
</head>

<body>
<form method="POST">
Email Address: $input -email -id'email'$<br />
First Name: $input -conta   roperty''firstname' -id'firstname'$<br />
```

You could use that email address to find the contact in iService with the Find/Create Contact action as shown below.



Find/Create Contact

Find By Email:	$form -id'email'$
OR Find By Contact ID:	
Property Group:	*Specified with $input ... -group'xxx'$
☐ Leave existing values when properties in form are blank.	
Result Name:	contactID *Can be referenced in other steps as $result -id'name'$

You can reference any $Input value from your form body within actions using the $Form command.

Using $Form -id within the form body

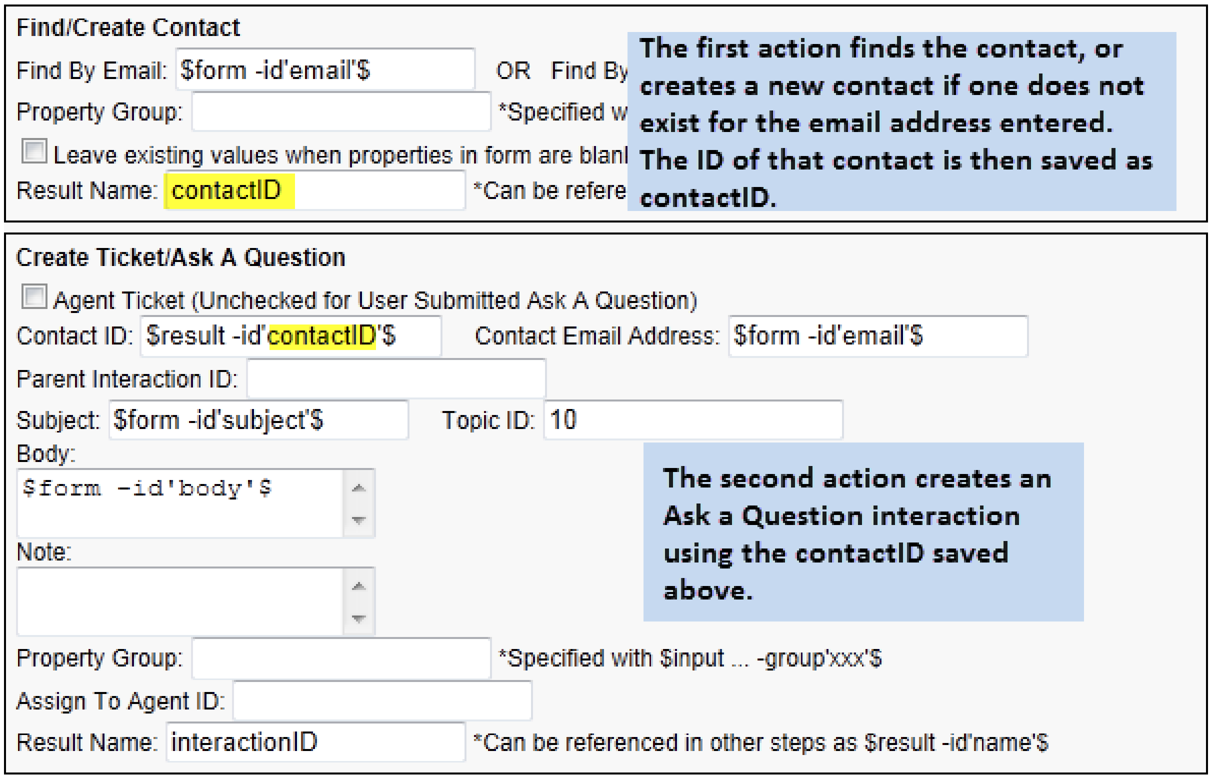
In addition to using the $Form -id command within form actions, you can use the parameter within the form body to display a value.

3.14 Action Command - $Result

The result command is used to label the result that is created within an action. This label can then be referenced in the next action that is taken. The syntax for this command is as follows.

```
$result -id'NameOfResult'$
```

For instance, when a new contact is created the resulting contactID created by iService can be labeled contactID or newContact. Then, the Create Ticket/Ask a Question action can use this to generate a new interaction for that contact. An example of the usage of the $result command is as follows.



Find/Create Contact

Find By Email: $form -id'email'$ OR Find By

Property Group: *Specified w

☐ Leave existing values when properties in form are blanl

Result Name: contactID *Can be refere

The first action finds the contact, or creates a new contact if one does not exist for the email address entered. The ID of that contact is then saved as contactID.

Create Ticket/Ask A Question

☐ Agent Ticket (Unchecked for User Submitted Ask A Question)

Contact ID: $result -id'contactID'$ Contact Email Address: $form -id'email'$

Parent Interaction ID:

Subject: $form -id'subject'$ Topic ID: 10

Body:

```
$form -id'body'$
```

Note:

Property Group: *Specified with $input ... -group'xxx'$

Assign To Agent ID:

Result Name: interactionID *Can be referenced in other steps as $result -id'name'$

The second action creates an Ask a Question interaction using the contactID saved above.

Form Actions

4 Form Actions

What makes the iService forms unique is their ability to trigger actions directly within iService when submitted. Most of the actions that you take as an agent, such as looking up contacts and creating tickets, can be triggered by the submission of an iService form. Each form may also have multiple actions, and each action can include multiple steps.

Consider a simple form used to capture a customer question. The first step in the process is to find the customer within iService, and if they don't exist create a contact for them. The second step is then to create the question either as a Ticket or an Ask a Question interaction. in this example the form would include two steps within the "Create Ticket" action. The first step would be to find the customer, and the second step would be to create the Ticket.

The forms interface uses the same action objects as iService **Filters** and **Alerts**. However, some action steps are not appropriate in the context of a form submission.
The action steps supported within iService forms include the following.

- Find/Create Contact – This action is used to capture contact information from the user’s input and update iService. It can be used independently to capture and update contact information, or in conjunction with the Create Ticket/Ask a Question action. For example, before you create a Ticket you must identify or create the contact for which you are creating the ticket.

- Create Ticket/Ask a Question – This action creates a new interaction and requires the Find/Create Contact action to be run first. Forms that create tickets require an agent login, but only agents are authorized to use the Customer Info-Contact-Create Ticket functionality.

- Create Note – This action can create either a Public or Private note for the selected contact.

- Create Agent Email – This action sends and Agent Email to the selected contact. All of the options available from the Customer Info – Contact – Agent Email tab are available (send secure, expect reply, etc.), plus the message can be added to an existing thread by use of the Parent Interaction ID parameter.
- My Account Subscriptions - This action enables My Account subscriptions to update the iService database.
- Chat Actions - Special actions are used to create and manage live chats, including the following:
  - Create - to create the new chat interaction requested by the customer
  - End - to end an active chat
  - Get Next - action when an agent takes a chat that is waiting
  - Keep Alive - to keep the chat interaction alive
  - Send - to send typed comments into the chat
  - Toggle Availability - for the agent to indicate they are available for chatting

The Form Action Panel

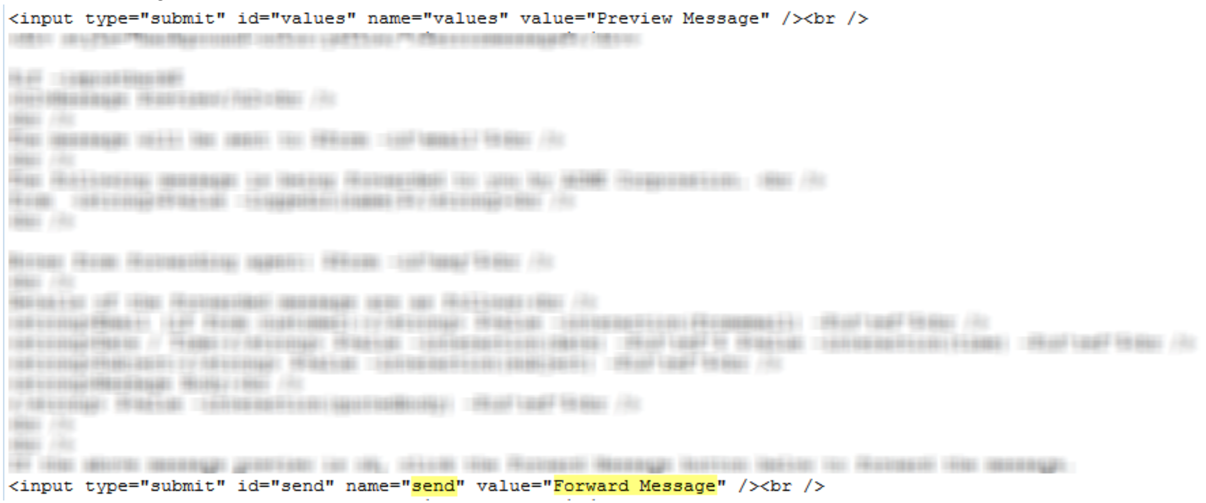
The form action panel, shown below, is used to construct the iService actions triggered from a form submission.
1 - Check form input name - Since forms can contain multiple actions, the Check Input Name option is used to determine when the action is performed. If an input is used to trigger the action, then it will only execute when the input contains the value that matches the Regex value (see 2 below). For example, a form could include a drop down menu for indicating whether to create a note or ticket when the form is submitted. That input

box might have the name of TYPE and its values could be NOTE and TICKET. In this example, the input name is TYPE and the regex value entered into the form action would be TICKET to indicate the user wants to create a ticket. If this section is left blank, the action will be triggered each time the form is submitted.

2 - For value (regex) - When you use the "Check form input name" option (see 1 above), you must specify the expression from that input that matches this action.

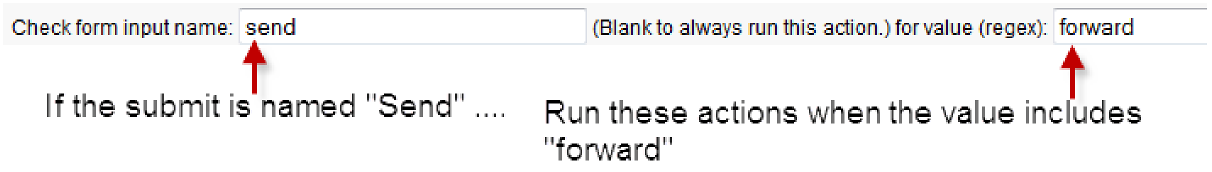
The "Check Form Input Name" and "For Value" fields are used when you have a form with multiple submit buttons, or you only want the submit to trigger actions at certain times. As shown below, the form actions are tied to the form body through the HTML name and value elements of the Input tag.

Form body with two submit actions:



```
<input type="submit" id="values" name="values" value="Preview Message" /><br />
[illegible]
<input type="submit" id="send" name="send" value="Forward Message" /><br />
```

Form action setting tied to the Forward submit:



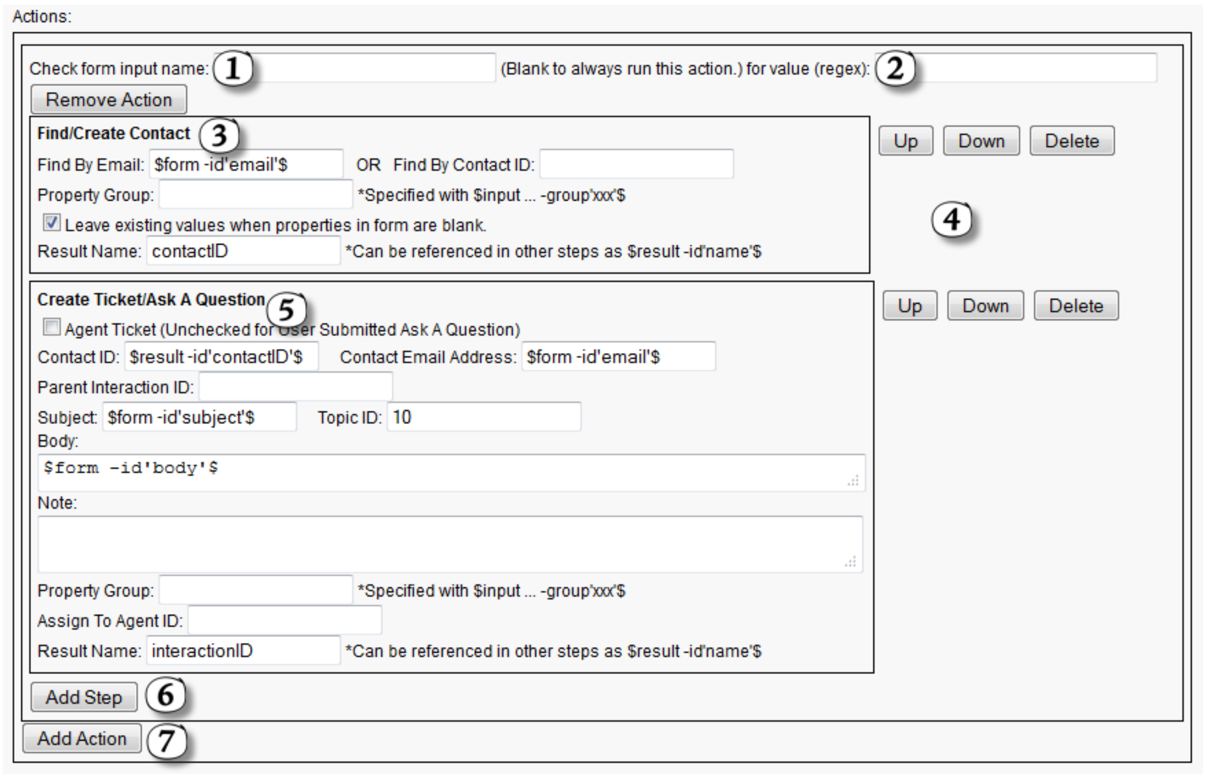
Check form input name: send (Blank to always run this action.) for value (regex): forward

If the submit is named "Send" ....
Run these actions when the value includes "forward"

3 & 5 - Action Steps - Each action is composed of various action steps, such as find a contact and create a ticket. This section contains the list of action steps and their configured parameters.

4 - Action Step Order - Certain action steps must be completed as prerequisites for other action steps. In this example, you must first find or create the contact before you create a ticket for them. The Up, Down and Delete buttons are used to adjust the order of action steps within the action.

6 - Add Step To Action - The "Add Step" button is used to add a new step into the action.
7 - Add Action - The "Add Action" button is used to add another action to the form. When you have more than one action, you will want to use the Check Form Input Name parameter to ensure the proper action is triggered.


Actions:

Check form input name: 1 (Blank to always run this action.) for value (regex): 2

3 Find/Create Contact

Find By Email: $form -id'email'$ OR Find By Contact ID:

Property Group: *Specified with $input ... -group'xxx'$

☑ Leave existing values when properties in form are blank.

Result Name: contactID *Can be referenced in other steps as $result -id'name'$

5 Create Ticket/Ask A Question

☐ Agent Ticket (Unchecked for User Submitted Ask A Question)

Contact ID: $result -id'contactID'$ Contact Email Address: $form -id'email'$

Parent Interaction ID:

Subject: $form -id'subject'$ Topic ID: 10

Body:

