



Phone Mashups Developer Guide: How to Use the Phone Network with Your Web Applications



Table of Contents

1. Introduction to the Guide	5
1.1. What You Need to Know and What You Will Learn	5
1.2. Developer Accounts	6
1.3. Application Programming Interface (API)	6
1.4. About the Documentation	6
1.5. Roadmap	7
1.6. Ifbyphone on the Web	7
2. Why This is Great	8
2.1. Basic Idea: Looks Like the Web, Acts Like a Phone	8
2.1.1. Ifbyphone Fundamentals	8
2.1.2. Can't I Do This Myself?	9
2.2. Ifbyphone's Smart Forms: Additional Documentation	9
2.3. Basic Ideas about Smart Forms	10
2.3.1. Some are Persistent, Some are Disposable	10
2.3.2. Configuration: The Web Site and the API	10
2.3.3. Many Copies	10
3. Example Phone Mashup	13
3.1. Ifbyphone: Extending the Web	13
3.2. Our Example: Order Status	13
3.2.1. First Draft Solution	13
3.2.2. First Draft Solution: Timelines	15
3.2.3. First Draft Solution: Setting Up the Smart Form	16
3.2.4. First Draft Solution: Setting Up Your Servers	17
3.2.5. The Next Draft and Beyond	18
3.3. Wrap Up	19
4. API Basics	20
4.1. How Ifbyphone Interacts with Your Web Page	20
4.2. The Different Versions of the Ifbyphone API	20
4.3. How to Create Links with Ifbyphone's Wizard	21
4.4. Developers: Create Links Directly	22
4.4.1. GET or POST?	22
4.4.2. Parameters: How to Use HTTP to Send Data to Ifbyphone	23
4.4.3. Format of Complex Data	25
4.4.4. Phone Numbers	26
4.4.5. Receiving Data from Ifbyphone	26
4.5. Troubleshooting/Debugging	26
5. Security and Phone Mashups	28
5.1. Keys: Public, "API," and Private	28
5.1.1. Use of Keys	29
5.1.2. Lower Security: Use of "Public Key"	29
5.1.3. Higher Security: "API Key"	30

5.1.4. Other Security: Private Key	30
5.2. Account Number and Password	30
5.3. Registered Numbers	31
5.4. Summary of Precautions	31
6. "Click-to" Smart Forms API	32
6.1. Settings	32
6.2. Responses	32
6.3. Services	33
6.3.1. Click-to-Call	33
6.3.2. Click-to-Find Me	34
6.3.3. Click-to-Virtual Receptionist	39
6.3.4. Click-to-Voice Mail	39
6.3.5. Click-to-SurVo	39
6.3.5.1. Schedule Voice Broadcast	41
6.3.5.2. User Parameters: How to Create Customized Announcements	43
6.3.5.3. NetGet: How to Receive Data from a SurVo Smart Form	44
6.3.5.3.a. Configuration	45
6.3.5.3.b. Action	46
6.3.5.3.c. Submit Type	47
6.3.5.3.d. Domain	47
6.3.5.3.e. Page	47
6.3.5.3.f. Static Parameters	47
6.3.5.3.g. "SurVo-Generated Parameters": The Dynamic Parameters	48
6.3.5.4. Response to the SurVo NetGet from Your Server	50
6.3.5.5. Pseudo-code for the Destination URL	51
7. Reminder and Schedule	53
7.1. Reminder	53
7.2. Find Schedules & Set Schedule Mode	54
8. New Style Web Requests	58
8.1. Format of New Style Web Requests	58
8.2. Voice Mail	59
8.2.1. Create Voice Mailbox	59
8.2.2. Get Messages	60
8.2.3. Record Voice Mail Greeting	62
8.3. Report	63
8.4. SurVo	65
8.5. Verify-Me-Now	66
8.6. Find Me	68
8.7. Voice Broadcast	81
8.8. Conference	89
9. Appendix: Web Service Code Samples	109
9.1. ASP Example	109
9.2. Example: Basic PHP Sample	110

9.3. Example: Advanced PHP Sample	110
9.4. Example: Python	116
10. Appendix: Ifbyphone Glossary	119

1. Introduction to the Guide

This guide shows how to use Ifbyphone's services to integrate World Wide Web applications with the public telephone network. We call these applications "phone mashups."

Here's a quick sample of some of the basic capabilities provided by Ifbyphone:

- Make an outbound phone call and play an announcement.
- Answer an inbound call and ask a question.
- Verify a caller's identity via a PIN number.
- Record a caller's answers to a survey questionnaire.

Here are a few examples of interesting phone mashups built on top of these basic services:

- "Click to call" links. When the user clicks on the link -- on a web page or in an email -- the user can make a phone call to your office.
- Conference calls. The user puts check marks next to the names of people to include in a phone call. When the caller clicks "OK," Ifbyphone calls everyone on the list and includes them in the call.
- Surveys. The user can create an outbound calling campaign by uploading a list of phone numbers and clicking "Start survey." Ifbyphone calls everyone on the list and asks survey questions.

How hard is it? At the simplest level, Ifbyphone is just a link -- you can use a Wizard on Ifbyphone's web site to generate links to Ifbyphone services; you can then paste these links into email or include them on web pages. More experienced developers can create web pages that combine the usual web design elements with Ifbyphone's services. Many of our developers also use their web expertise to create telephone-only services.

In Chapter 3, we'll demonstrate an interesting phone mashup in detail.

1.1. What You Need to Know and What You Will Learn

If you'd just like to create a link to an Ifbyphone service to place in your email, or if you want to cut and paste a link and place it on your web page, you can use Ifbyphone's Wizard to create the correct HTML code. Please see Section 4.3 for details.

For most other uses, we assume that you know the basics of how to write web pages. We don't assume any particular level of web expertise: some services are

available by simply cutting and pasting HTML from a form on the Ifbyphone web site into your web page. Other services require more advanced web design skills.

This Guide explains each service available from Ifbyphone, how to use that service, how to send information to Ifbyphone over the web, and how to receive information back from Ifbyphone.

We provide examples (see Appendix A) that show the basics of how to incorporate Ifbyphone services into web pages if you use PHP, ASP, and other web programming languages.

Developers can use the Phone Mashup Developers Guide to build complex voice applications. For example, a developer can create a chain of "SurVo" blocks (see Section 6.3.5) and control, through the web, the order in which the caller interacts with the blocks.

1.2. Developer Accounts

In order to create Ifbyphone services, you need a developer account. Ifbyphone offers developers free accounts along with 100 free minutes of telephone connection time each month -- plenty of time to test and develop applications. Once you've created your service, you can upgrade your account to place your phone mashup into production.

Paid premium accounts automatically include access to the Ifbyphone phone mashup API. To check whether you have access to the API, log into the Ifbyphone web site and go to Tools->Building Block Ids. If you see a list of "keys" at the top of the page, you may use the API. Otherwise, please contact your Ifbyphone account manager for more information.

1.3. Application Programming Interface (API)

"Application Programming Interface," or API, is the technical term for what this document describes. The API lets you, the developer, access Ifbyphone's services by following certain rules and providing certain information. This Guide describes the services Ifbyphone provides, what information you can send and receive, and other information you need to access Ifbyphone services over the web.

1.4. About the Documentation

The API (Phone Mashup) Developers Guide includes the following documentation:

- Descriptions of the Smart Click-to-Call and the Administrative APIs
- Code samples
- Screenshots and helpful tips for productivity

1.5. Roadmap

Here's a roadmap to the rest of the document.

If you are...	Then Read This...
Beginner	If you're not familiar with Ifbyphone's services or what they do, we recommend that you read all the chapters in this Guide in order.
Intermediate	If you're familiar with Ifbyphone's services through the web site, but haven't created Phone Mashups, you should read all the chapters in order.
Advanced	If you've created Ifbyphone Phone Mashups before, you can skip the first chapters and proceed directly to Chapter 5, "Security and Phone Mashups."

1.6. Ifbyphone on the Web

- Specialized information, as well as our developers forum for phone mashup developers is available at www.phonemashup.com.
- Ifbyphone's Web site (www.ifbyphone.com) contains:
 - Your developer account home page
 - Documentation, such as additional user guides and tutorials
 - Business-related details, such as the price of calls for paid accounts
 - Customer service contact information

You'll find technical and business posts in our blog at <http://public.ifbyphone.com/blog/>. You may also wish to follow our Twitter account, @Ifbyphone.

2. Why This is Great

We're not embarrassed to say that we think our products are great. In this chapter we'll review what Ifbyphone provides and why you'd want to use it.