```
$form -id'body'$
```

Note:

Property Group: *Specified with $input ... -group'xxx'$

Assign To Agent ID:

Result Name: interactionID *Can be referenced in other steps as $result -id'name'$

Actions Not Supported by Forms

The following actions are not currently supported from the iService Forms interface. Adding these actions to a form will have no effect on the form submission and will simply be ignored. Only those actions with names preceded by _Form are available for use.

- Change Topic
- Filter
- Resolve Interaction
- Select Agents

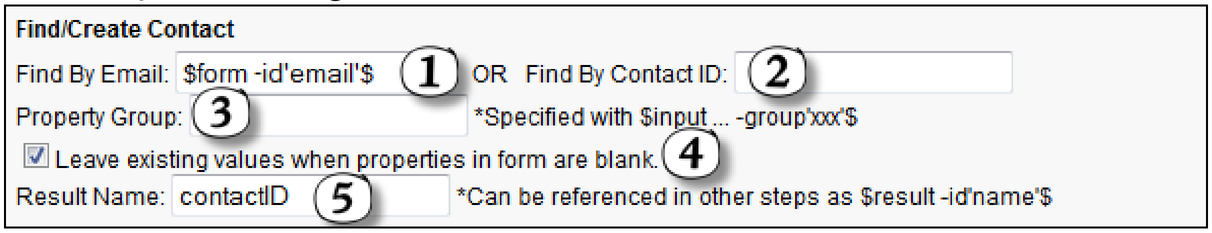
- Select Auto Responses
- Set Interaction Property
- Change Interaction Parent
- Forward External

4.1 Find / Create Contact

The Find/Create contact action step is used to locate a contact within iService, and if one does not exist to create a new contact. This action can be used alone to construct a contact import form, or as part of an action to create tickets, notes, and other actions taken for a contact. This action is similar to using the Customer Info - Search page in iService to find a contact, and and the Customer Info - Contact - Add Contact option to create a contact.

The parameters for this action step are shown below.
1 and 2 - Find Contact - The form body will specify an input for finding the contact within iService. The contact can be found either by their email address (or any of the login's specified on their contact record) or the contact ID of their record. The iService Contact ID is used in cases where you have replicated iService contacts within another operational system and have recorded the iService contact ID in that system. In most forms, you will use the email address of the contact typed into the form body.
3 - Property Group - When you have multiple contacts on a form (e.g., an HR form that specifies a supervisor and a new employee), you will need to specify with contact the various contact properties are associated with. For example, you might have two contacts in the form and for each contact you capture FirstName and LastName. You will use the -group parameter within the form body, and in this form action you specify which contact properties are used.
4 - Leave existing values - When a user submits a form they might not complete all of the fields (if not required). Check this box to leave any existing values in iService when the field is left blank. For instance, if a user did not enter their name on the form but they have an existing contact

record in iService, their existing name would be replaced with a blank unless this box is checked.
5 - Once you find or create the contact, iService will obtain the contact ID for that user. You can then use that ID in another action step, such as Create Ticket, by referencing this ID with the $Result command.


Find/Create Contact

Find By Email: $form -id'email'$ (1) OR Find By Contact ID: (2)
Property Group: (3) *Specified with $input ... -group'xxx'$
☑ Leave existing values when properties in form are blank. (4)
Result Name: contactID (5) *Can be referenced in other steps as $result -id'name'$

The Find/Create Action Step

4.2 Create Ticket / Ask a Question Interaction

The Create Ticket / Ask a Question action step is used to create a new customer inquiry in iService. It is used with the Find / Create Contact action step to determine the contact that is submitting the question. The parameters for the Create Ticket / Ask a Question action step are explained below. Those marked with an * are required elements of an interaction and must be populated.

1. Agent Ticket – If this box is checked, the new interaction will be created as an Agent Ticket instead of an Ask a Question interaction. Agent tickets do not send automated responses acknowledging the new interaction. Only iService agents with access to the Customer Info tab have the ability to create tickets. Therefore, the form body must include the Form Login and Form Password commands that allow an agent to enter their login and password to generate tickets (or the agent must be logged into iService in another browser window when submitting the form).

2. Contact ID* – The interaction must be associated with a contact, which is specified in the Find / Create Contact action step. The $Result command is used to specify the result saved by action step used to find or create the contact.

3. Contact Email Address* – Every incoming question must be associated with an email address. The email address is specified in the form body and given an ID, which in this example is named ‘email’.

4. Parent Interaction ID - iService can thread an incoming interaction into the history or a prior message thread using the ID of the parent interaction. If your form captures the ID of that existing message thread, this parameter can be used to place the new ticket or AAQ interaction into that existing thread.

5. Subject* - In this example, the subject line of the message is derived from the form input using the field with an ID of “subject".

6. Topic* - Every interaction must be associated with a Topic. The ID of the target topic is entered here. In this example it is hard coded to a specific topic ID, but you can also allow the user to select the topic from your topic tree within the form body using the topic tree parameter.

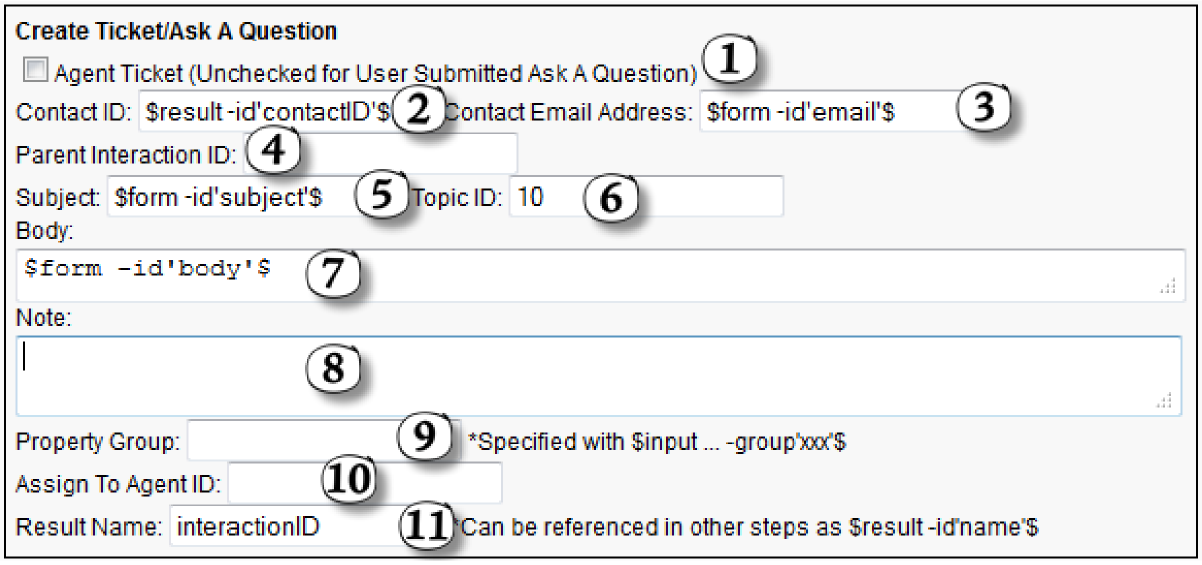
7. Body* - Every interaction must have a body. In this example, the body is derived from the form input using the field with an ID of ‘body’. However, you can place any text desired into this parameter and can include any input elements from the form body to compose custom ticket bodies.

8. Note – Agent Notes can be added to the new interaction, which will be viewable by agents from within iService. This is an optional field and is displayed within iService as Private Comments.

9. Property Group - If your form creates multiple interactions, use the -group parameter in the form body to specify which interaction properties are associated with this action step.

10. Assign to Agent ID – Tickets can optionally be assigned to a specific agent. The default is for the ticket to be unassigned, but you can enter the ID of the agent here for direct assignment.

11. Result Name – If a value is entered here, the InteractionID created by the form can be used within additional submit actions. This is optional. For more information on using results, see the $Result action command.


Create Ticket/Ask A Question
☐ Agent Ticket (Unchecked for User Submitted Ask A Question) 1
Contact ID: $result -id'contactID'$ 2 Contact Email Address: $form -id'email'$ 3
Parent Interaction ID: 4
Subject: $form -id'subject'$ 5 Topic ID: 10 6
Body:
$form -id'body'$ 7
Note:
8
Property Group: 9 *Specified with $input ... -group'xxx'$
Assign To Agent ID: 10
Result Name: interactionID 11 Can be referenced in other steps as $result -id'name'$

The Create Ticket / Ask a Question Action Step

4.3 Create Note

The Create Note action step is used to add a note to the history of the contact selected. It is used with the Find / Create Contact action step to determine the contact to which you are adding the note. The parameters for the Create Note action step are explained below. Those marked with an * are required elements of an Note and must be populated.

1. Public Note – If this box is checked, the Note will be created as Public Note that is viewable by the contact from their My Account page within iService. To make the note private so it is only viewable by agents with access to the contact, leave the box unchecked.

2. Contact ID* – The note must be associated with a contact, which is specified in the Find / Create Contact action step. The $Result command is used to specify the result saved by action step used to find or create the contact.

3. Parent Interaction ID - iService can thread notes into the history or a prior message thread using the ID of the parent interaction. If your form captures the ID of that existing message thread, this parameter can be used to place the new note into that existing thread.

4. Subject* - In this example, the subject line of the note is derived from the form input using the field with an ID of “subject".

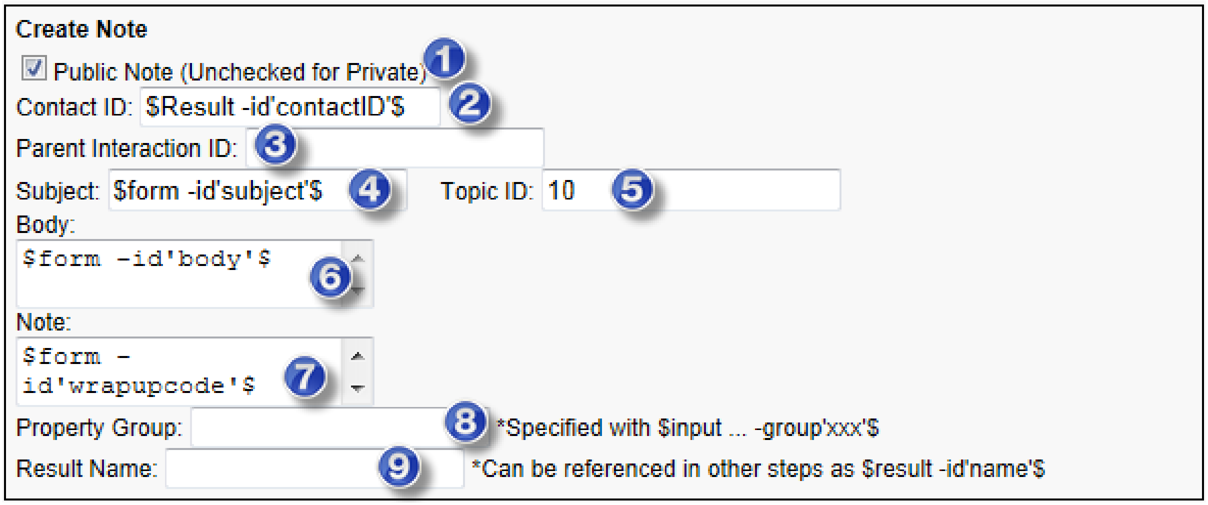
5. Topic* - Every interaction (including notes) must be associated with a Topic. The ID of the target topic is entered here. In this example it is hard coded to a specific topic ID, but you can also allow the user to select the topic from your topic tree within the form body using the topic tree parameter.

6. Body* - Every interaction must have a body. In this example, the body is derived from the form input using the field with an ID of ‘body’. However, you can place any text desired into this parameter and can include any input elements from the form body to compose custom note bodies.

7. Note – Agent Notes can be added, which will be viewable by agents from within iService. This is an optional field and is displayed within iService as Private Comments and cannot be seen by the contact from the My Account page even if the note is marked as public.

8. Property Group - If your form creates multiple interactions, use the -group parameter in the form body to specify which interaction properties are associated with this action step.

9. Result Name – If a value is entered here, the InteractionID created by the form can be used within additional submit actions. This is optional. For more information on using results, see the $Result action command.


Create Note

☑ Public Note (Unchecked for Private) 1

Contact ID: $Result -id'contactID'$ 2

Parent Interaction ID: 3

Subject: $form -id'subject'$ 4 Topic ID: 10 5

Body:
```
$form -id'body'$
```
6

Note:
```
$form -
id'wrapupcode'$
```
7

Property Group: 8 *Specified with $input ... -group'xxx'$

Result Name: 9 *Can be referenced in other steps as $result -id'name'$

The Create Note Action Step

4.4 Create Agent Email

The Create Agent Email action step is used to create a new agent email in iService that is sent to the specified contact. It is used with the Find / Create Contact action step to determine the contact that will receive the email. The parameters for the Create Agent Email action step are explained below. Similar to the create ticket action step, and agent e-mail can only be sent by an agent that is authenticated to iService. The form body used with this action step must include the login commands, or the agent must be logged in within another browser window when submitting this form. Those marked with an * are required elements of an interaction and must be populated.

1. Send As Secure Message – If this box is checked, the agent email will be sent as a secure message. This will send the contact the specified secure message auto response template instead of the actual e-mail. the contact will then be required to log into a secure website to read the message.

2. Secure Message Notification ID - This is the ID of the auto response template that will be sent to the contact if the send as secure message box is checked. This is a required field within the agent e-mail is sent as a secure message.

3. Expect Customer Response - If this box is checked, a copy of the sent message will be placed into the agents pending tab within their Msg Queue - My Queue page.

4. From Mailbox ID - Specify the ID of the mailbox from which the agent e-mail will be sent.

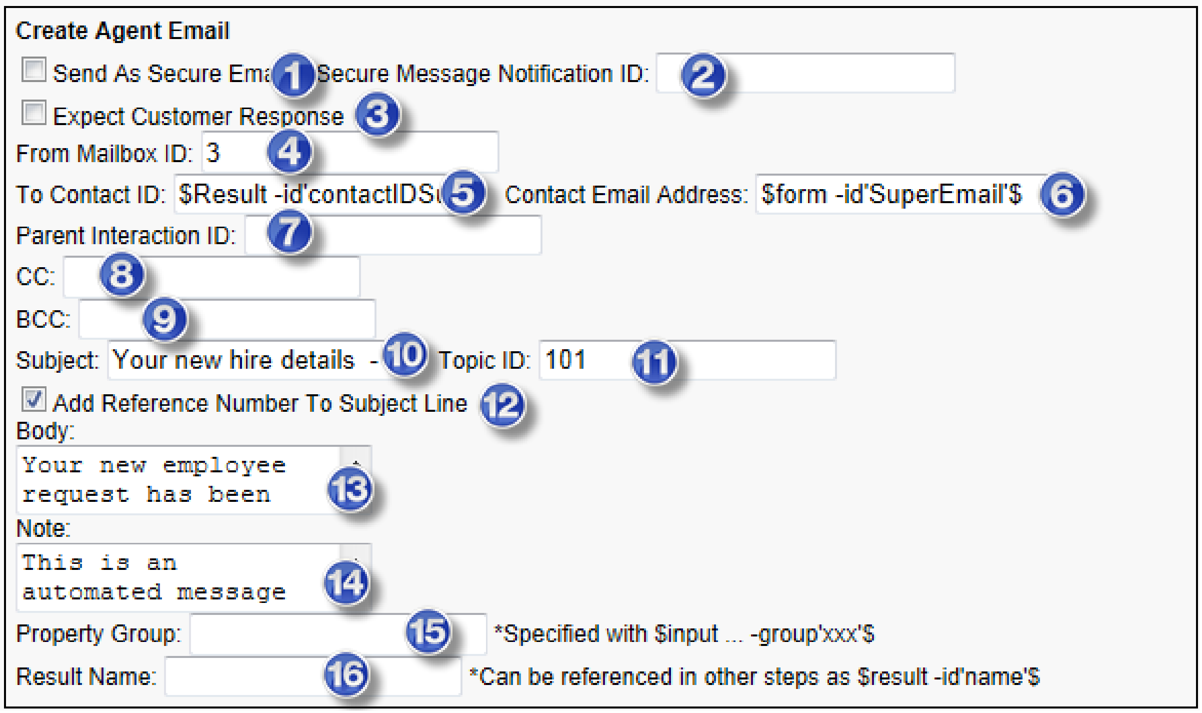
5. To Contact ID* – The agent e-mail must be associated with a contact, which is specified in the Find / Create Contact action step. The $Result command is used to specify the result saved by action step used to find or create the contact.

6. Contact Email Address* – Every incoming question must be associated with an email address. The email address is specified in the form body and given an ID, which in this example is named ‘SuperEmail’. Since a contact may have multiple e-mail addresses, e-mail address used must be specified here.

7. Parent Interaction ID - iService can thread interactions into the history or a prior message thread using the ID of the parent interaction. If your form captures the ID of that existing message thread, this parameter can be used to place the agent e-mail into that existing thread.

8. CC: - A carbon copy of the agent e-mail can be sent to other recipients by specifying their e-mail addresses here, or capturing their e-mail addresses from the form body input.

9. BCC: - A blind carbon copy of the agent e-mail can be sent to the recipients by specifying their e-mail addresses here, or capturing their e-mail addresses from the form body input.
10. Subject* - In this example, the subject line of the message is hard coded as a parameter of the action step (Your new hire details).
11. Topic* - Every interaction must be associated with a Topic. The ID of the target topic is entered here. In this example it is hard coded to a specific topic ID, but you can also allow the user to select the topic from your topic tree within the form body using the topic tree parameter.
12. Add Reference Number to Subject Line - Agent e-mail messages can have the iService reference number suppressed from the subject line. To include a reference number, check this box.
13. Body* - Every interaction must have a body. In this example, the body is derived from the form input and combined with text to create a more descriptive e-mail message.
14. Note – Agent Notes can be added to the new interaction, which will be viewable by agents from within iService. This is an optional field and is displayed within iService as Private Comments.
15. Property Group - If your form creates multiple interactions, use the -group parameter in the form body to specify which interaction properties are associated with this action step.
16. Result Name – If a value is entered here, the InteractionID created by this action step can be used within additional submit actions. This is optional. For more information on using results, see the $Result action command.


Create Agent Email

- [ ] Send As Secure Ema 1 Secure Message Notification ID: 2
- [ ] Expect Customer Response 3

From Mailbox ID: 3 4

To Contact ID: $Result -id'contactIDS 5 Contact Email Address: $form -id'SuperEmail'$ 6

Parent Interaction ID: 7

CC: 8

BCC: 9

Subject: Your new hire details - 10 Topic ID: 101 11

- [✓] Add Reference Number To Subject Line 12

Body:

```
Your new employee
request has been
```
13

Note:

```
This is an
automated message
```
14

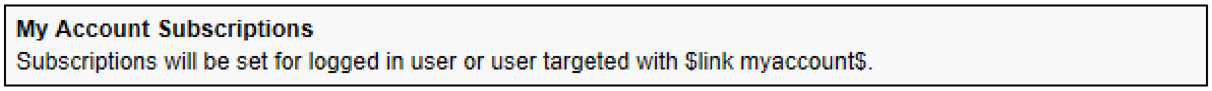
Property Group: 15 *Specified with $input ... -group'xxx'$

Result Name: 16 *Can be referenced in other steps as $result -id'name'$

The Create Agent Email Action Step

4.5 My Account Subscription

The My account subscription action is used to enable updates to iService from the my account form body commands. Without this action, the my account page will display subscription information for the contact but will not allow those values to be updated.


My Account Subscriptions

Subscriptions will be set for logged in user or user targeted with $link myaccount$.

The My Account Subscription Action Step

For an example My Account Subscription form, see the Form Examples section of this user guide.

Form Examples

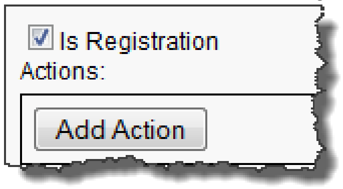
5 Form Examples

This section contains a variety of iService forms with sample code.

5.1 iService Registration (Embedded)

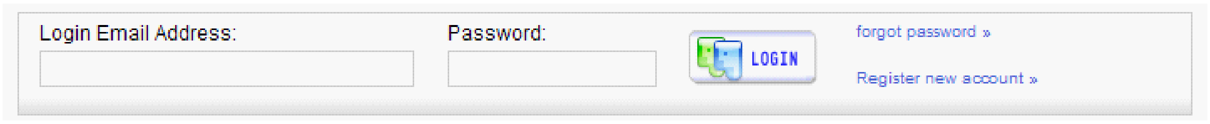
Registration forms are presented to users when they click the “Register new account” link within iService. Registered iService users have access to a My Account tab where they can view the history of their interactions with the system, and manage their subscriptions to mailing lists and knowledge base articles. Since the registration process is integrated directly into the iService website, these forms do not contain HTML begin and end tags and are unique to the way forms are used. They only render properly when loaded from within the standard iService web interface.

Registration forms are identified based upon the Is Registration checkbox below the form body.

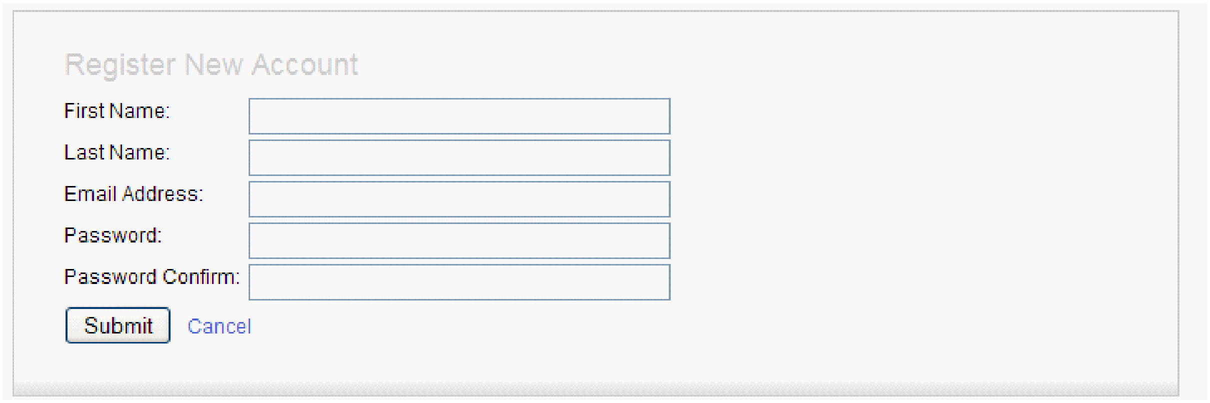


If this box is checked, the form will be available for selection in the Admin Tools > Websites page. When the Is Registration box is checked, the Preview link on the form is removed since the page is not meant to be displayed as a standard web page. If a user attempts to load a form that is labeled as registration from the form.aspx page, they will receive an “Invalid Form ID Specified” error. If the box is unchecked, iService will assume the page is meant to be displayed as a standard web page.

When an anonymous user (i.e., not logged into iService) clicks on the "Register new account" link shown below, the registration page associated with that iService website is presented.

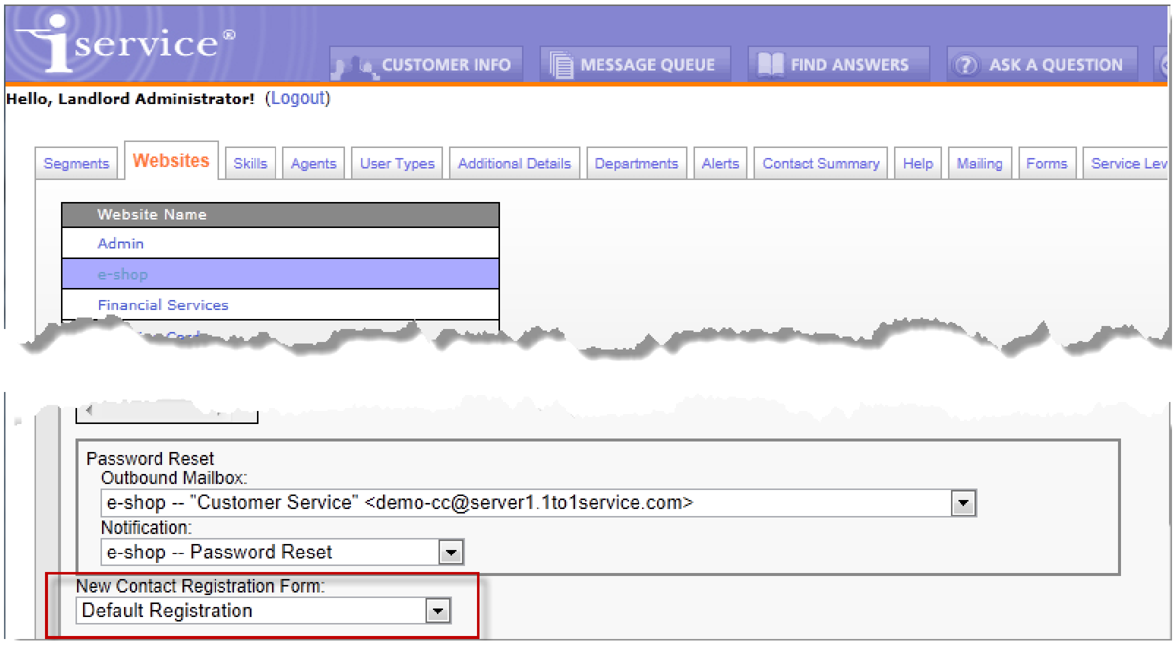


Registration forms are designed to perform two functions: create a new contact in iService, and save contact property values for the new contact. The default registration form captures the user’s email address, first Name, last name, and desired password. This can be modified within the form body to include any contact property available to customers. The standard registration page is shown below.



Selecting The Registration Form In iService

Each iService Website within a Tenant (Admin Tools > Websites) can have its own registration page. The registration form used for each website is selected from this page as shown below.



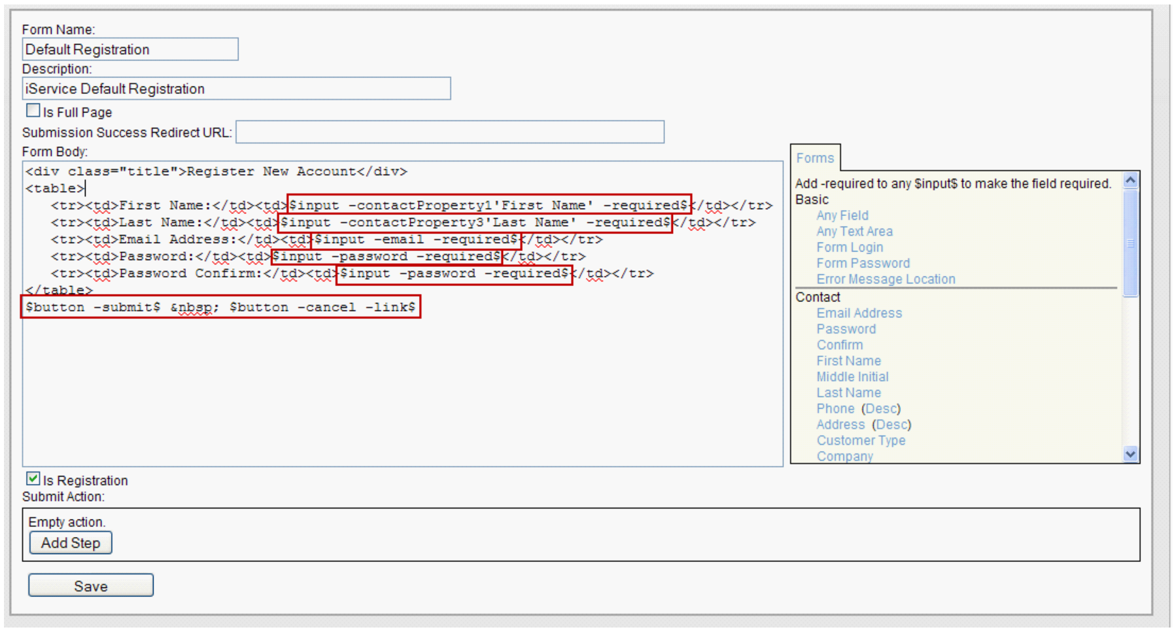
Website Name
Admin
e-shop
Financial Services

Password Reset
Outbound Mailbox: e-shop -- "Customer Service" <demo-cc@server1.1to1service.com>
Notification: e-shop -- Password Reset
New Contact Registration Form: Default Registration

Registration Form

Modifying the Registration Forms

To revise the registration page, you can either edit the default registration page included with iService, or create a new registration form altogether. The registration form does not include Submit Action steps because it is embedded within the iService system itself. The web services used to create the contact and populate its contact properties are managed automatically by iService for registration forms, so those actions do not need to be specified.


Form Name:
Default Registration
Description:
iService Default Registration
☐ Is Full Page
Submission Success Redirect URL:
Form Body:

```
<div class="title">Register New Account</div>
<table>
   <tr><td>First Name:</td><td>$input -contactProperty1'First Name' -required$</td></tr>
   <tr><td>Last Name:</td><td>$input -contactProperty3'Last Name' -required$</td></tr>
   <tr><td>Email Address:</td><td>$input -email -required$</td></tr>
   <tr><td>Password:</td><td>$input -password -required$</td></tr>
   <tr><td>Password Confirm:</td><td>$input -password -required$</td></tr>
</table>
$button -submit$ &nbsp; $button -cancel -link$
```

Forms
Add -required to any $input$ to make the field required.
Basic
Any Field
Any Text Area
Form Login
Form Password
Error Message Location
Contact
Email Address
Password
Confirm
First Name
Middle Initial
Last Name
Phone (Desc)
Address (Desc)
Customer Type
Company

☑ Is Registration
Submit Action:
Empty action.
Add Step
Save

Modifying a Registration Form

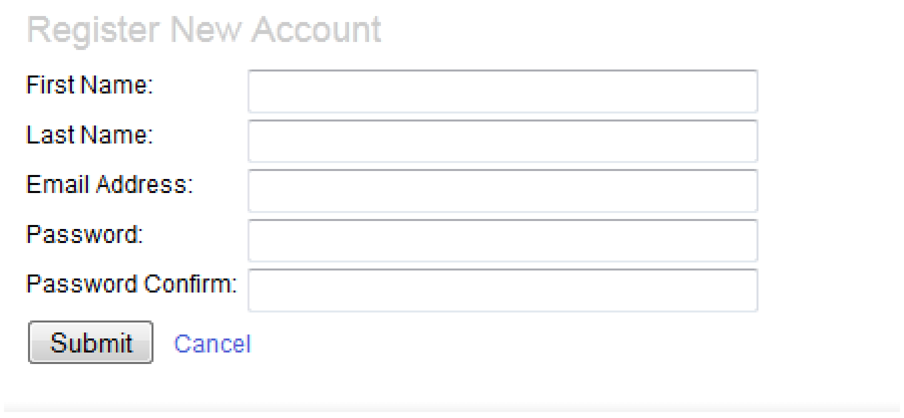
The form commands within the standard registration page are outlined above in red. All properties are required, as indicated by the '–required' parameter within each command. The Submit and Cancel buttons are outlined in red at the bottom of the form. Including the –link parameter within the button command causes the form to display that button as a link, rather than a standard button.

The $button Command

Registration forms use a special iService command named "button". This command generates the "submit" and "cancel" buttons that are unique to the registration process. The –link parameter is optional and converts the button to a hyperlink. The form of this command is as follows:

```
$button -submit$     $button -cancel -link$
```

In the example above, the Submit function is rendered as a button and the Cancel function is rendered as a link.


Register New Account

First Name:

Last Name:

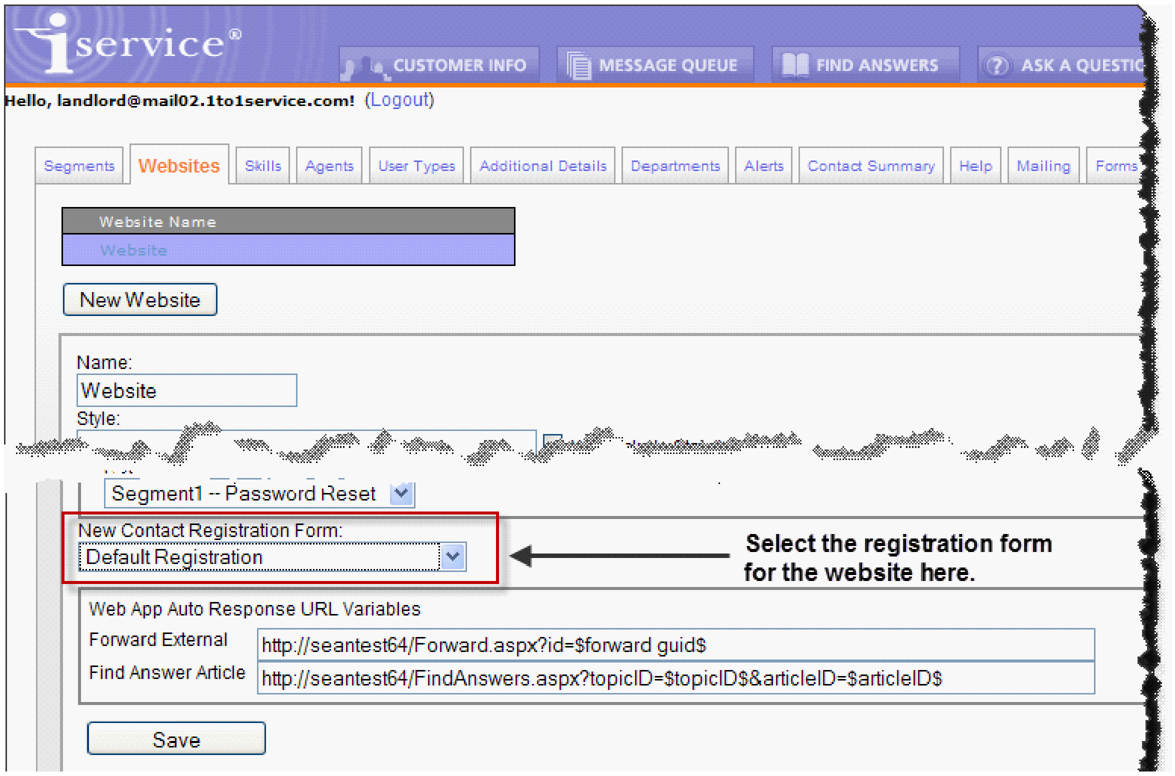
Email Address:

Password:

Password Confirm:

Testing New Registration Forms

To test registration forms, you can create a test iService website from Admin Tools>Websites, and select the new registration form.



Hello, landlord@mail02.1to1service.com! (Logout)

Website Name
Website

Name: Website

Style:

Segment1 -- Password Reset

New Contact Registration Form: Default Registration

Select the registration form for the website here.

Web App Auto Response URL Variables

Forward External	http://seantest64/Forward.aspx?id=$forward guid$
Find Answer Article	http://seantest64/FindAnswers.aspx?topicID=$topicID$&articleID=$articleID$

Selecting the Registration Form in the Websites Tab

For additional information regarding creating and managing iService websites, see the iService Setup Guide.

5.2 Contact Update/Import Forms

The iService forms interface includes a wizard for creating contact update/import forms. These forms can be used to manually add or update contact information, or with the iService batch forms update utility to import contacts from a.CSV file.

The Contact Import Form Wizard allows you to select any contact property defined for your tenant, and automatically creates the form body and action required for the form. The form body generated will include an input command for each contact property, and a single action (Find/Create Contact) for generating or updating a contact based upon the information entered into the form.

If the form is going to be used manually, you should add a “Submission Success Redirect URL” that is displayed when the form submits successfully.

Example Form Body for Contact Update / Import

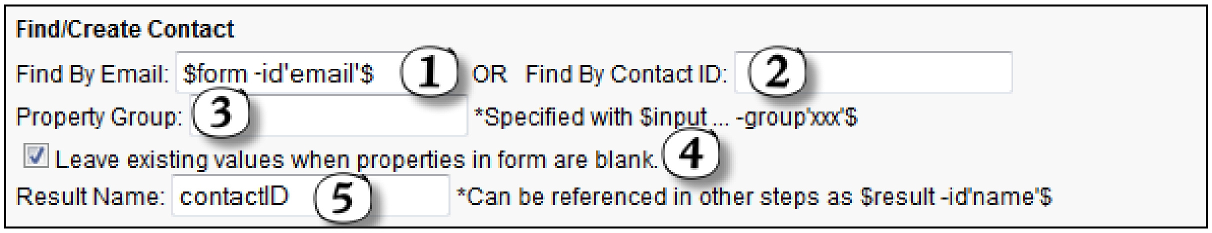
An example of a completed form body for a contact update/import form is shown below. The form is essentially a list of contact properties.

```
<html>
<body>
<form method="POST">

Email Address: $input -email -id'email' -required$<br />
First Name: $input -contactProperty1'firstname' -id'firstnam
Last Name: $input -contactProperty3'lastname' -id'lastname'$
Company: $input -contactProperty7'company' -id'company'$<br
Address1: $input -contactProperty15'address1' -id'address1'$
Address2: $input -contactProperty16'address2' -id'address2'$
City: $input -contactProperty17'city' -id'city'$<br />
State: $input -contactProperty18'state' -id'state'$<br />
Postal Code: $input -contactProperty19'postalcode' -id'posta
Customer Type: $input -contactProperty6'Customer Type' -id'c

<input type="submit" id="ok" name="ok" value="Create/Update
<div style="background-color:yellow;">$errormessage$</div>
</form>
```

```
</body>
</html>
```

Example Form Action for Contact Update / Import

An example of the “Submit Action” used with the form body shown about is as follows.


Find/Create Contact

Find By Email: $form -id'email'$ OR Find By Contact ID:

Property Group: *Specified with $input ... -group'xxx'$

☑ Leave existing values when properties in form are blank.

Result Name: contactID *Can be referenced in other steps as $result -id'name'$

The Find/Create Contact Action Step

Create/Find Contact Elements:

1. Find By Email – To lookup the contact within iService using email address, populate the ‘Find By Email’ input box. In this example, email is the ID given to the field within the form that captures email address.
2. Find By Contact ID – To lookup the contact based upon the ContactID from the iService database, this input box would be populated.
3. Property Group - Your form body can include input for multiple contacts in a single form. This input can be used to create or find multiple contacts within iService. The property group is used to specify which contact the properties along to. For instance, if you have two first names in your form, you can specify which contact each name is associated with.
4. “Leave existing values when properties in form are blank” is recommended when the form has optional contact properties. If this box is not checked, blank entries on the form will overwrite existing values that the contact might have within iService for that property. For instance, if an existing contact completes the form but leaves Company blank, the form submission would remove any value the contact already has for company.
5. Result Name - The ‘Result Name’ field is used to save the created contact ID as a result that can be reference in other actions. In this example, the term ContactID is used. However, other text could be used and reference in actions below.

5.2.1 Using a Contact Import Form to Add Contacts to a Mailing List

If you are importing a list of contacts and want to add them to one of your mailing lists, you can use the -addtolist or “addtocampaign” parameter within your form body. This parameter will convert the input command into a check box for subscribing the contact to the list or campaign specified. If the user was previously on the list and opted out, their preference will not be overridden. So, it is safe to add contacts this way even if they were previously on the list.

The example below shows how to modify an existing contact import form to allow subscription. In the example below, each contact is added to the mailing list whose ID is 1. The name of the list that is shown below (CustomerList) can be anything and is used to make the form easier to understand. We suggest using the name of the list as specified in the Mailing List page. The full set of lists and campaigns available are shown in the Form Variable Picker on the forms page.

When you import your file using the iService Batch Submit Utility, include a column within your CSV input file with the ID that corresponds to the list. In the example below the column name of your CSV input file would be "Form-List". Then, includes the value “true” for each contact that needs to be added to the list or campaign, and the value "False" for each contact you want excluded from the list.

The full form body for this example is shown below, with the changes highlighted. There are no changes required to the form action.

```
<html>
<body>
<form method="POST">

Email Address: $input -email -id'email' -required$<br />
First Name: $input -contactProperty1'firstname' -id'firstname'$<br /
Last Name: $input -contactProperty3'lastname' -id'lastname'$<br />
Company: $input -contactProperty7'company' -id'company'$<br />
Address1: $input -contactProperty15'address1' -id'address1'$<br />
Address2: $input -contactProperty16'address2' -id'address2'$<br />
City: $input -contactProperty17'city' -id'city'$<br />
State: $input -contactProperty18'state' -id'state'$<br />
Postal Code: $input -contactProperty19'postalcode' -id'postalcode'$<
Customer Type: $input -contactProperty6'Customer Type' -id'customert
Click to Join: $input -id'List' -addtolist1'CustomerList'$<br />

<input type="submit" id="ok" name="ok" value="Create/Update Contact"
```

```
<div style="background-color:yellow;">$errormessage$</div>
</form>
</body>
</html>
```

5.3 My Account – Subscriptions Form

NOTE: Although this section uses -myaccountlist in its example, the syntax is identical for -myaccountarticle and -myaccountcampaign.

If you've integrated iService mailing lists into your website, you might want to provide a customized e-mail preferences page where your customers can manage their mailing list subscriptions. Subscription management pages use the following commands in the form body to generate a custom subscription management page.

`$if -myaccountlists$` - This command checks to see if any lists are available for display for the customer.

If there are lists available, the command $repeat -myaccountlists$ is used to generate details for all of the lists and their subscription status. If there are no lists, it will move to the $else $ command.
`$repeat -myaccountlists$` - The repeat command tells iService to create a row for each list available and to display the fields specified between “repeat” and “endrepeat”. The fields that can be displayed are:

`$value -myaccountlist(name)$` - displays the list name.
`$value -myaccountlist(description)$` - displays the list description.
`$input -id'list' -myaccount(list)$` - displays a check box for the list that can be used to subscribe or unsubscribe.

`$endrepeat$` - indicates the end of the fields to be displayed for each list.

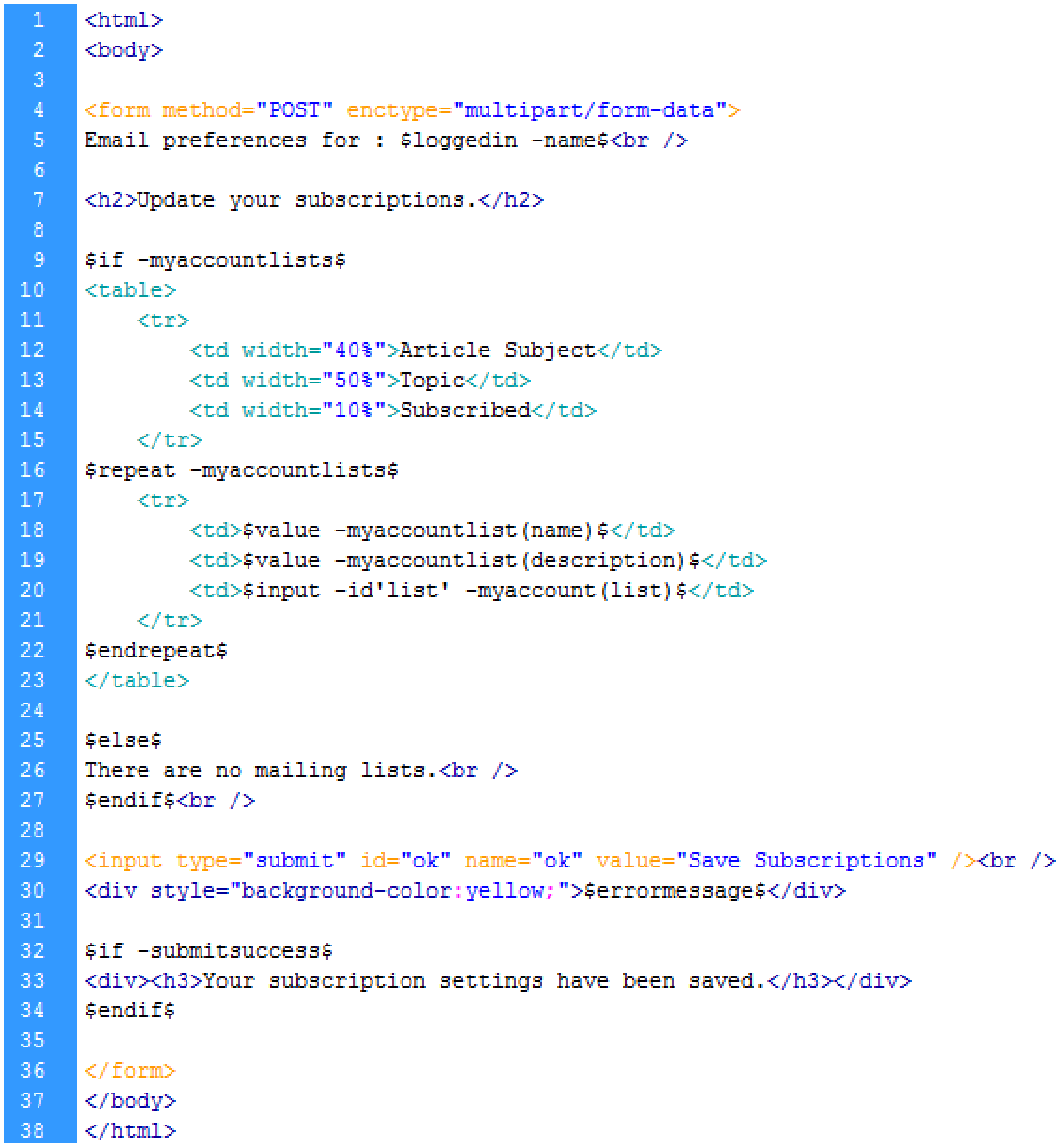
`$else$` - The else command is used when there are no lists available. If there are no lists available, the form will display the text that follows after $else$
`$endif$` - The endif command marks the and of the account list table.

The process is identical for campaigns and find answers subscriptions, using myaccountcampaigns and myaccountarticle instead of myaccountlist.

You can use the $link myaccount$ command inside of a mass mailing to direct recipients of a mass mailing message directly to this custom subscriptions page, and they will not be required to login to view their settings.

Example Form Body for Mailing List Subscription Page

An example of a simple form body for a custom My Subscriptions form is shown below.

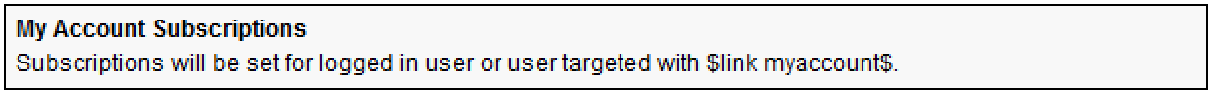


```
<html>
<body>

<form method="POST" enctype="multipart/form-data">
Email preferences for : $loggedin -name$<br />

<h2>Update your subscriptions.</h2>

$if -myaccountlists$
<table>
    <tr>
        <td width="40%">Article Subject</td>
        <td width="50%">Topic</td>
        <td width="10%">Subscribed</td>
    </tr>
$repeat -myaccountlists$
    <tr>
        <td>$value -myaccountlist(name)$</td>
        <td>$value -myaccountlist(description)$</td>
        <td>$input -id'list' -myaccount(list)$</td>
    </tr>
$endrepeat$
</table>

$else$
There are no mailing lists.<br />
$endif$<br />

<input type="submit" id="ok" name="ok" value="Save Subscriptions" /><br />
<div style="background-color:yellow;">$errormessage$</div>

$if -submitsuccess$
<div><h3>Your subscription settings have been saved.</h3></div>
$endif$