2.1. Basic Idea: Looks Like the Web, Acts Like a Phone

Ifbyphone's basic business is to make the public telephone network act like an extension of the World Wide Web.

With Ifbyphone you, the web developer, don't need to learn anything special about the phone network to integrate telephone-based interactions into your Web-based applications. Better yet, you don't need to learn any special programming languages; just use standard web-based programming and Ifbyphone handles all the arcane stuff.

Here's an example. Let's say you have a web page with a form that people fill out with their account number and PIN. Now let's say that you'd like people to be able to call into a phone number and give you that same information.

- *On the web*, you use a "form" element on the web page. People fill out the fields on the web page, hit the "submit" button, and you receive the information via GET or POST.
- *Using Ifbyphone*, people call a phone number and say their account number and PIN. You receive the information via GET or POST.

And that's what makes Ifbyphone great: As far as your web programming is concerned, it doesn't make a difference if the data are collected by typing on a web page or speaking into a phone. The data arrive the same way at your web server, the data look the same, and your web-based application remains the same.

Like any other tool, Ifbyphone's services are quite flexible, and you can do some pretty amazing stuff. Of course, once you build a complex service around the telephone network, you move beyond simple "replace that web page" programming and your application will change accordingly. Regardless, you can still use standard web programming methods. From your perspective, Ifbyphone acts like a supercharged web browser.

2.1.1. Ifbyphone Fundamentals

Ifbyphone can accept incoming telephone calls and make outgoing telephone calls. Both incoming and outgoing calls can:

- Play announcements
- Make recordings

- Use DTMF (touch-tones) and speech recognition to accept input
- Use text-to-speech to play announcements
- Detect the difference between a person and an answering machine

Beyond this, Ifbyphone has an entire suite of "Smart Forms." Smart Forms provide tremendously convenient ways to accomplish common tasks. Need to send a pre-recorded message to a thousand people? Use the Voice Broadcast Smart Form and upload a list of telephone numbers. Want to accept incoming telephone calls and route them to the correct department? Use the Virtual Receptionist Smart Form. And so on and so forth for a wide variety of very interesting interactions. This Guide explains how to integrate these interesting Smart Form building blocks into your applications.

2.1.2. Can't I Do This Myself?

People sometimes want to know if they can do, by themselves, what Ifbyphone does.

Certainly you can. You'll need to get your speech technologies up and running: speech recognition, text-to-speech, and a platform to support them. You'll need to learn the specialized programming languages that support speech technologies. (Even if you don't use speech technologies, just DTMF, you'll still need most of this same infrastructure.) Verify that all this works correctly and come up with a method to test it periodically.

Next you'll need to connect to the telephone network. Find some telephony providers, test extensively for speech technology compatibility, and negotiate a contract. Use at least two providers (possibly more) in order to set up a properly redundant, always-up system. Again, learn how to program these telephony connections using the specialized programming languages that support telephony. Purchase blocks of regular and toll-free telephone numbers.

You'll need a data center to host all your servers and keep them running 24/7 in the face of power outages and other natural disasters -- in fact, you should really have at least two data centers, geographically dispersed. Don't forget to include physical security -- you need to guard the machines to prevent physical theft, not just data theft.

Once you've got all that in place you can start to replicate Ifbyphone's Smart Forms, support team, documentation, and auditing and billing services. Good luck!

2.2. Ifbyphone's Smart Forms: Additional Documentation

This Guide gives complete details about how to use Ifbyphone's Smart Forms in phone mashups, but it does not explain all the details and all the capabilities of each Smart Form. To learn more about a particular Smart Form, log into your account on the web site. Select the name of any Smart Form from the "Services" or

"Tools" menu and you will go to the Smart Form's configuration page, and that page will have links to further documentation about the Smart Form.

2.3. Basic Ideas about Smart Forms

Below you will find some basics instructions that explain how to use Smart Forms. (A word to the wise: to a computer scientist, Smart Forms are "objects.")

2.3.1. Some are Persistent, Some are Disposable

The API for the Smart Forms are all a little different from each other. Read this Guide to understand each individual Smart Form.

Some Smart Forms are persistent: When you create one of this type of Smart Forms it hangs around, essentially, forever; when you want to use it for a particular purpose you configure it with some extra data. For instance, you'll find that a Voice Broadcast Smart Form persists once you've defined it. To use it you add a "campaign" to the Smart Form with a list of telephone numbers to call as well as a schedule of when to call them.

Other Smart Forms are used just once. An example of this would be a Conference Smart Form. A Conference has a start time and a duration; once that time has passed the Conference can no longer be modified or re-used.

Some Smart Forms reference other Smart Forms. For example, you can create a Schedule Smart Form that assigns each hour of the day a particular tag; e.g., "Open," "Closed," and "Lunch Time." Another Smart Form can use that Schedule Smart Form to take actions based on the time of day.

2.3.2. Configuration: The Web Site and the API

Another difference to watch out for is variations in how to create and configure Smart Forms.

Ifbyphone continues to define new Smart Forms and improve this Phone Mashup API. These latest additions are fairly complete: you can create a Smart Form, modify it, and use it, all without using the Ifbyphone web site.

The API for some of the earlier Smart Forms, however, is not as complete. In particular, you'll notice that some of the Smart Forms cannot be created, completely configured, or fully accessed if you use only the API. For full access to these earlier Smart Forms, you'll need to log into the Ifbyphone web site. Even so, these earlier Smart Forms have proven to be very successful in web applications.

2.3.3. Many Copies

Just for the sake of clarification: you aren't restricted to a single Smart Form of a given type. If you need twenty-seven slightly different Virtual Receptionist Smart Forms, by all means create them.

Since you will no doubt have many Smart Forms, Ifbyphone assigns each one a unique Building Block ID. You will often need that ID in order to use the Phone Mashup API.

You will also need several "keys," as we explain in Chapter 5, "Security."

To obtain the ID for any of your Smart Forms, login to the Ifbyphone web site, and from your home page select Tools -> Building Block IDs (Figure 2 1).

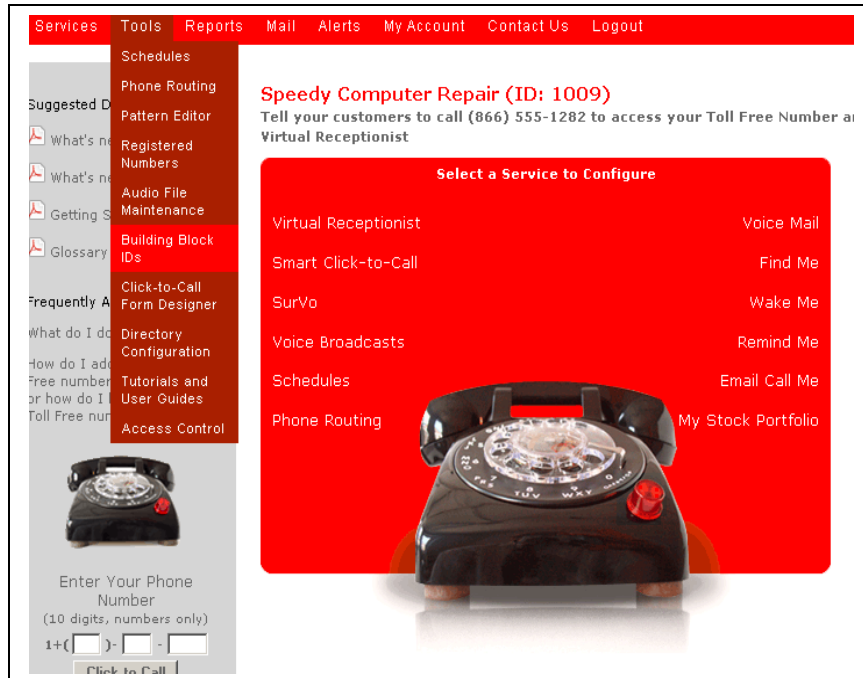


Figure 2 1: Menu item: Building Block IDs

Your Building Block IDs will be listed by type of Smart Form (Figure 2 2). You will also find your Public, Private, and API keys.

Building Block IDs for 'Greenberg Inc.'

You can use the values listed in each Building Block's ID column for techniques described in the [Advanced Techniques for Developers](#) tutorial, as well as for utilization in importing [Phone Directory](#) items from CSV files, providing you with complete control over intelligent transfers to telephone numbers and supported Building Blocks of that service in one step.