</form>
</body>
</html>
```

This form would include a single action, shown below, which enables the saving of changes made to the form.

Example Action for Mailing List Subscriptions Page

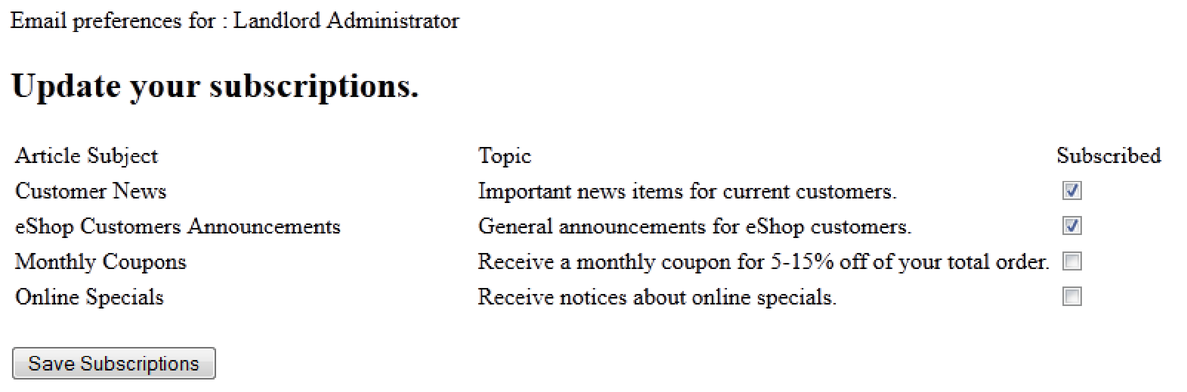
The only action required on a subscription page is _Form My Account Subscriptions. There are no parameters or action commands used within this action step.



My Account Subscriptions
Subscriptions will be set for logged in user or user targeted with $link myaccount$.

Example Display of the Subscription Form Sample

This form example would display to the user as shown below.



Email preferences for : Landlord Administrator

Update your subscriptions.

Article Subject	Topic	Subscribed
Customer News	Important news items for current customers.	✓
eShop Customers Announcements	General announcements for eShop customers.	✓
Monthly Coupons	Receive a monthly coupon for 5-15% off of your total order.	☐
Online Specials	Receive notices about online specials.	☐

5.4 Ask a Question / Ticket

To create a ticket or ask a question interaction, you will capture the necessary input in the form body and then use them in the following action steps:

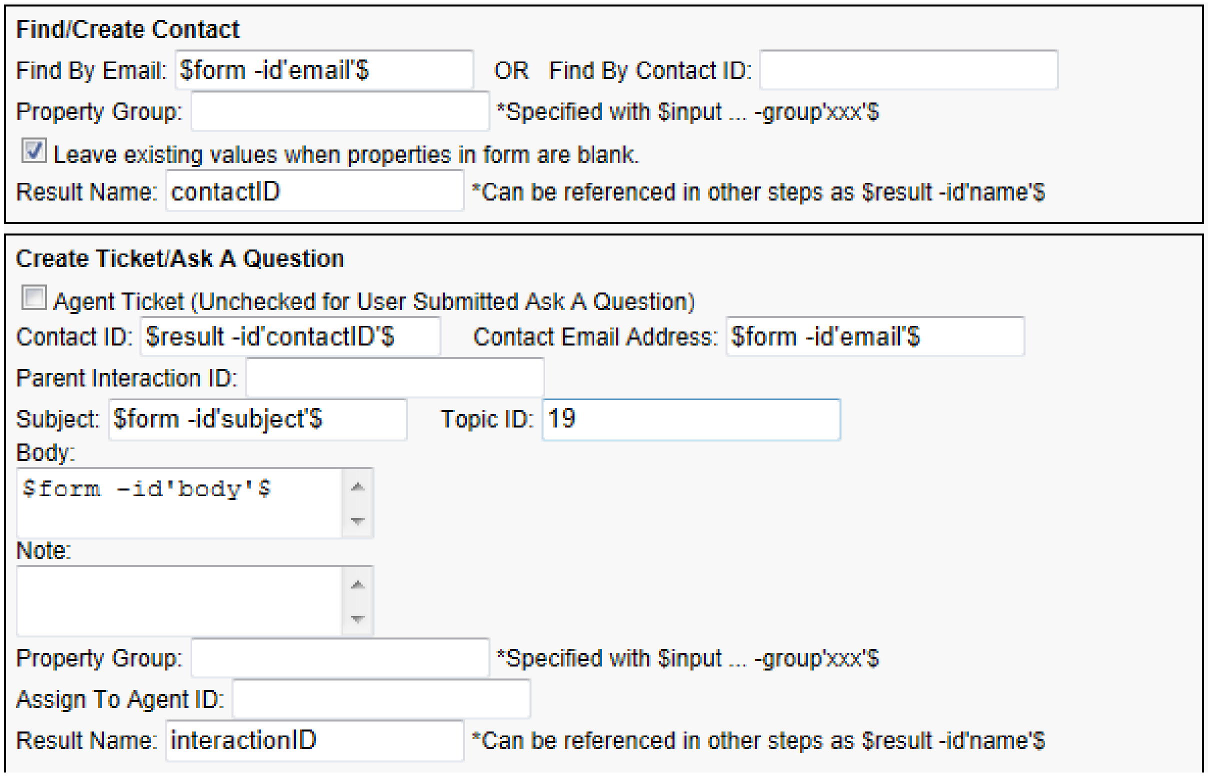
- **Find/Create Contact -** this will lookup the contact, usually based on their email address. If the contact doesn't exist iService will create one.
- **Create Ticket/Ask a Question -** this will create the ticket of AAQ interaction based on the details specified in the action.

Form Body

```
<html>
<body>
<form method="POST">

Email Address: $input -email -id'email' -required$<br />
First Name: $input -contactProperty1'firstname' -id'firstname'$<br /
Last Name: $input -contactProperty3'lastname' -id'lastname'$<br />
Subject: $input -id'subject' -required$<br />
Body: $input -textarea -id'body' -required$<br />

<input type="submit" id="ok" name="ok" value="Submit Question" /><br
<div style="background-color:yellow;">$errormessage$</div>
</form>
</body>
</html>
```

Action Steps

The find/create contact step must always be executed before the ticket can be created.



Find/Create Contact

Find By Email: $form -id'email'$ OR Find By Contact ID:

Property Group: *Specified with $input ... -group'xxx'$

☑ Leave existing values when properties in form are blank.

Result Name: contactID *Can be referenced in other steps as $result -id'name'$

Create Ticket/Ask A Question

☐ Agent Ticket (Unchecked for User Submitted Ask A Question)

Contact ID: $result -id'contactID'$ Contact Email Address: $form -id'email'$

Parent Interaction ID:

Subject: $form -id'subject'$ Topic ID: 19

Body:

```
$form -id'body'$
```

Note:

Property Group: *Specified with $input ... -group'xxx'$

Assign To Agent ID:

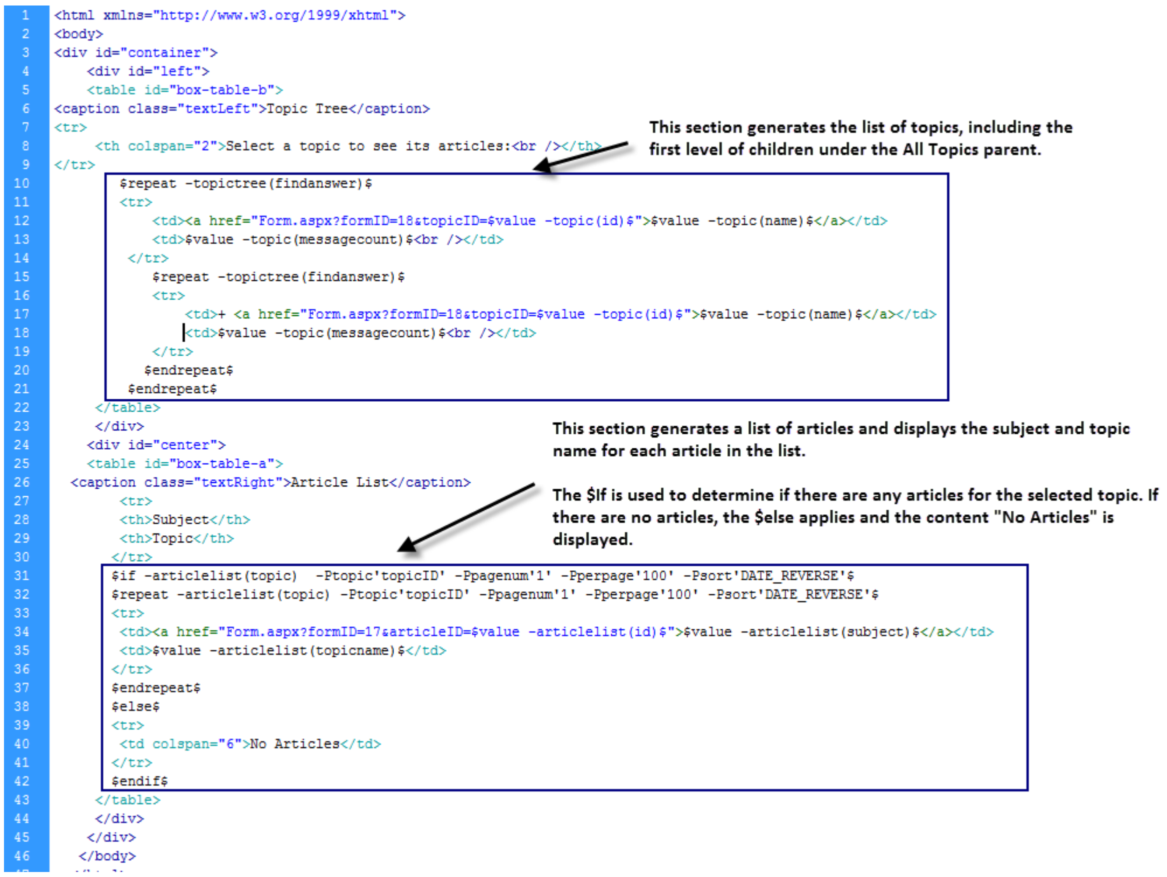
Result Name: interactionID *Can be referenced in other steps as $result -id'name'$

Actions Steps Needed To Create an AAQ Interaction

5.5 Find Answers Form

A Find Answers page typically has two distinct views: a list of articles, and the details for an article when one is selected.

Landing Page That Displays The List of Articles



```
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
<div id="container">
    <div id="left">
    <table id="box-table-b">
<caption class="textLeft">Topic Tree</caption>
<tr>
     <th colspan="2">Select a topic to see its articles:<br /></th>
</tr>
        $repeat -topictree(findanswer)$
        <tr>
            <td><a href="Form.aspx?formID=18&topicID=$value -topic(id)$">$value -topic(name)$</a></td>
            <td>$value -topic(messagecount)$<br /></td>
         </tr>
            $repeat -topictree(findanswer)$
            <tr>
                <td>+ <a href="Form.aspx?formID=18&topicID=$value -topic(id)$">$value -topic(name)$</a></td>
                <td>$value -topic(messagecount)$<br /></td>
            </tr>
          $endrepeat$
        $endrepeat$
     </table>
     </div>
    <div id="center">
    <table id="box-table-a">
  <caption class="textRight">Article List</caption>
        <tr>
        <th>Subject</th>
        <th>Topic</th>
       </tr>
       $if -articlelist(topic)  -Ptopic'topicID' -Ppagenum'1' -Pperpage'100' -Psort'DATE_REVERSE'$
       $repeat -articlelist(topic) -Ptopic'topicID' -Ppagenum'1' -Pperpage'100' -Psort'DATE_REVERSE'$
       <tr>
        <td><a href="Form.aspx?formID=17&articleID=$value -articlelist(id)$">$value -articlelist(subject)$</a></td>
        <td>$value -articlelist(topicname)$</td>
       </tr>
       $endrepeat$
       $else$
       <tr>
        <td colspan="6">No Articles</td>
       </tr>
       $endif$
     </table>
     </div>
    </div>
   </body>
```

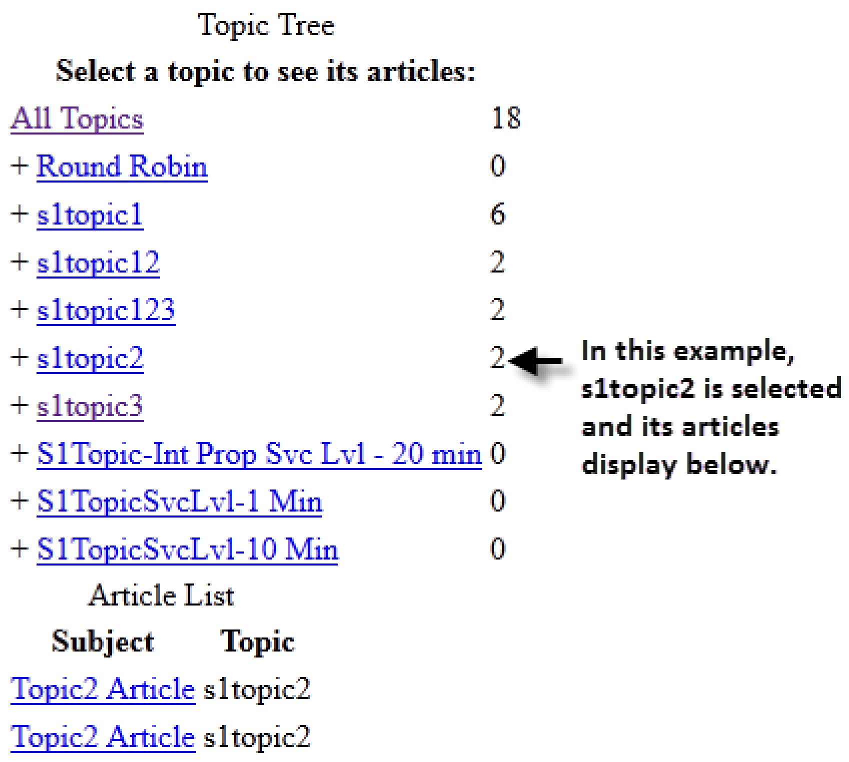
This section generates the list of topics, including the first level of children under the All Topics parent.

This section generates a list of articles and displays the subject and topic name for each article in the list.

The $If is used to determine if there are any articles for the selected topic. If there are no articles, the $else applies and the content "No Articles" is displayed.

A Simple Find Answers Article List

This example has no HTML styling, and simply places the list of articles below the list of topics. It would generate a display that looks like the following.



Topic Tree

Select a topic to see its articles:

All Topics	18
+ Round Robin	0
+ s1topic1	6
+ s1topic12	2
+ s1topic123	2
+ s1topic2	2
+ s1topic3	2
+ S1Topic-Int Prop Svc Lvl - 20 min	0
+ S1TopicSvcLvl-1 Min	0
+ S1TopicSvcLvl-10 Min	0

In this example, s1topic2 is selected and its articles display below.

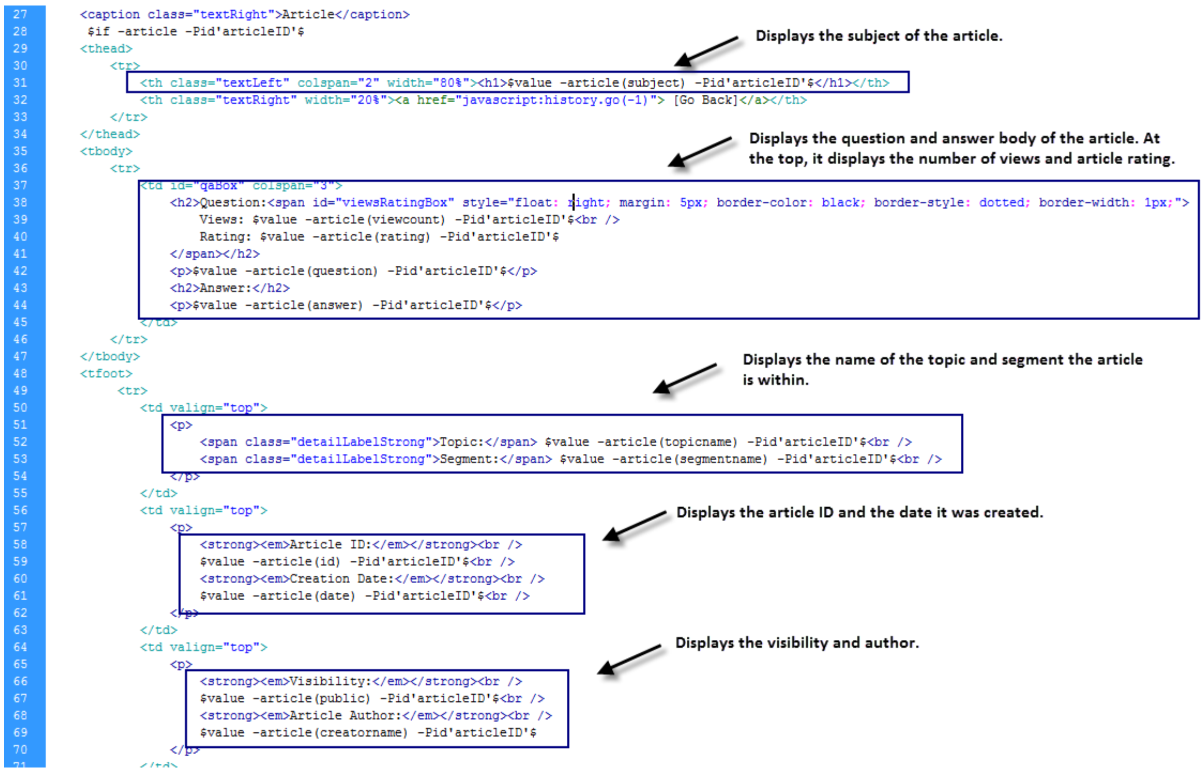
Article List

Subject	Topic
Topic2 Article	s1topic2
Topic2 Article	s1topic2

Display With Topic Selected

Article Display

In the example above, the article is displayed within a separate form for simplicity. However, in practice additional coding would be used to combine the two forms. The relevant aspects of the form used to display the article are shown below.

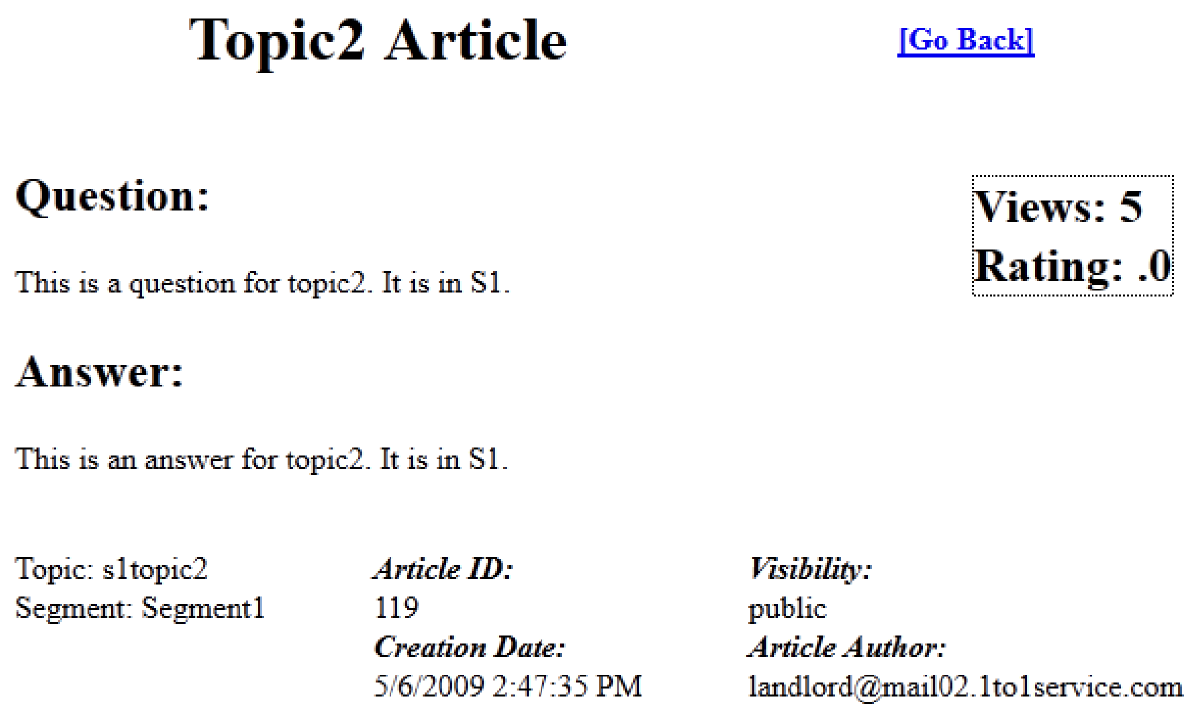


```
<caption class="textRight">Article</caption>
 $if -article -Pid'articleID'$
<thead>
    <tr>
        <th class="textLeft" colspan="2" width="80%"><h1>$value -article(subject) -Pid'articleID'$</h1></th>
        <th class="textRight" width="20%"><a href="javascript:history.go(-1)"> [Go Back]</a></th>
    </tr>
</thead>
<tbody>
    <tr>
        <td id="qaBox" colspan="3">
            <h2>Question:<span id="viewsRatingBox" style="float: right; margin: 5px; border-color: black; border-style: dotted; border-width: 1px;">
                Views: $value -article(viewcount) -Pid'articleID'$<br />
                Rating: $value -article(rating) -Pid'articleID'$
            </span></h2>
            <p>$value -article(question) -Pid'articleID'$</p>
            <h2>Answer:</h2>
            <p>$value -article(answer) -Pid'articleID'$</p>
        </td>
    </tr>
</tbody>
<tfoot>
     <tr>
        <td valign="top">
            <p>
                <span class="detailLabelStrong">Topic:</span> $value -article(topicname) -Pid'articleID'$<br />
                <span class="detailLabelStrong">Segment:</span> $value -article(segmentname) -Pid'articleID'$<br />
            </p>
        </td>
        <td valign="top">
            <p>
                <strong><em>Article ID:</em></strong><br />
                $value -article(id) -Pid'articleID'$<br />
                <strong><em>Creation Date:</em></strong><br />
                $value -article(date) -Pid'articleID'$<br />
            </p>
        </td>
        <td valign="top">
            <p>
                <strong><em>Visibility:</em></strong><br />
                $value -article(public) -Pid'articleID'$<br />
                <strong><em>Article Author:</em></strong><br />
                $value -article(creatorname) -Pid'articleID'$
            </p>
```

Displays the subject of the article.
Displays the question and answer body of the article. At the top, it displays the number of views and article rating.
Displays the name of the topic and segment the article is within.
Displays the article ID and the date it was created.
Displays the visibility and author.

Example HTML For Simple Article Display

The example above would produce an article that looks like the following.



Topic2 Article [Go Back]

Question:

Views: 5
Rating: .0

This is a question for topic2. It is in S1.

Answer:

This is an answer for topic2. It is in S1.

Topic: s1topic2	***Article ID:***	***Visibility:***
Segment: Segment1	119	public
	Creation Date:	***Article Author:***
	5/6/2009 2:47:35 PM	landlord@mail02.1to1service.com

Example Article Display

5.6 Feedback Form

iService forms allow you to quickly generate web forms to capture customer input, including the ability to save input from the form into separate interaction properties within the iService database.
This enhancement takes advantage of that capability by presenting a standard feedback form to your customers and capturing their responses within predefined interaction properties. These properties are then used to generate management reports that provide insight about your overall support process, and the performance of individual agents.

The iService Feedback process is integrated into the email response process. A link is included in the agent's reply directing the customer to an iService form that captures the feedback. This link includes the Ref # of the question that was answered, which allows iService to associate the feedback with the agent. The iService form captures the details of the feedback and stores it in the iService database where it can be analyzed with predefined reports.

Configuring iService Feedback In Your Tenant

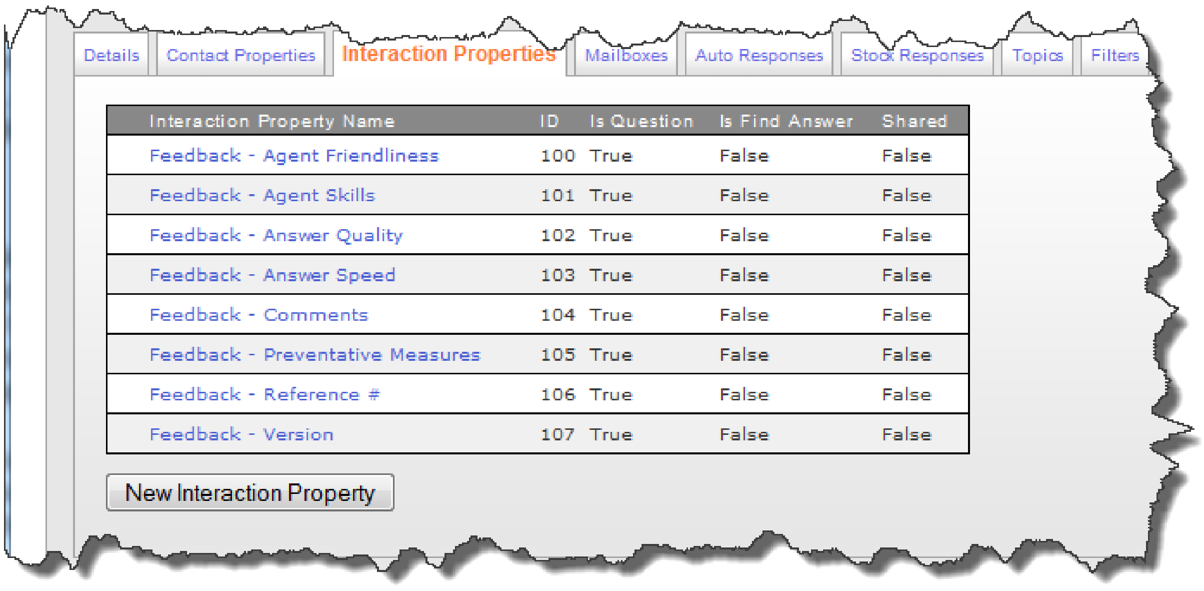
The overall setup of feedback within your iService tenant can be accomplished in less than an hour. The steps include creating new interaction properties to capture the feedback, creating the feedback form with your branding, and developing a stock response that can be used to ask for customer feedback. Each of these is explained in more detail below.

To see the feedback process in action, view the iService Feedback Flash demonstration.

Step 1 - Create the Feedback Properties

The first consideration when implementing iService feedback is where you will place your feedback properties. These interaction properties will store the answers to questions such as was the agent knowledgeable, was the response timely, etc. Although customer service representatives do not have the access rights to edit these fields, it is possible for managers or administrators to change their values. An audit trail is created for these changes, but it is usually best to create a new segment named "Feedback" and place the interaction properties into that segment.

We've developed a set of iService forms and reports based upon the following interaction properties. These correspond to the questions presented on the standard feedback form and are used to capture responses in the iService database. You can modify the form to capture answers to any question you'd like, however you will need to modify the standard reports to reflect your additional questions.



Interaction Property Name	ID	Is Question	Is Find Answer	Shared
Feedback - Agent Friendliness	100	True	False	False
Feedback - Agent Skills	101	True	False	False
Feedback - Answer Quality	102	True	False	False
Feedback - Answer Speed	103	True	False	False
Feedback - Comments	104	True	False	False
Feedback - Preventative Measures	105	True	False	False
Feedback - Reference #	106	True	False	False
Feedback - Version	107	True	False	False

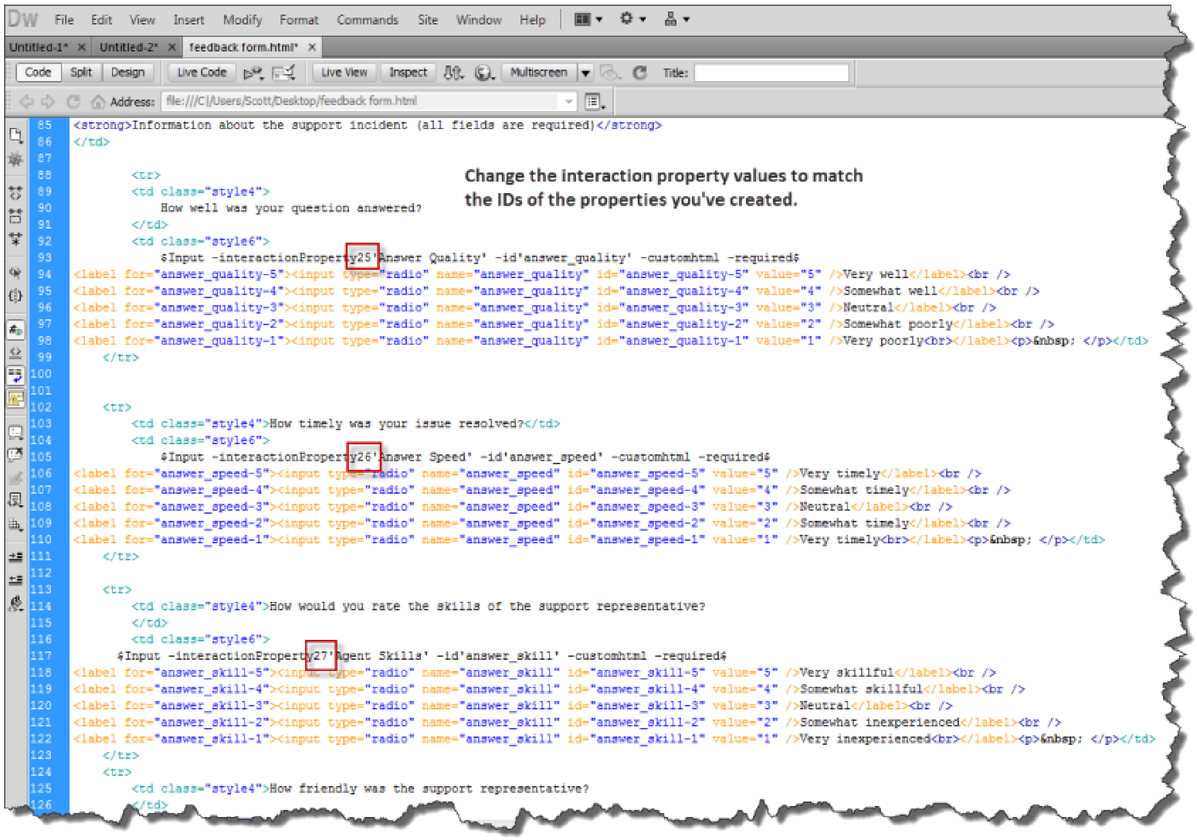
Interaction Properties Used with the Feedback Form

- Agent Friendliness (Rate 1-5)
- Agent Skills (Rate 1-5)
- Answer Quality(Rate 1-5)
- Answer Speed (Rate 1-5)
- Preventative Measures (text box)
- Reference Number
- Comments (text box)

A script is available on the One-to-One Service.com website for automatically creating these properties within your iService tenant. The script uses the Selenium IDE plug-in for Firefox to web page automation.

⊟ Step 2 - Create the Feedback Form Body

After you create your interaction properties, you can modify the sample feedback form by simply changing the property IDs within the form. You can also add your own custom branding or other HTML changes as desired. Simply open the provided HTML file (FeebackForm.html) with an HTML editor of your choice, make the changes, and then paste the form body into a new for created within the iService Forms interface (Admin Tools > Forms).


DW File Edit View Insert Modify Format Commands Site Window Help
Untitled-1* Untitled-2* feedback form.html*
Code Split Design Live Code Live View Inspect Multiscreen Title:
Address: file:///C|/Users/Scott/Desktop/feedback form.html

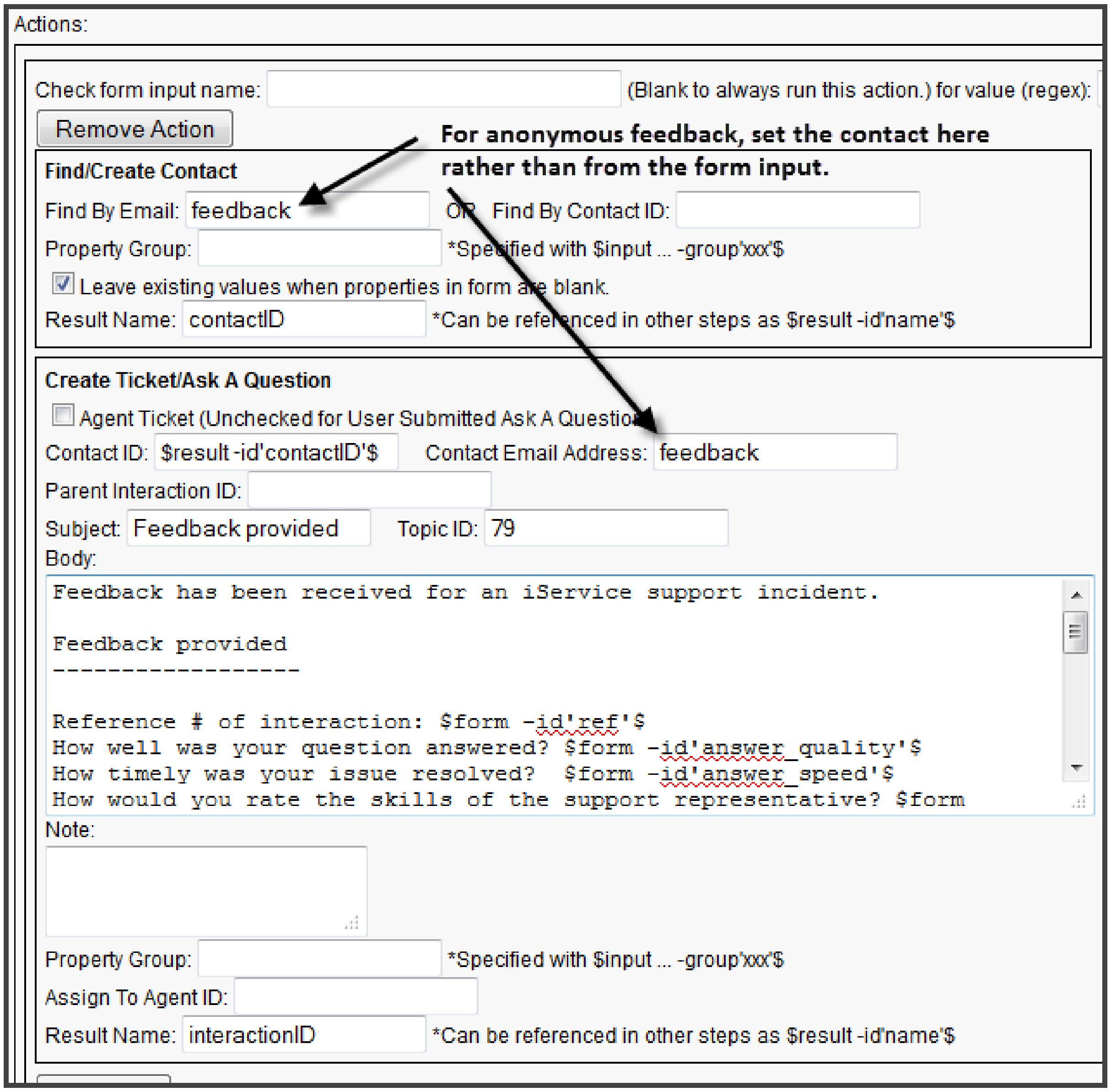
```
<strong>Information about the support incident (all fields are required)</strong>
</td>