Your Public Key (use with click_to links) =
296[REDACTED]361

Your API Key (use with api calls) = ccd1[REDACTED]90e

Your Private Key = 077[REDACTED]9e6

Virtual Receptionist Menus (app: CTVR)

ID	Menu Name
62	WELCOME
70	ADAM GREENBERG
17	VACATION
80	SALES GUY ONE
80	SALES GUY TWO
80	SALES GUY THREE
21	TRANSFER
22	NEW MEXICO

Find Me Lists (app: CTF)

ID	List Name
43	FIND TECHNICIAN
54	MOTHER

Figure 2 2: Smart Click-to-Call Object IDs

3. Example Phone Mashup

In this chapter, we show an example of an interesting phone mashup. If you're reading this Guide in chronological order, you won't have studied any of the Smart Forms that we use to build the mashup. That's fine; our goal is to show you the typical interactions between Ifbyphone and your servers -- and get you interested enough to read up on the Smart Forms.

3.1. Ifbyphone: Extending the Web

As we mentioned earlier in this Guide, at its simplest Ifbyphone lets you replace web-based forms with a phone call. Your caller can dial a telephone number (or receive a phone call) and answer questions. The answers become text and arrive at your servers in the exactly the same format you'd receive from a Web-based form.

Ifbyphone also offers richer, more complex services. In this example we'll show how to use them -- without getting bogged down in details. The goal is to give an overview, not to examine each detail.

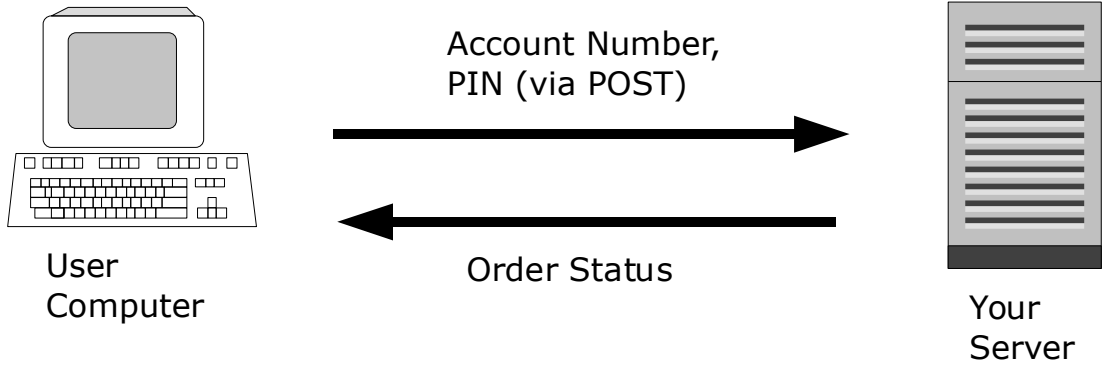
3.2. Our Example: Order Status

Your sales team is on the road and they want an order status. Although you have a web-based form they can use, when they're driving or at the airport, they always call in to ask questions instead.

By using Ifbyphone Smart Forms, you can create a telephone-based interface to your database. You start out by replacing the web page with a simple query, and quickly find that with some extra work you can provide a very useful application that does even more.

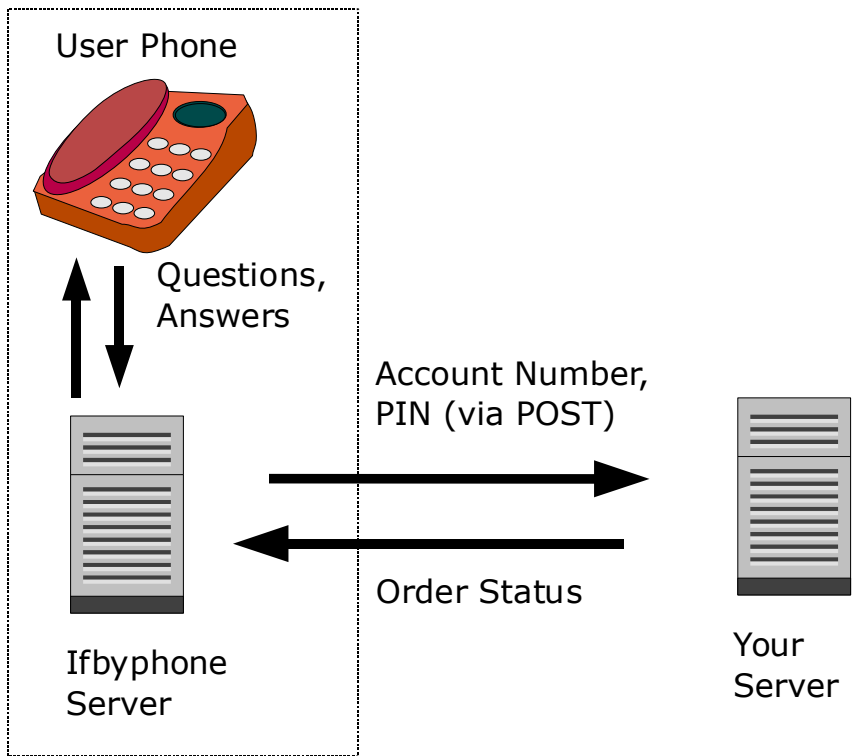
3.2.1. First Draft Solution

The first draft solution is quite simple: to replace the web page form with a Smart Form. Take a look at the following before and after diagrams. First, the current solution:



Drawing 3.1: Web-based Order Status

Next, a first-draft solution for a telephone-based solution.

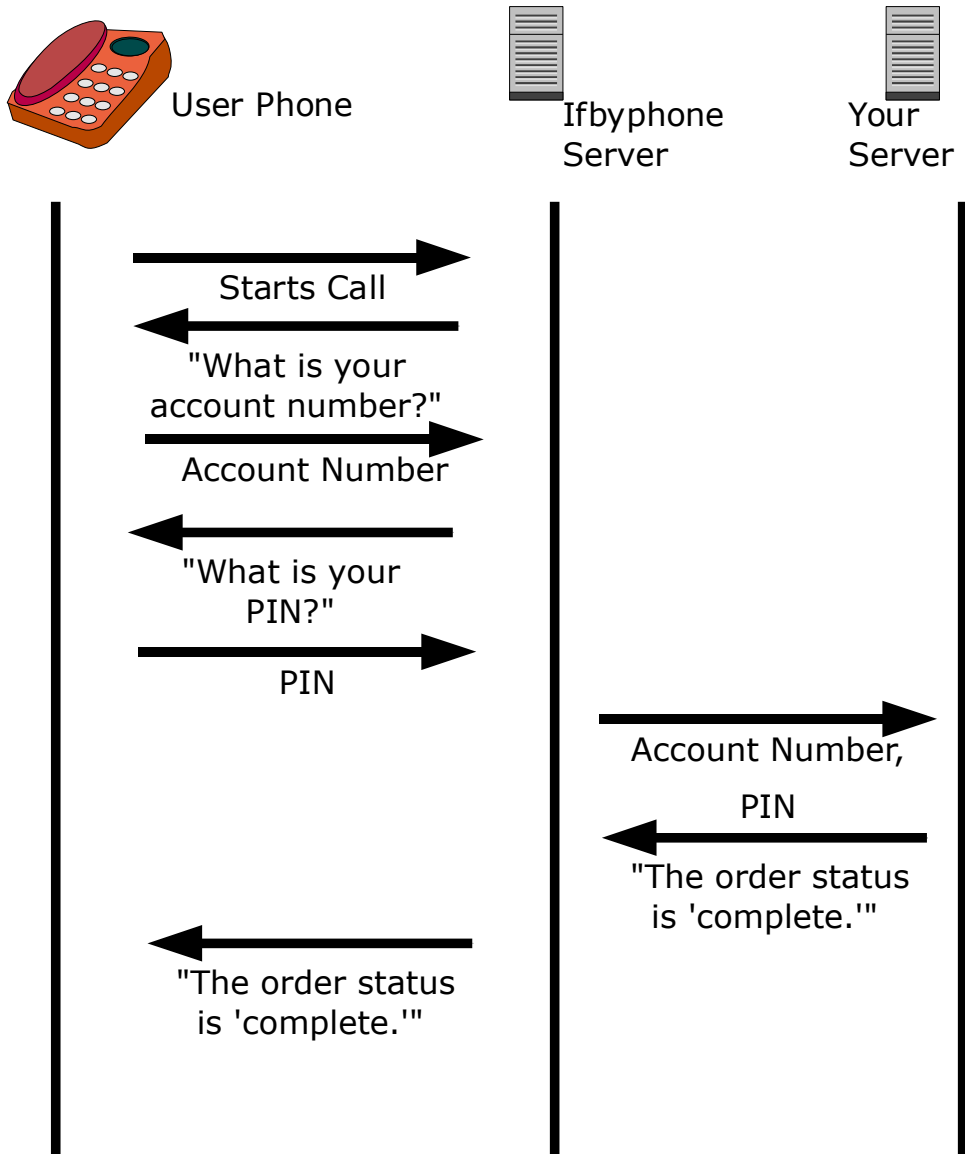


Drawing 3.2: Telephone-based Order Status

In the previous drawing (3.1), we had a "User Computer." In this solution, we replace the keyboard and display screen of that computer with a combination of the User Phone and the Ifbyphone server. For input, the caller can use speech or the keypad of the phone; for output, the caller hears recordings or text-to-speech. Your server still receives the information via POST, but from Ifbyphone's servers.