        <tr>
        <td class="style4">
            How well was your question answered?
        </td>
        <td class="style6">
            $Input -interactionProperty25'Answer Quality' -id'answer_quality' -customhtml -required$
<label for="answer_quality-5"><input type="radio" name="answer_quality" id="answer_quality-5" value="5" />Very well</label><br />
<label for="answer_quality-4"><input type="radio" name="answer_quality" id="answer_quality-4" value="4" />Somewhat well</label><br />
<label for="answer_quality-3"><input type="radio" name="answer_quality" id="answer_quality-3" value="3" />Neutral</label><br />
<label for="answer_quality-2"><input type="radio" name="answer_quality" id="answer_quality-2" value="2" />Somewhat poorly</label><br />
<label for="answer_quality-1"><input type="radio" name="answer_quality" id="answer_quality-1" value="1" />Very poorly<br></label><p>&nbsp; </p></td>
    </tr>


    <tr>
        <td class="style4">How timely was your issue resolved?</td>
        <td class="style6">
            $Input -interactionProperty26'Answer Speed' -id'answer_speed' -customhtml -required$
<label for="answer_speed-5"><input type="radio" name="answer_speed" id="answer_speed-5" value="5" />Very timely</label><br />
<label for="answer_speed-4"><input type="radio" name="answer_speed" id="answer_speed-4" value="4" />Somewhat timely</label><br />
<label for="answer_speed-3"><input type="radio" name="answer_speed" id="answer_speed-3" value="3" />Neutral</label><br />
<label for="answer_speed-2"><input type="radio" name="answer_speed" id="answer_speed-2" value="2" />Somewhat timely</label><br />
<label for="answer_speed-1"><input type="radio" name="answer_speed" id="answer_speed-1" value="1" />Very timely<br></label><p>&nbsp; </p></td>
    </tr>

    <tr>
        <td class="style4">How would you rate the skills of the support representative?
        </td>
        <td class="style6">
      $Input -interactionProperty27'Agent Skills' -id'answer_skill' -customhtml -required$
<label for="answer_skill-5"><input type="radio" name="answer_skill" id="answer_skill-5" value="5" />Very skillful</label><br />
<label for="answer_skill-4"><input type="radio" name="answer_skill" id="answer_skill-4" value="4" />Somewhat skillful</label><br />
<label for="answer_skill-3"><input type="radio" name="answer_skill" id="answer_skill-3" value="3" />Neutral</label><br />
<label for="answer_skill-2"><input type="radio" name="answer_skill" id="answer_skill-2" value="2" />Somewhat inexperienced</label><br />
<label for="answer_skill-1"><input type="radio" name="answer_skill" id="answer_skill-1" value="1" />Very inexperienced<br></label><p>&nbsp; </p></td>
    </tr>
    <tr>
        <td class="style4">How friendly was the support representative?
        </td>
```

Change the interaction property values to match the IDs of the properties you've created.

Feedback Form HTML

⊟ Step 3 - Create the Feedback Form Action

When the feedback form is submitted, you will need to create an interaction that includes the content of the form. As with most question submissions, this will include two actions: find or create a contact submitting the form, and create the interaction for that contact. In most cases you will allow the user to submit their feedback anonymously, so you will submit the form as a user that you designate (e.g., feedback).


Actions:

Check form input name: (Blank to always run this action.) for value (regex):

For anonymous feedback, set the contact here rather than from the form input.

Find/Create Contact

Find By Email: feedback OR Find By Contact ID:

Property Group: *Specified with $input ... -group'xxx'$

☑ Leave existing values when properties in form are blank.

Result Name: contactID *Can be referenced in other steps as $result -id'name'$

Create Ticket/Ask A Question

☐ Agent Ticket (Unchecked for User Submitted Ask A Question)

Contact ID: $result -id'contactID'$ Contact Email Address: feedback

Parent Interaction ID:

Subject: Feedback provided Topic ID: 79

Body:

```
Feedback has been received for an iService support incident.

Feedback provided
------------------

Reference # of interaction: $form -id'ref'$
How well was your question answered? $form -id'answer_quality'$
How timely was your issue resolved?  $form -id'answer_speed'$
How would you rate the skills of the support representative? $form
```

Note:

Property Group: *Specified with $input ... -group'xxx'$

Assign To Agent ID:

Result Name: interactionID *Can be referenced in other steps as $result -id'name'$

The Action Step Used with Feedback Forms

Step 4 - Create a Stock Response to Generate the Feedback Link

Once your form is created, you can capture feedback at any time. To link feedback to a specific interaction, create a stock response that appends the Ref # of the agent's response using the format below for the URL.

```
https://[tenantURL].iservicecrm.com/Form.aspx?formID=[YourFo
```

In the example above, the ref variable will insert the reference number of the agent's email reply. This will be passed into the iService Feedback form automatically and allow you to associate the customer's feedback with the agent's reply.

Deploying Forms

6 Deploying Forms

Once your iService form is developed, you can deploy it in several ways. The three most common approaches are accessing the form directly, embedding the form into another website using an iFrame, and posting to the form from another website. You can also use the form web services to run forms and collect data for your custom applications.

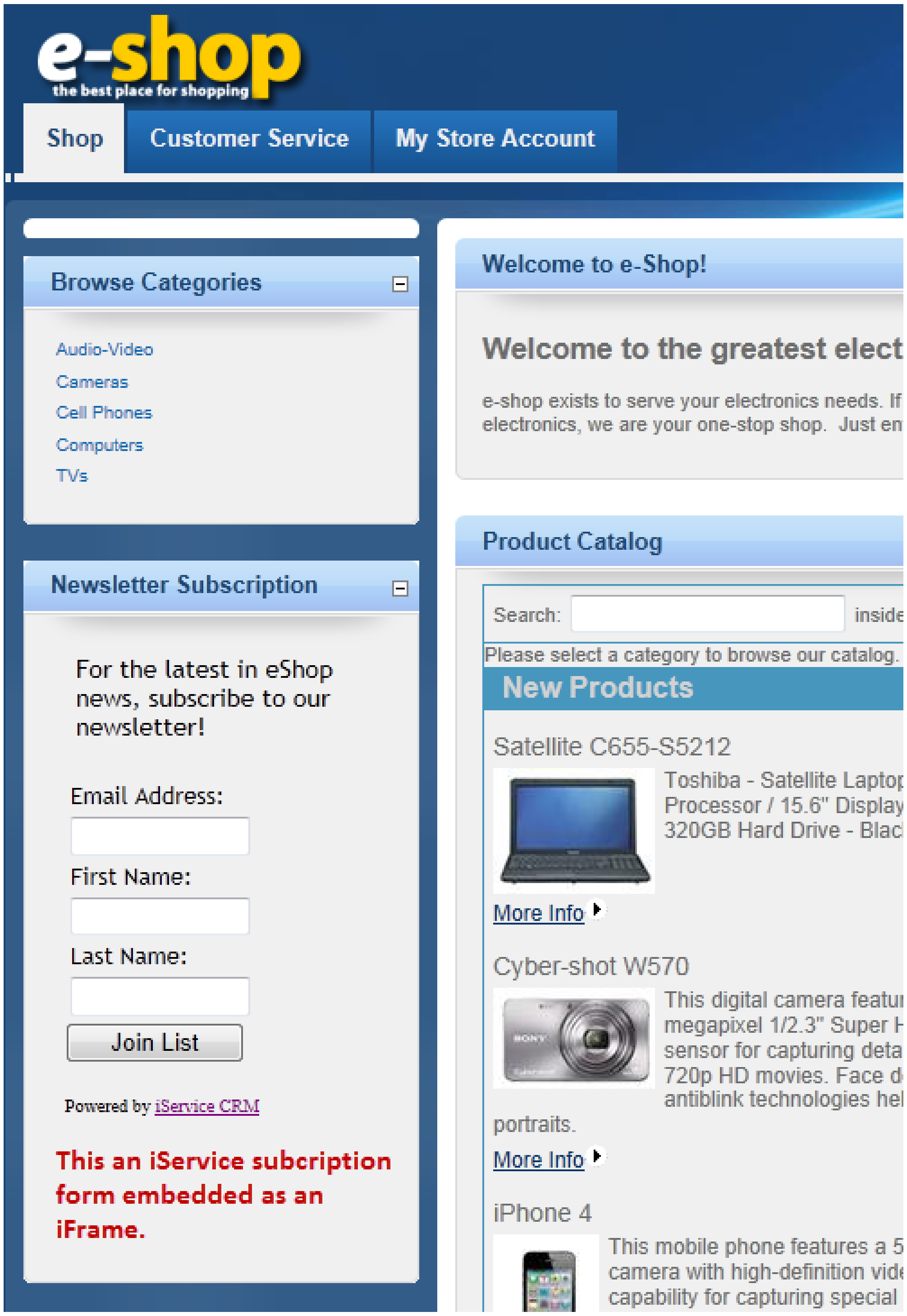
⊟ Accessing the Form Directly

The simplest way to use an iService form is to access it directly by appending /f/# to the URL of your iService website and referencing the form id for #. In this case, you would style the HTML directly within the form body to provide the desired look. The form body can contain any HTML tags desired. If you are integrating the form with an existing website, you can copy the HTML for the graphics, navigation, etc. directly into the form body.

Once your form is created, you can access it using the base URL of your tenant along with the form ID. For example, the following URL is a reference to a form on the 1to1service tenant of iService, where the form ID is 4. The form ID is displayed in the list of forms on the Admin Tools - Forms page.
https://1to1service.iservicecrm.com/F/4

⊟ Embedding the form with an iFrame

If you are inserting a small form into an existing website, you can use an iFrame. This is commonly done for forms like email subscription sign up and ask a question forms as shown below.



Welcome to the greatest elect

e-shop exists to serve your electronics needs. If electronics, we are your one-stop shop. Just en

For the latest in eShop news, subscribe to our newsletter!

Email Address:

First Name:

Last Name:

Powered by iService CRM

This an iService subcription form embedded as an iFrame.

Please select a category to browse our catalog.

Satellite C655-S5212

Toshiba - Satellite Laptop Processor / 15.6" Display 320GB Hard Drive - Blac

Cyber-shot W570

This digital camera featu megapixel 1/2.3" Super H sensor for capturing deta 720p HD movies. Face d antiblink technologies hel portraits.

iPhone 4

This mobile phone features a 5 camera with high-definition vide capability for capturing special

The HTML for inserting an iFrame into your existing website is as shown below.

```
<iframe src="insert the URL for your iService form here"></iframe>
```

Learn more about inserting pages into your site using an iFrame on the W3Schools website.

⊟ Posting to an iService form from another website

In cases where you have existing web forms and don't want to embed their HTML and graphics into your iService form, you can post the results from your existing website. This is done by setting the action within your current form similar to the following.

```
<form name="inputForm" action="https://1to1.iservicecrm.com/
```

This will post the contents of your existing form to the iService form specified. You will need to ensure that the field names (the -id values in your iService form) match exactly between the two forms.

HTML Snippets and Examples

7 HTML Snippets and Examples

From time to time we come across small scripts or form examples that are useful in making iService forms. This section includes a few examples.

iService also has example forms built into each tenant. See https://1to1service.iservicecrm.com/f/sample-index for usage examples.

7.1 Saving iService Form Data on Postback

iService will perform input validation for user input when you specify the -required and -confirm parameters. This validation is performed on the server and your page will reload with any error details specified within the $errormessage command. However, If you are using the -customhtml option, the page will reload without the values the user set when they submitted the form. **This is only an issue when using the -customHTML option**.
You can pass the submitted values returned by the server back to the -customerHTML fields by adding some additional code to the fields that are using -customHTML. This code will update the fields with the information passed back from iService after the validation occurs, preserving whatever the user entered.
To preserve the value of a standard input element, use the HTML value parameter as shown below. This tells the form that if the form reloads to populate the field with whatever value the server received (in this case, the last name supplied when the form was submitted).

```
$input  -id'lastname' -customhtml -required$
<input value="$form -id'lastname'$" name="lastname" type="text" />
```

The following example illustrates using the $If command to reload a value that was selected in a drop down menu.

```
<label>I have a question about:</label>
<div>$input -id'requesttype' -required -customhtml$</div>
 <select name="requesttype">
   <option value="">Please Choose One</option>
   <option $if -fieldregex'requesttype'='Product'$selected$endif$ va
   <option $if -fieldregex'requesttype'='Service'$selected$endif$ va
   <option $if -fieldregex'requesttype'='Other'$selected$endif$ valu
 </select>
</div>
```

The same example for a selection using radio buttons is shown below.