3.2.2. First Draft Solution: Timelines

In the next few paragraphs you will learn details about the first-draft solution. First, you will see a diagram of how information flows between the caller, Ifbyphone, and your server.



Drawing 3.3: Timeline of Data Flows for First Draft Solution

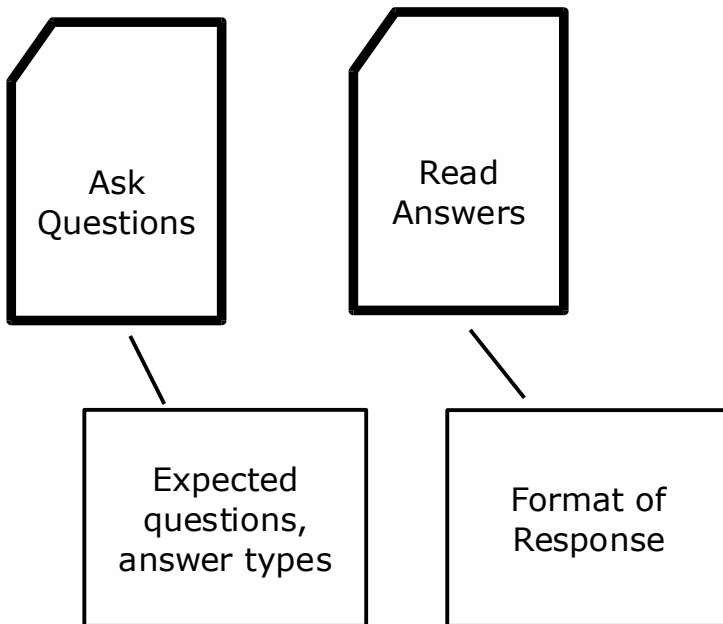
Here's a text explanation of the timeline:

1. The caller dials a phone number of the application. That phone number is hosted by Ifbyphone.

2. The call activates a Smart Form (more details on the Smart Form, below). The Smart Form plays a recording or text-to-speech to ask for the account number.
3. The caller says the account number.
4. The Smart Form asks for the PIN.
5. The caller says the PIN.
6. The Smart Form sends the account number and the PIN from Ifbyphone's server to your server.
7. Your server sends (properly formatted, see below) the response, "The order form status is 'complete.'"
8. The Smart Form uses text-to-speech to play that response to the caller.
9. Either the caller hangs up or the Smart Form asks for a new account number and PIN.

3.2.3. First Draft Solution: Setting Up the Smart Form

Later on in the document we'll discuss the details of the various types of Smart Forms. In the meantime, here's what you have to do at the Ifbyphone web site in order to set up this solution.



Drawing 3.4: First-Draft Solution: Two Smart Forms and their Configuration Data

The Smart Forms shown above are called "Ask Questions" and "Read Answers." They're called "SurVo" Smart Forms for historical reasons, but a better name would probably be "Question and Answer Smart Forms." We'll use these to ask questions and collect answers, or to just read text to the caller.

1. Create a pair of SurVo Smart Forms.
2. Configure the first SurVo Smart Form with questions:
 - a) Enter the text of the questions, and/or a recording of the questions.
 - b) Configure the SurVo with the type of answer to expect, e.g., a four-digit number.
 - c) Enter the Internet address to which the responses, account number, and PIN should be sent.
 - d) Associate a telephone number with this SurVo Smart Form.
3. Configure the second Smart Form with the format of the response you expect to read back to the caller. You might, for example, simply format the response to be the exact text sent by your server, or you might decide to have a standard announcement and fill in the blanks with words you send from your server.

That's about it. The web site has forms to configure the SurVos and to get a phone number assigned.

3.2.4. First Draft Solution: Setting Up Your Servers

As we mentioned earlier, the input into your server looks just like the input you'd expect from a Web-based form.

The **output** from your server, however, will be different. You're not providing HTML for a web page; you're providing some text and instructions to the SurVo Smart Form.

However, the output you send is far less complicated than HTML. The output is a short XML form; all major web programming languages support XML. The output contains a couple of pieces of information:

1. The ID of the SurVo that should read the information back to the caller.
2. The text you want to have read to the caller.

In the previous section we defined *two* SurVo Smart Forms. You respond to the HTTP request from the first Smart Form with this output, and that first SurVo Smart Form will activate a second SurVo Smart Form you designate. That's a very powerful system, because it means you can decide, dynamically, what Smart Form to use next. For example, if the caller made a mistake and gave you an invalid PIN

or invalid account number, you can direct the call to a separate SurVo Smart Form that handles errors.

The XML output will look something like this:

```
<action>
  <app>survo</app>
  <parameters>
    <id>3987</id>
    <user_parameters>
      <status>The order status is "complete."</status>
    </user_parameters>
  </action>
```

In other words, this output tells Ifbyphone to use SurVo Smart Form with ID 3987 to read back the words "The order status is 'complete.'"

Later in this Guide we'll discuss in detail how the SurVo Smart Form sends information to your server (see section Error: Reference source not found).

3.2.5. The Next Draft and Beyond

As we've seen, the SurVo Smart Form (like many other Smart Forms) has an option to transfer control to a different Smart Form. That means you have the same ability for complex tasks that you would have on the Web. On the Web, you can present your users with a series of forms to lead them through a complex set of related tasks; you can do the same thing with Smart Forms.

Returning to our example, the next draft of this application might include an option to update the status. How would that work?

First of all, we'd modify the "Read Answer" Smart Form to read back the answer and then ask the caller if the caller wants to submit an update. And the choices for the update wouldn't have to be fixed; you could send the text of the choices along with the text of the response:

```
<action>
  <app>survo</app>
  <parameters>
    <id>3987</id>
    <user_parameters>
      <status>The order status is "complete."</status>
      <choice1>Closed</choice1>
      <choice2>Open</choice2>
      <choice3>Deleted</choice3>
    </user_parameters>
  </action>
```

The modified "Read Answer" SurVo would take these choices, present them to the caller, and ask the caller to choose. The SurVo would then send the choice to

your server. Your server would respond with the ID of the next SurVo to activate any special text to read to the caller. And so on and so forth as the call progresses.

Instead of answering the call with a SurVo Smart Form, you might choose a Virtual Receptionist Smart Form to answer the call ("Say 'clerk' to speak to a clerk, or say 'order status' to inquire about an order"). The Virtual Receptionist Smart Form would route you to either a clerk, a Voicemail Smart Form, or to the SurVo Smart Form for order status inquiries.

3.3. Wrap Up

To summarize what we've learned:

- The information that Ifbyphone collects from callers is sent to your servers in the same format as the one sent by web pages.
- Your response to Ifbyphone is sent in XML format and can include the next Smart Form to activate as well as customized announcements.
- Smart Forms can transfer control of the call to other Smart Forms; the Smart Form that lets you ask and answer questions (SurVo) lets you control -- dynamically -- which Smart Form receives control of the call next. This lets you create very rich applications.

4. API Basics

This section discusses the basics of how to use the Ifbyphone Phone Mashup API.

4.1. How Ifbyphone Interacts with Your Web Page

You send and receive information from Ifbyphone by using the HTTP protocol -- the same protocol you currently use to send and receive web pages. This makes using the Ifbyphone web services very simple indeed. In fact, because all the information is sent and received over the web, for many useful services you don't even need to have a web page -- your callers can simply click on a link in an email, and the information will go to Ifbyphone's web service and start a phone call.