```
<div class="row">
 <label>I have a question about a:</label>
   <div>$input -id'requesttype' -required -customhtml$</div>
   <label class="options inline"><input type="radio" name="requestty
   <label class="options inline"><input type="radio" name="requestty
   <label class="options inline"><input type="radio" name="requestty
</div>
```

7.2 Using Rich Text Editor On Forms

You can use a rich text editor within any textarea on your forms. Since forms are generated as HTML pages, you can use any editor desired as long as you include the links to their javascript files. Beginning with v6, iService has a built-in editor based on CLEditor, and you can incorporate this into your forms as shown in the example below.

Insert into the HEAD of the form body:

```
<head>
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js"></script>
  <link rel="stylesheet" type="text/css" href="cleditor/jquery.cleditor.css" />
  <script type="text/javascript" src="cleditor/jquery.cleditor.js"></script>
  <script>
    function turnOnHtmlEditors() {
      $("textarea.htmleditor").each(function (i) {
        var editor = $(this).cleditor({
          width: "100%",
          height: "100%"
        });
      });
    };

    $(document).ready(function () {
      turnOnHtmlEditors();
    });
  </script>
</head>
```

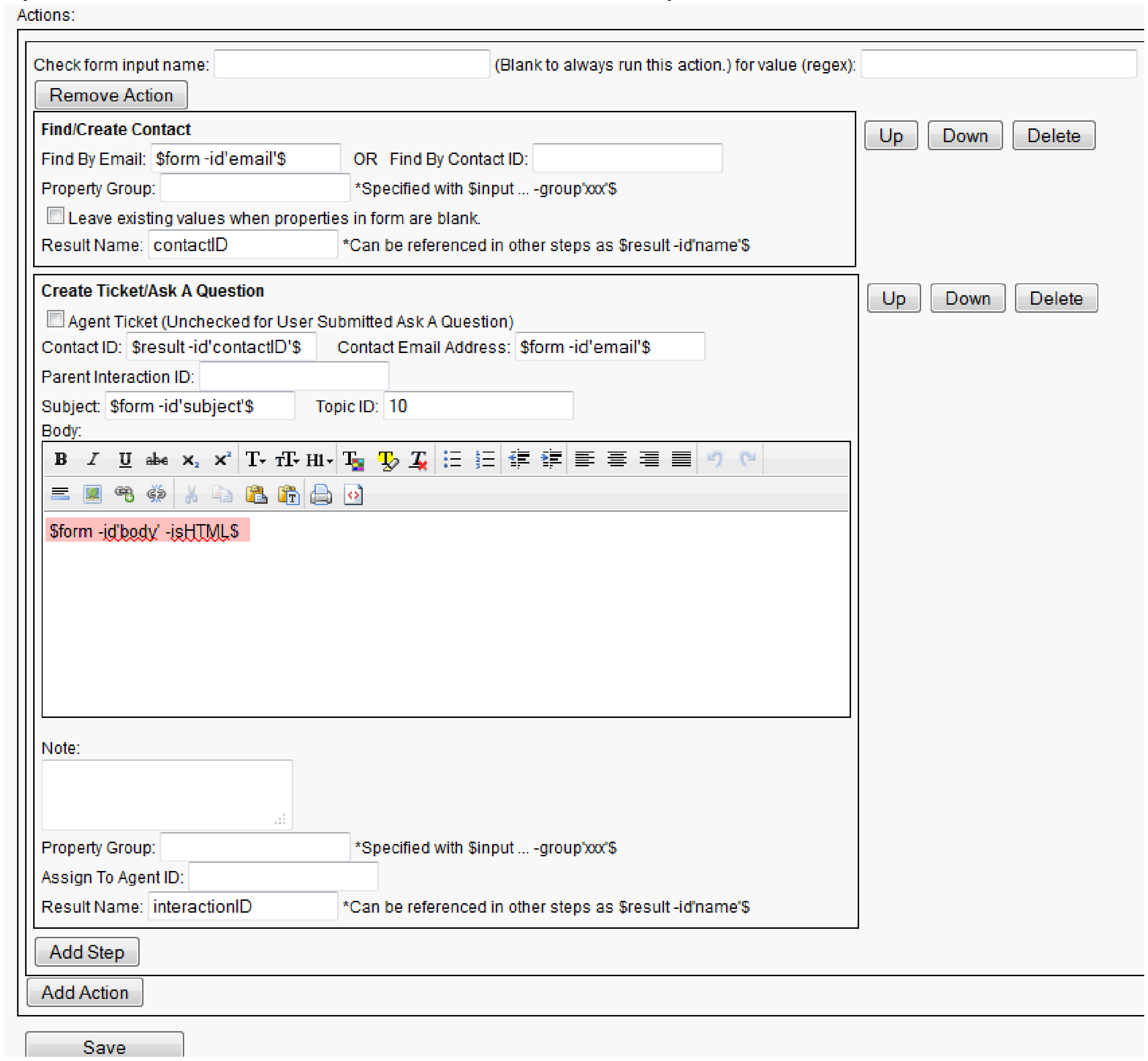
Use -customhtml and the class HTMLEDITOR in the body section of the iService Form Body:

```
<body>
  <form id="form1" method="POST">
    Email: $input -email -required -id'email'$ <br />
    First Name: $input -contactproperty1'first name' -id'firstname'$<br />
    Last Name: $input -contactproperty3'last name' -id'lastname'$<br />
    Subject: $input -id'subject'$<br />
    Body: $input -id'body' -textarea -customhtml$
    <div style="border: 1px solid #000; padding: 0px 2px 2px 0px;">
```

```
      <div style="width: 100%; height: 200px;">
        <textarea id="body" name="body" class="htmleditor">$form -id'body'$</textarea>
      </div>
    </div>
    <input type="submit" id="ok" name="ok" value="Submit Question" />
  </form>
  <div style="background-color:yellow;">$errormessage$</div>
</body>
```

<u>In the body of the action step (Ticket/AAQ/Note), indicate the input is HTML using -isHTML:</u>

Since the input into the rich text area will contain HTML markup, you will need to specify on the form action that the input is HTML. Use the -isHTML option to indicate that as shown in the example below.



Actions:

Check form input name: (Blank to always run this action.) for value (regex):

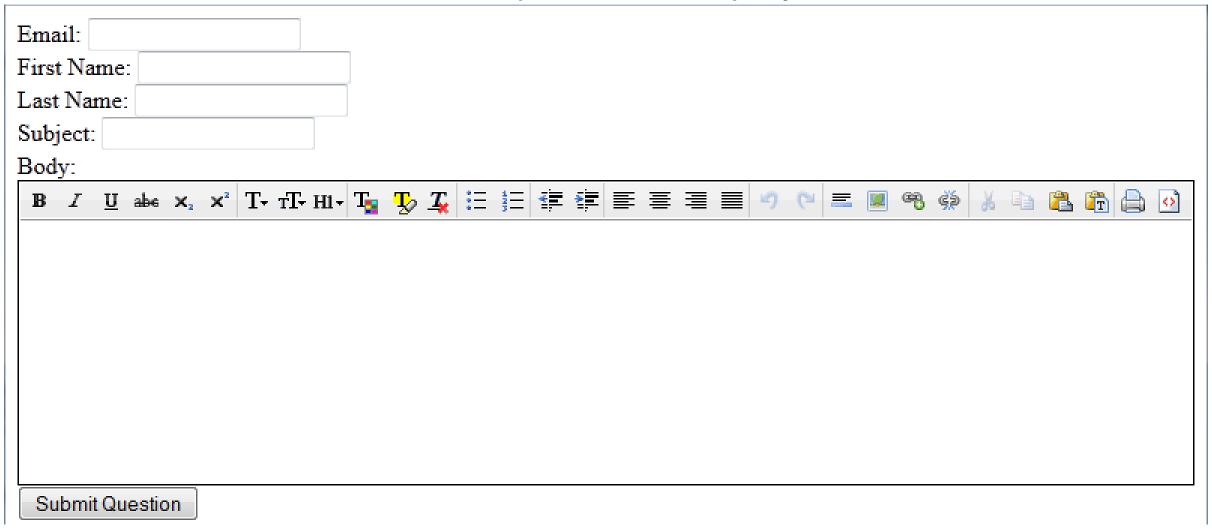
Find/Create Contact
Find By Email: $form -id'email'$ OR Find By Contact ID:
Property Group: *Specified with $input ... -group'xxx'$
Leave existing values when properties in form are blank.
Result Name: contactID *Can be referenced in other steps as $result -id'name'$

Create Ticket/Ask A Question
Agent Ticket (Unchecked for User Submitted Ask A Question)
Contact ID: $result -id'contactID'$ Contact Email Address: $form -id'email'$
Parent Interaction ID:
Subject: $form -id'subject'$ Topic ID: 10
Body:
$form -id'body' -isHTML$
Note:
Property Group: *Specified with $input ... -group'xxx'$
Assign To Agent ID:
Result Name: interactionID *Can be referenced in other steps as $result -id'name'$

Click to expand

Form Example

The form illustrated in this example would display as shown below.


Email:
First Name:
Last Name:
Subject:
Body:

Click to expand

7.3 Limit Number of Characters in a Form Field

This little JavaScript has been very useful. Thanks to http://www.mediacollege.com/internet/javascript/form/limit-characters.html for nice tight code.

This script allows you to set a limit on the number of characters a user can enter into a textarea or text field, like so:

Step 1 - Create Function for Head

Insert the following code into the page head:

```
<script language="javascript" type="text/javascript">
function limitText(limitField, limitCount, limitNum) {
     if (limitField.value.length > limitNum) {
          limitField.value = limitField.value.substring(0, limitNum);
     } else {
          limitCount.value = limitNum - limitField.value.length;
     }
}
</script>
```

Step 2 - Use -customHTML in the Form Body

Use the following code to create the text area (change the name of the text area to suit your needs):

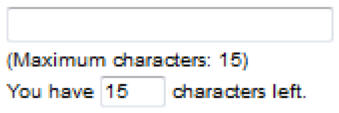
```
$input -id'limitedtextarea' -customHTML$
<textarea name="limitedtextarea" onKeyDown="limitText(this.form.limi
</textarea><br>
<font size="1">(Maximum characters: 100)<br>
You have <input readonly type="text" name="countdown" size="3" value
```

(Note: replace "LimitedTextArea" with the id of your form item)

To create a single-line text field instead of a text area, use the following code:

```
<input name="limitedtextfield" type="text" onKeyDown="limitText(this
onKeyUp="limitText(this.form.limitedtextfield,this.form.countdown,15
<font size="1">(Maximum characters: 15)<br>
You have <input readonly type="text" name="countdown" size="3" value
```

The result looks like this:


(Maximum characters: 15)
You have 15 characters left.

Note: You can have multiple text areas and/or fields on the same page using the same function. Just make sure you give each field a unique name.

7.4 JQuery Date Picker

Date pickers are very common in forms, and iService supports date fields within contact and interaction properties. If you are passing a date to one of these properties it should always be in the format YYYY-MM-DD.
The code below can be used to embed a simple date picker into iService forms.

```
<html>
<head>
<link rel="stylesheet" type="text/css"
href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.15/themes/redmond/jque
ry-ui.css" />
</head>
<body>

Enter Date: <input class="datepicker" type="text" name="StartDate"
id="StartDate" /><br />
```

```
$input -id'StartDate' -customhtml$


<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js"></script
>
<script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.15/jquery-
ui.min.js"></script>
<script>
$(document).ready(function () {
$(".datepicker").datepicker({ dateFormat: 'yy-mm-dd', changeMonth: true,
changeYear: true, selectDefaultDate: false, defaultDate: null });
});
</script>
</body>
</html>
```

7.5 Adding reCaptcha

Beginning with 7.8, iService includes support for Google reCaptcha to protect forms from automated submission. For information on setting up reCaptcha, visit the Google site at https://developers.google.com/recaptcha/intro. iService supports reCaptcha v2 (I'm not a robot and Invisible).

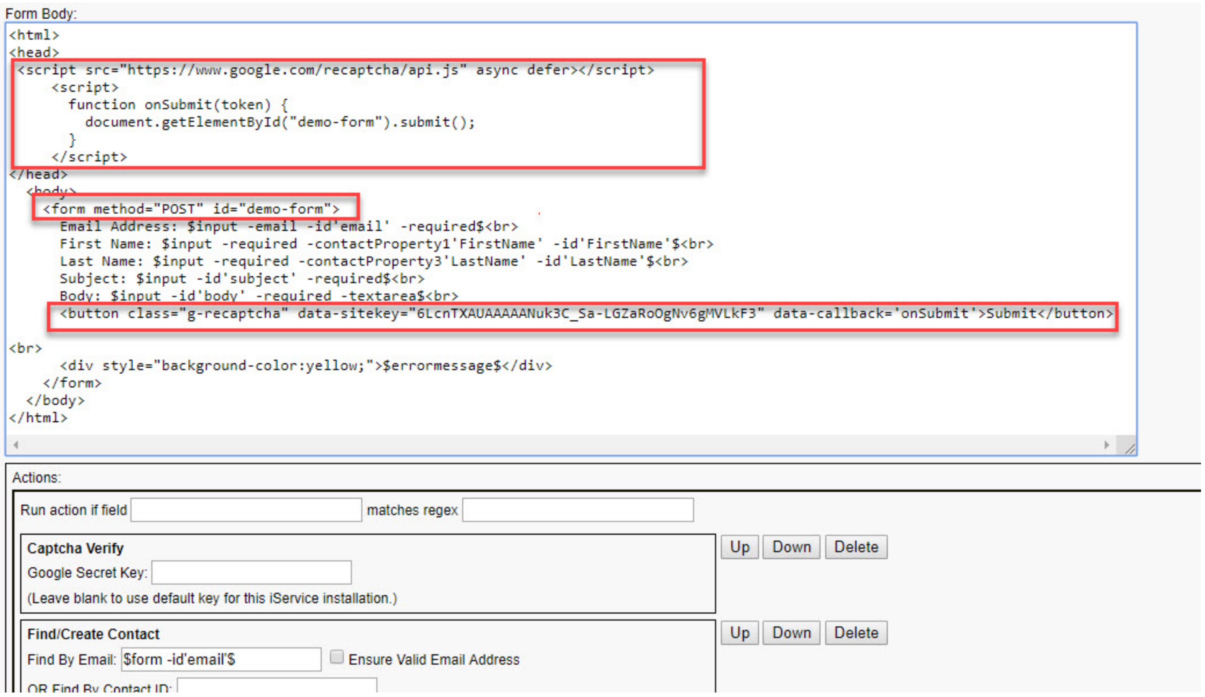
Prerequisites

Before implementing reCaptcha you need an API Key Pair, which includes a public Site Key and private Secret Key. These keys are generated by Google and associated with a specific domain. iService on-demand is configured with an API Key Pair created for iServiceCRM.com. To use the built-in iService reCaptcha support, you will only need to insert the Site Key into the form body of your form (`6LcnTXAUAAAAANuk3C_Sa-LGZaRoOgNv6gMVLkF3`). The Secret Key is already stored in the iService database. On-demand users that wish to use their own API Key pairs can override the built-in reCaptcha support by entering their own keys in their iService form.
iService on-premise users will need to register their domain with Google and generate their own API Key Pair before implementing reCaptcha. To update the iService master database with your company Secret Key, update the Master Database Settings table with a name of GoogleRecaptchaSecretKey and the Secret Key value. Users may then leave the vlue in the reCaptcha Verify action step blank. Otherwise, the Secret Key must be entered into the action step of every form.

reCaptcha Setup

There are two requirements for configuring reCaptcha in an iService form: Update the form body with required values (including a Site Key), and add the reCaptcha Verify step to the form action that creates your interaction (including a Secret Key). For configuration options related to the form body setup please visit the Google reCaptcha site at https://developers.google.com/recaptcha/intro.

An example of a form body updated with the reCaptcha changes is shown below for illustrative purposes only. See the Google documentation for configuration options.



Form Body:

```
<html>
<head>
 <script src="https://www.google.com/recaptcha/api.js" async defer></script>
     <script>
       function onSubmit(token) {
         document.getElementById("demo-form").submit();
       }
     </script>
</head>
   <body>
    <form method="POST" id="demo-form">
      Email Address: $input -email -id'email' -required$<br>
      First Name: $input -required -contactProperty1'FirstName' -id'FirstName'$<br>
      Last Name: $input -required -contactProperty3'LastName' -id'LastName'$<br>
      Subject: $input -id'subject' -required$<br>
      Body: $input -id'body' -required -textarea$<br>
      <button class="g-recaptcha" data-sitekey="6LcnTXAUAAAAANuk3C_Sa-LGZaRoOgNv6gMVLkF3" data-callback='onSubmit'>Submit</button>

<br>
      <div style="background-color:yellow;">$errormessage$</div>
    </form>
  </body>
</html>
```

Actions:

Run action if field matches regex

Captcha Verify
Google Secret Key:
(Leave blank to use default key for this iService installation.)

Find/Create Contact
Find By Email: $form -id'email'$ ☐ Ensure Valid Email Address
OR Find By Contact ID:

Example form body changes for reCaptcha