As for what this means as you develop your own web pages, there are three methods to incorporate Ifbyphone services in your web pages:

1. You can use the "Wizard" on Ifbyphone's web site to create a link that operates a Smart Form. You can then cut and paste that link (the HTML code) onto a web page or into your email.
2. The next chapter of this guide provides very detailed information on how to format a link ("web service call," to be more precise). Instead of using "Wizard" to write the code for a link, you can use this information to write links that are highly customized to your needs and use them to operate your Smart Forms.
3. If you use PHP, ASP, or similar server-side programming languages to generate your web pages, you can make HTTP calls from within that language. This method is the most flexible way to use Smart Forms because you can interact with your Smart Forms through HTTP while at the same time taking advantage of all the interesting things that server-side programming languages can do when it happens on your servers.

Finally, we should mention something on the expert level: an experienced programmer can use any number of standard programming languages (Python, Perl, Java, and Ruby, to mention a few) to interact with Ifbyphone services. Ifbyphone's services are called "web services" because they're accessible over the web, but that doesn't mean a web page must actually be involved.

4.2. The Different Versions of the Ifbyphone API

Like all web services, we here at Ifbyphone worked hard to improve the Phone Mashup API. At the same time that we created new services and improvements we had to make certain that our current customers didn't have to re-write their phone mashups. As a result, the Phone Mashup API has three distinct looks, depending on what services you're using:

1. If you generate services using the Wizard, you'll get one type of URL to cut and paste into your code.
2. If you develop Phone Mashups using web services with names that start with "Click-to," you'll use the "click_to_xyz" form of the URLs.
3. If you develop services using our newer web services, you'll use the "object" style URLs, which are easier to read at a glance.

We'll describe each type in detail later in this Guide, but it's important to keep the basic principles in mind as we go through examples. Different services use slightly different URLs, but the overall concepts are the same.

4.3. How to Create Links with Ifbyphone's Wizard

Ifbyphone's Wizard will create links for the following types of Smart Forms:

1. Click-to-Call
2. Click-to-Survo
3. Click-to-Voicemail
4. Click-to-Virtual Receptionist
5. Click-to-Find Me


To use the Wizard, you must first create a Smart Form of the appropriate type. Let's assume you want to create a link that users can click to make a phone call to your sales department. To get started, create a Click-to-Call Smart Form; Smart Forms have names -- we'll call this one "Sales Department."

After you've created the Smart Form, while logged into the Ifbyphone site, you can use the Wizard to create the link. To start, go to the "Services" menu and select which service you're interested in. In our example, that would be the Click-to-Call service. You will then get a list of the Smart Forms you've created for that service. In this case, we'll see the "Sales Department" Smart Form.

Next to the Smart Form is a small icon marked "code." Click on this icon. You'll next answer whether you want to use the Wizard to create a web page link or a link suitable for email. Select which type you'd like; the next page will give you a choice of links; Illustration 4.1 shows the output of the Wizard, which also offers several options to customize the links.

Developers can use the links generated by the Wizard, in conjunction with the documentation later in this Guide, as a starting point to create more highly customized links.


Image Pop-up (HTML/Javascript) Code for putting a text or graphic link on your page that opens a new small window to capture the user's phone number.



```
'Clickto' , 'toolbar=no,location=no, menubar=no, scrollbars=no, copyhistory=no,resizable=yes')">

</a>
```

Input Box (HTML/Javascript) Code for putting a input box and button on your page for capturing the user's phone number. When submitted, a small browser window with a page from the ifbyphone site will open with call status. With this approach, the user's current page remains visible in the browser.



```
<input type="text" id="phone" name="phone" maxlength="10"
size="10"><br>
<input type="submit" value="Click-to-Call"
onClick="window.open('http://www.ifbyphone.com/clickto_status.php?click
+ getElementById('phone').value, 'Clickto' ,
'width=200,height=200,toolbar=no,location=no, menubar=no,
```

Just the URL This is just the URL for getting to the page. If you are using a website builder it may only want the URL.

```
http://www.ifbyphone.com/clickto_getphone.php?click_id=109
```

LINK

Illustration 4.1: Screenshot of Code Created by Wizard

4.4. Developers: Create Links Directly

In the previous section we discussed how to use the Wizard to create links suitable for use on a web page or via email. These links work because if a user clicks on a link in an email or on a web page, the user's computer sends the information contained in the link via HTTP to Ifbyphone's servers through the user's web browser. For most of the Wizard-generated links, Ifbyphone's servers respond with a web page.

As we stated earlier, Ifbyphone's servers send and receive information via the HTTP protocol, but they don't require that a web browser be involved. In this section, we'll discuss how to use HTTP to use Ifbyphone's web services. You can use these instructions to build visually oriented mashups or to build ones that don't use web pages at all.

In this section, we assume that you are familiar with how to use HTTP and how to write the HTML code for a URL.

4.4.1. GET or POST?

There are two main methods to send information via HTTP: GET and POST.

- GET includes the information in the URL itself; this is by far the most common method of writing a URL. GET puts all information into the URL, actually as part of the URL itself, and that information can be read by anyone who can see the URL.

- The other method to send information via HTTP is through POST, in which the information is transmitted separately from the URL. Using POST is a bit more difficult as it requires some HTML coding or some programming. On the other hand, by using POST you can send information securely, because the information isn't part of the URL. We'll discuss why some of your information should be kept secret in Chapter 5.

Ifbyphone supports both GET and POST. The Wizard creates GET-style links, both because GET is ideal for cut-and-paste of links into a web page or email, and because none of the services that you access with links generated by the Wizard require any "secret" information.

4.4.2. Parameters: How to Use HTTP to Send Data to Ifbyphone

The tables in Chapters 6 and beyond list all information you need to send to Ifbyphone to use various services. As a developer you can use GET or POST to send this information, but you should use POST (and HTTPS) whenever possible and certainly when you use your API key.

The information you send to Ifbyphone through GET or POST is called a "parameter." A parameter has two parts: a "name" and a "value." In a GET-style URL, a parameter will appear as "name=value." We'll have some examples below.

Here's a list of information you need to send information via HTTP:

1. The location of the Ifbyphone servers: "secure.ifbyphone.com." While you can use either HTTP or HTTPS to send information to these servers, Ifbyphone recommends HTTPS (and POST). Ifbyphone supports both SSL and TLS security.
2. The URI portion of the URL. This comes in three variations:
 - (a) A couple of these URIs are specific to a particular type of Smart Form. We'll note the specific URI when we discuss that API.
 - (b) Some other Smart Forms use the URI "click_to_xzy.php." In this style you indicate the type of Smart Form by sending the "app" parameter with an appropriate value. For example, to access a "Click-to-Call" service (see Chapter 6), send the value "ctc" for the "app" parameter.
 - (c) Other Smart Forms use the URI "ibp_api.php" and indicate the type of Smart Form by sending the "action" parameter. For example, to access a Conference Smart Form to schedule a conference (see section 8.8), send the value "conference.schedule" for the parameter "action."
3. You must send the parameters required by the service. Typical parameters include:
 - (a) A key: A unique token that identifies you. Chapter 5 discusses keys in detail.

- (b) An ID which identifies which particular Smart Form you wish to access.
- (c) One or more phone numbers are often included when you ask Ifbyphone to make a phone calls.
- (d) "Pass-through" parameters which provide a useful way for you to keep track of data about the history of a particular call, the same way you might use hidden form parameters or cookies in a web-based application.

What are "pass-through" parameters? Some Smart Forms interact with a caller and afterwards send the results back to your servers. If you send pass-through parameters to the Smart Form when you make your web request, the Smart Form will send the pass-through parameters back to your servers along with the results. The parameters "pass through" the Smart Form.

A complete explanation of pass-through parameters can be found in Section 6.3.5.

By way of example, here are two complete URLs, one in each of the two current styles:

```
https://secure.ifbyphone.com/click_to_xyz.php
    ?app=ctc
    &id=27
    &phone_to_call=8475551212
    &key=barbarbar
```

This URL includes four parameters and will access a Click-to-Call Smart Form with Building Block ID 27. The phone number to be called is +1 847 555 1212, and the "key" -- which authenticates that you're allowed to take this action -- is "barbarbar." (In practice, a key is a much longer string of letters and digits.)

A second example:

```
https://secure.ifbyphone.com/ibp_api.php
    ?api_key=foofoofoo
    &action=conference.schedule
    &conference_name=Conference%20%2317
    &scheduled_time=2008-11-18%2014%3A18%3A00%20-0600
    &conference_length=10
    &inboundonly=1
    &invitations=1
    &attendee_list=7735551212%7CJohn%20Smith%7Cjsmith%40example.com%7C
%7C7735552222%7CMary%20Cellphone
```

This second URL includes five parameters, and it schedules a conference call ("conference.schedule"), using the API key of "foofoofoo" to authenticate.

The scheduled start time is given in ISO format: "2008-11-18 14:18:00 -0600," encoded as per the rules for sending data via HTTP.

The conference will last 10 minutes, and the attendee list is given as a phone number followed by a name and optionally an email address -- one attendee, John Smith, has the email address "jsmith@example.com," and again the list has been properly encoded.

4.4.3. Format of Complex Data

When you send complex data to Ifbyphone you need to be aware of some rules.

LISTS. Some Ifbyphone services require lists -- for example, you can use the Phone Mashup API to create a "Find-me" Smart Form on the fly. To send a list, separate each item in the list by a bar (the character "|").

Some parameters accept lists of lists. In that case, you separate the items in each individual list with a bar, and then put the lists together with a double bar (the characters "||") between each list.

For example, the "attendee_list" we showed above in encoded format had a value of:

```
7735551212%7CJohn%20Smith%7Cjsmith%40example.com%7C%7C7735552222%7CMary%20Cellphone
```

Un-encoded, the value looks like this:

```
7735551212|John Smith|jsmith@example.com||7735552222|Mary Cellphone
```

The attendee list consists of two contact lists. The first is for John Smith and contains his name, his phone number, and his email address. The second list, which starts after the double bar, is for Mary and contains just her name and phone number.

PASS-THROUGH DATA. As we mentioned earlier in this section, some web services result in a return web services call from Ifbyphone back to your server, and you can add "pass through" data that will be sent back to your server. In order to do use this, format your data to pass through to Ifbyphone inside the parameter "p_t". We will pass the data back to you exactly as you have passed to us.

For example, if you add

```
&p_t=parameter1|value1||parameter2|value2
```

The URL that Ifbyphone will send will look as follows:

```
&p_t=parameter1|value1||parameter2|value2
```

It will be up to you to properly pull apart the information in the p_t data.

NOTE: Parameters (both names and values) cannot contain any of the reserved characters: ";", "/", "?", ":", "@", "&", "=", "+", ",", and "\$," and of course the

space character. Use URL escape sequences (often known as "percent encoding" or "URL encoding") if you must pass this type of data.

4.4.4. Phone Numbers

By default, Ifbyphone assumes that all phone numbers are US or Canadian numbers. When an API request requires a telephone number, the number should be given as 10 digits only, without the country code ("1"), without the country code designator ("+"), and without any other punctuation such as spaces, dashes, dots, or parentheses.

Developers can make arrangements with Ifbyphone to use non-US numbers with their API requests. Once these arrangements are made, please follow the following rules:

1. US and Canadian numbers, as well as other calls in the North American Numbering Plan (e.g., calls that can be reached with the "1" country code) do not require a prefix.
2. Calls outside the North American Numbering Plan require the US access code, "011," followed by the country code, followed by a single dash ("-"), followed by the phone number.

Please note that you are making an international call. Many countries use a leading zero ("0") to designate an area code, and when calling such a number from outside that country, the leading zero is not dialed.

For example, a phone to call to 7654321 in Jerusalem, which has area code 02 and country code 972, would be sent as

```
&phone_number=01197227654321
```

In other words, the "02" becomes "2."

4.4.5. Receiving Data from Ifbyphone

In many instances, Ifbyphone will respond to your GET/POST with either a web page, a brief status report, or with information formatted in XML. A web page response is for use in a web browser. A brief status report or an XML response contains information from Ifbyphone for you to use in programming: for example, a response code ("the call went through") or administrative information ("here's your list of calls"). The expected output for each type of web request will be discussed later in the Guide as we get into the details of the API.

4.5. Troubleshooting/Debugging

As you work your way through Chapter 6 and beyond, and start to build applications, you will likely find yourself debugging.

One of the easier ways to troubleshoot is to download test PHP and Python programs. Here at Ifbyphone we use a bundle of elementary programs for internal testing, and we've made these programs available online for you to download and use. Read the documentation, edit a few parameters, and then use one of the test programs to send a web request from your command line. If these elementary test programs fail you likely have a fundamental problem -- a firewall blocking your web request, an error in your account number, or something similar. If they work, they provide a good example of the parameters (names and values) you must send and the output to expect. And, of course, the programs are a good way to send *wrong* parameters and values to gain better understanding of how Smart Forms act under error conditions.

To find these test programs, go to the Phone Mashup web site at www.phonemashup.com, and then navigate until you find "API Testing Scripts". If you download these you'll find a bundle of very short programs that exercise just about all of the API. See Chapter 9 for details of how these programs work.

For example, if you'd like to see if you can use Click-to-Call, use the `ctc.py` (Python) program or `ctc.php` (PHP) program. After you correctly configure the scripts with your own telephone numbers and keys, execute this program by typing "`python ctc.py`" or "`php ctc.php`" and you will see Ifbyphone's response on your terminal. If the file is executed correctly, Ifbyphone will initiate a phone call to the number you provide.

The other method of troubleshooting is to make certain that your account works when you use the web site instead of the API to create, modify, and schedule Smart Forms. Sometimes a few minutes at the web site, imitating the actions of the API, will reveal more clearly the correct methods to use Smart Forms.

If you need additional help, you can contact Ifbyphone technical support through the web site.

5. Security and Phone Mashups

A good question to ask at this point is, "what about security?" Many of Ifbyphone's API web services cost real money -- in particular, the ones that make telephone calls. Not to mention, of course, that you don't want someone else accessing or modifying your Smart Form information via administrative web services. And Ifbyphone doesn't want you to be annoyed by random phone calls from pranksters. For all these reasons, Ifbyphone includes security measures in the Phone Mashup API.

Ifbyphone provides different security measures based on how the web service works and what it does, as well as the level of monetary loss that might be imposed if someone attempts to misuse the service. This is the same security calculation that lets you take \$100 out of your account anonymously at an ATM with only a PIN number but requires your physical signature and identification on a \$5,000 check: different security measures for different amounts of risk.

The Phone Mashup API web service requests use "keys" to authorize calls. A key consists of a string of alphanumeric digits; one of the keys is "public" and can be transmitted on the Internet in the clear, but the others should be treated much more stringently.

Another security measure is "registered numbers." Registered telephone numbers are telephone numbers you have designated that have special privileges, such as the ability to receive various calls via Click-to Smart Forms.

Finally, some services require that you include your account password in the request. Send this data in secure mode only -- use POST and HTTPS.

The usual rules of security apply: "need to know." Unless you distribute something that must include a key, such as a link to a Click-to-Call or similar Smart Form, you should not expose any key information on the web -- and certainly you should never expose credentials, such as keys and account numbers. Keep these data off web pages even when you integrate Ifbyphone and the web: use programming languages such as PHP, ASP, JSP, Rails, Python, etc., to have your server interact directly with Ifbyphone's servers. Your data can travel directly server-to-server via HTTPS and POST from your server to Ifbyphone's server and vice-versa; your users will receive services but never see the data.

5.1. Keys: Public, "API," and Private

We'll start with a review of the three types of keys you use with API web service requests. These keys can be found by logging into the Ifbyphone web site and choosing Tools -> Building Block IDs. Ifbyphone generates these keys, which consist of long strings of alphanumeric characters.

WARNING: Lost Keys. If you believe your keys have been compromised, contact Ifbyphone support immediately. Remember: You are responsible for the security of your keys and for all transactions performed with your keys.

5.1.1. Use of Keys

All API calls now require a key. Although some of the older APIs did not require a key in the past, and some may still work without a key, in the near future a key will always be required -- so always include the appropriate key

5.1.2. Lower Security: Use of "Public Key"

The "Public key" does not have to be kept secret -- in fact, it's intended for distribution in email and on web pages just as ordinary links are. All the "Click-to" Smart Form web services, as described in Chapter 6, use the Public key.

One goal of the Public key is identify your account without putting your account number or telephone number into a link you send by email: the key identifies you to Ifbyphone so that Ifbyphone can match web service calls to your Smart Forms. By using the Public key, we make it impossible for a potential hijacker to learn anything about your account name or telephone number from reading the link.

On the other hand, in some cases the Public key is used in conjunction with other credentials to give access to important services, such as creating Smart Forms. Be careful: in these cases, you must guard those additional credentials.

The reason we can use this lower level of security -- keys that can be freely distributed -- is because many web requests for "Click-to" Smart Forms result in a phone call between a "registered" telephone number known to Ifbyphone in advance (e.g., your office number) and the person who uses the service. Therefore there's no real point to fraudulent use of the link. For example, if you create a link to a Click-to-Call Smart Form to connect users to your office telephone number, and you distribute that link in email and someone uses that link, all that happens is that the user reaches your office telephone number -- they can't use that link to make arbitrary telephone calls to anyone they please.

Here is a detailed example, complete with a code sample. Let's say you've created a "Click-to-Call" Smart Form that lets someone call your sales team telephone number, and now you wish to distribute a link in your email so people can call your sales team. When you check the Ifbyphone web site's "Building Block IDs" page, you find that your Smart Form's ID is 27, and that "Your Public Key" is "foofoofoo." Using the information in Chapter 6, you can create a link to distribute in your email:

```
https://secure.ifbyphone.com/click_to_xyz.php
    ?app=ctc
    &click_id=27
    &phone_to_call=8475551212
    &id=8475550000
```

`&key=foofoofoo`

Notice that your account information -- the account ID, your account password -- doesn't appear anywhere. The key lets Ifbyphone know that this link is associated with a valid account, find the Smart Form with Building Block ID 27 for that account, and then make an outbound call to connect your sales team number at +1 847 555 1212 with the person who clicked on the link. (See below for information about Registered Numbers.)

At some point in the future, Ifbyphone may introduce additional security measures. We will keep our developer community informed.

5.1.3. Higher Security: "API Key"

Some web service requests require high security. For example, assume that you're managing a conference through the API (see Chapter 8). You wouldn't want just anyone to be able to add or delete conferees, set up a conference on your account, or do very much of anything else with this service. These Phone Mashup API requests and others like them require security to prevent fraud, hacking, and mischief.

In Chapter 8, you will find a list of Phone Mashup API requests that require an "API key." When you send these keys and accompanying information, use POST and HTTPS to prevent disclosure of your API key (instead of the more usual GET and HTTP). Furthermore, the API key should never appear in the text of a web page visible to your clients; the key should only used for server-side programming.

5.1.4. Other Security: Private Key

At present the "Private Key" is not used by the API. In the future you will be able to use the Private key to "sign" your transactions with Ifbyphone to provide additional security.

The Private key should never be visible to a user; in fact, the Private key should never be transmitted anywhere, even to Ifbyphone. The Private key will only be used to "sign" a transaction.

5.2. Account Number and Password

Some services require that you include your account number and password; for example, when making a Click-to-Call connection between two numbers when neither is registered. You should only send your password through a secure service (HTTPS) and POST in order to keep the information encrypted and invisible to casual inspection.

You can find your account number on the web site at your account's Home page under the name "Account ID." The account password is the PIN you use to log into the web site, usually a four digit number. In many cases, you can provide your API

key as an alternative to passing in the Account number and Password. Please follow the guidelines when using an API key.

5.3. Registered Numbers

A "Registered Number" is a telephone number with special privileges. You manage your list of Registered Numbers from your account Home page via Tools -> Registered Numbers. For the most part, Registered Numbers are the ones that can receive telephone calls when you use the API with just the Public key.

If you distribute a link on the Internet, such as one that links to a Click-to-Call Smart Form, the link will contain the Public key and the telephone number that will receive the call. Why can't someone simply substitute a different destination number into that link and use it -- in other words, steal telephone service to make calls to any destination, not just the one you put into the link? The answer is that the telephone number in the link is registered; if an attacker substitutes an unregistered telephone number, Ifbyphone will refuse to make the call unless the link includes additional authentication -- and if you've guarded your credentials correctly, the attacker won't be able to authenticate.

5.4. Summary of Precautions

The simple rules are:

- Only your Public key should ever be visible to anyone other than yourself. That's what the Public key is designed for: situations when your key must be public.
- Always use HTTPS and POST to send parameters. The only exception is for "Click-to" links that you distribute via email and on web pages -- these can use GET and HTTP.
- Never reveal your API key, your Private key, or your account and password information. If you want to make a web request that requires these items and integrate it with a web page, make certain you use server-side programming only, which will keep the secrets hidden in your server. These secrets should never be sent inside a web page form or otherwise revealed to a user.

6. "Click-to" Smart Forms API

This chapter provides details on the "Click-to" Smart Forms. These Smart Forms provide you with access to a wide range of Ifbyphone's services, and can often be initiated by you, or by your customer, by clicking on a single link.

If you are not familiar with what a specific Smart Form does, consult the documentation on the web site, which provides detailed information.

6.1. Settings

To use these web service requests, you need to know:

1. Server name: secure.ifbyphone.com. You can use GET or POST, and secure or non-secure (HTTPS or HTTP) service.
2. URI: /click_to_xyz.php
3. The "app" parameter. For example, "Click-to-Find" web service requests must set the "app" parameter to value "CTF."
4. Your Public key. See Chapter 5 for a discussion of keys. You can also pass your API Key if you want to omit passing an Acct ID and Password (if applicable). Note that passing your API Key should only be done on hidden & secured pages.
5. The Building Block ID of the particular Smart Form you want to use. See Chapter 4.
6. Parameters: Each service has a list of parameters; see the tables below.

6.2. Responses

Responses from a "Click-to" Smart Form are generally strings of text:

- "Call Connected." This is received immediately after the first leg of the call connects, *regardless* of whether the second call leg connects.
- Error messages include:
 - "Missing Application Specification" if you forget to include the "app" parameter.
 - "The link you clicked to reach here was invalid" followed by more text if you leave off any of the required parameters or if they have incorrect values.

- "Account needs to be verified" followed by more text if you make an error in your key.

In some instances, you may receive XML or HTML in response. For example, if you make a web service request to the wrong URL (e.g., you neglect to send it to `click_to_xyz.php`), you will get HTML in response because Ifbyphone's servers assume you tried to reach a web page.

6.3. Services

Each of the following tables include the name of the service and the list of parameters for that service, and is generally followed by an example.

6.3.1. Click-to-Call

Application	Parameter	Value/Notes
Click-to-Call	app	CTC
	type	1 = Call "phone_to_call" value first. 2 = Call "id" first.
	id	A registered number or the main number of the account.
	phone_to_call	Telephone number to call.
	key	Public key.
	acct	If the value of id (above) is not a registered number, include the account ID.
	pwd	If the value of id (above) is not a registered number, include the account's password.

Here's a sample click-to-call between a registered number and another number:

```
https://secure.ifbyphone.com/click_to_xyz.php
?app=ctc
&id=8475550000
&phone_to_call=8475551212
&type=1
&key=foofoofoo
```

The server will call 847.555.1212, and then call the (registered) number 847.555.0000.

In this next example, we make a call between two unregistered numbers:

```
https://secure.ifbyphone.com/click_to_xyz.php
?app=ctc
&id=8475559999
&phone_to_call=8475551212
&type=1
&key=fooffoofoo
&acct=1234
&pwd=4321
```

This link uses account ID number 1234, with password (login PIN) 4321. The system will call 847.555.1212 and then call 847.555.9999.

6.3.2. Click-to-Find Me

Click-to-Find Me	app	CTF
	key	Public key.
	phone_to_call	Telephone number to call.
	findme_id	The ID of the Find Me Smart Form to use.

Sample code for Click-to-Find Me Smart Form:

```
https://secure.ifbyphone.com/click_to_xyz.php
?app=ctf
&findme_id=261
&phone_to_call=8475551212
&key=fooffoofoo
```

This link will cause Ifbyphone to use the key to both authenticate the request and determine which account information to access; make a call to telephone number +1.847.555.1212; and use the account holder's Find Me Smart Object number 261 in an attempt to connect that call.

Click-to-Find Me List Attempts to connect "phone_to_call" with one of the numbers in "list"	app	CTFL
	list	List of of phone numbers to which we attempt to connect caller.
	screen_prompt	Message to play to caller. See Note, below.
	phone_to_call	Telephone number to call (the "caller").
	key	Public key.
	acct	Account ID.
	pwd	Account password.
<i>See next two tables for optional additional parameters</i>		

NOTE: The parameter "screen_prompt" lets you provide an announcement that will be read to the caller using text-to-speech. By default, the Ifbyphone server will play a prompt asking for the caller to record his name; then it will play a tone and record. If you use screen_prompt to provide your own announcement, you should bear in mind that the Ifbyphone sever will play a beep and start recording, and that if the caller does not leave a name, the callee will get a "caller did not give name" message. You can change this default behavior by using the "use_screen" parameter; see below.

Unlike the example for the Click-to-Find Me web request, there's no pre-defined Smart Form for this particular set of actions -- no "id" parameter. Instead of configuring a list of numbers in advance inside a Click-to-Find Me Smart Form, you supply a list of telephone numbers along with the web request using the parameter "list."

Sample code for an ordinary Click-to-Find Me List web request:

```
https://secure.ifbyphone.com/click_to_xyz.php
?app=ctfl
&phone_to_call=8475551212
&screen_prompt=At+the+tone%2C+please+say+your+name.
&list=7735551111|3125559999
&key=foofoofoo
&acct=1234
&pwd=4321
```

This will result in a call to telephone number +1.847.555.1212. The Smart Form will use text-to-speech to speak the announcement given in "screen_prompt" and then attempt to connect that call to one of the numbers given in the parameter "list."

You can use the parameter "nextaction" to customize how the Click-to-Find Me List works. Some of these customizations allow you to link to other Click-to Smart Forms, such as voice mail or a virtual receptionist. (As we stressed back in Chapter 3, this ability to choose the next Smart Form gives you quite a bit of flexibility in a dynamic programming environment.)

Customize Click-to-Find Me List Web Request Values of "nextaction" and "nextactionitem" parameters		
Custom Action	nextaction Value	nextactionitem Value (if required)
If unable to connect, tell user no one is available and disconnect (default).	1	(not required)
If unable to connect, transfer to Voice Mail.	2	Voice Mail ID
Automatically accept the call on the last number tried.	3	(not required)
If unable to connect, route to a Virtual Receptionist.	4	Virtual Receptionist ID
If unable to connect, route to a Click-to-Find Me Smart Form.	5	Find Me ID
If unable to connect, route to a SurVo Smart Form.	6	SurVo ID
Automatically accept call on last number tried without introduction.	7	(not required)

Table 6.1: Find Me Smart Form Extra Parameters

Here's an example:

```
https://secure.ifbyphone.com/click_to_xyz.php
?app=ctfl
&phone_to_call=8475551212
&screen_prompt=At+the+tone%2C+please+say+your+name.
&list=773551212|3125551212
&key=foofoofoo
&pwd=barbarbar
&nextaction=2
&nextactionitem=277
```

This is essentially the same as the previous web request, but we've customized it. If there are no answers, the caller is routed to a Voice Mail Smart Form, the one with Building Block ID 277.

You can also add the following extra parameters to customize your Click-to-Find Me List web service request.

Additional Customized Find Me List Parameters		
Description	Parameter	Valid Values
Specify type of Find Me.	usr_findme_type	1 – Individual Find Me (default) 2 – Customer Service Hunt Group
Number of times to go through the call list before triggering last action.	loop_count	1 – 5 (default is 1)
Specify if numbers should be randomized.	randomize	0 – No (default) 1 – Yes
Record the call once connected.	record	0 – No (default) 1 – Yes
Option to screen caller for information.	use_screen	0 – No 1 – Yes (default)
Optional TTS screening message (if none provided then system	screen_prompt	String length > 0

will use default messaging).		
Force operator to use keypad to accept call.	dtmf_only	0 – No (default) 1 - Yes
Optional audio file to play while caller is waiting for operator to accept call. If none provided, default audio is used.	holdmusic	Name of audio file which must be in the users "holdme" directory.
Custom hold audio looping settings.	holdrepeat	0 – Only once per Find Me session 1 – Only once between each Find Me attempt 2 – Repeat continuously between each Find Me attempt (default)
Number of seconds each number on Find Me list gets before timing out and moving to next number (or end action).	timeout	10 – 60 (default 30)
Message whispered to operator. Only valid if not screening caller.	whisper_phrase	String length > 0

Example Click-to-Find Me List with a parameter from the table above:

```
https://secure.ifbyphone.com/click_to_xyz.php
?app=ctfl
&phone_to_call=8475551212
&screen_prompt=At+the+tone%2C+please+say+your+name.
&list=773551212|3125551212
&key=foofoofoo
&pwd=barbarbar
&nextaction=2
&nextactionitem=277
&timeout=10
```

The results are a more customized version of the previous results. The "timeout" parameter causes Ifbyphone to wait for only 10 seconds for the callee to pick up a phone; if he does not, the phone tries the next number in the list.

6.3.3. Click-to-Virtual Receptionist

Click-to-Virtual Receptionist	app	CTVR
	key	Public key.
	menu_id	The ID of the selected Virtual Receptionist menu.
	phone_to_call	Telephone number to call.

Example:

```
https://secure.ifbyphone.com/click_to_xyz.php
  ?app=ctvr
  &phone_to_call=8475551212
  &menu_id=276
  &key=foofoofoo
```

Ifbyphone's servers will place a call to the +1.847.555.1212, and connect it to the account's Virtual Receptionist Smart Form that has Building Block ID 276.

6.3.4. Click-to-Voice Mail

Click-to-Voice Mail	app	CTVM
	key	Public key.
	vmail_box_id	The ID for the voice mailbox.
	phone_to_call	Telephone number to call.

Example:

```
https://secure.ifbyphone.com/click_to_xyz.php
  ?app=ctvm
  &phone_to_call=8475551212
  &vmail_box_id=426
  &key=foofoofoo
```

Ifbyphone's servers will place a call to +1.847.555.1212 and connect it to the account's Click-to Voice Mail Smart Form that has Building Block ID 426.

6.3.5. Click-to-SurVo

The SurVo Smart Form deserves a few extra words of introduction. This is the most flexible and powerful of all the Smart Forms, because it's designed to play recordings and text-to-speech, accept input in the form of speech or DTMF tones,

make recordings, send you results, and transfer control -- at your command -- to other Smart Forms. The real name for this Smart Form ought to be "Question and Answer" Smart Form, or maybe "User Interaction" Smart Form. We'll stick with the name SurVo in the meantime.

The SurVo Smart Form can make outbound telephone calls to a list of telephone numbers. If you need to call a thousand people and ask them some questions, this is probably the Smart Form to use.

To understand how to use the SurVo Smart Form, read the SurVo documentation. This API lets you *use* a SurVo, but the API does not let you *configure* or *create* the SurVo. That's done on the web site, and we'll discuss some configuration topics below.

Click-to-SurVo	app	CTS
	acct	Account ID.
	key	Public key.
	survo_id	ID of the SurVo.
	user_parameters	(Optional) List of parameters. See section "SurVo User Parameter Substitution," below.
	p_t	(Optional) Pass-through data.
	phone_to_call	Telephone number to call. Not valid if "scheduleonly" flag is set.
	scheduleonly	(Optional) Flag - this parameter may consist of a name without any value. No call placed. See section "Schedule Voice Broadcast" below.

NOTE: If scheduleonly flag is set, see next table for additional parameters.

Example:

```
https://secure.ifbyphone.com/click_to_xyz.php
  ?app=cts
  &acct=1234
  &phone_to_call=8475551212
  &survo_id=890
  &key=foofoofoo
```

This example will call 847.555.1212 and start SurVo Smart Form ID number 890.

6.3.5.1. Schedule Voice Broadcast

The Click-to-SurVo web request can be used to schedule, in advance, a series of telephone calls.

To set the "scheduleonly" parameter, add the line

```
&scheduleonly
```

or, if for some reason it's easier, use

```
&scheduleonly=some_value
```

If scheduleonly is set, you may use parameters from this next table. Remember that the "phone_to_call" parameter of the previous table is not valid when you use scheduleonly.

The return message for a successful web request will be in the form:

```
1 Voice Broadcast(s) scheduled. Broadcast ID: 34059
```

The "broadcast ID" in the return message can be used by other API calls to modify the schedule, as shown later in this Guide.

Extra Parameters for Click-to-SurVo Schedule	
phone	The phone number(s) scheduled to receive the SurVo Broadcast call. May be a single phone number, a " " delimited list of phone numbers, or a fully qualified URL to a web-accessible and valid CSV file. See Notes below. If you use "phone," do not use "phone_to_call."
sdate	The official Voice Broadcast starting date/time in the format YYYY-MM-DD HH:MM (24 hour format).
edate	The official Voice Broadcast ending date/time in the format YYYY-MM-DD HH:MM (24 hour format). Must be chronologically later than "sdate". Note that this parameter is ignored if the "type" parameter is set to "2" (As fast as possible).
tz	(Optional) The time zone all specified date/time values are considered to be in. (Valid values are "Eastern", "Central", "Mountain", "Pacific", "Alaska", "Hawaii"; default is "Eastern").
dstime	(Optional) The "Daily Call Range" starting time when Voice Broadcast calls can be made (Valid values meet the HH:MM 24 hour format).
detime	(Optional) The "Daily Call Range" ending time

	when Voice Broadcast calls can be made (valid values meet the HH:MM 24 hour format and come later in the day than the value for "dstime").
type	The scheduling algorithm to use when scheduling calls for this Voice Broadcast. (Valid values are 1 for "Spread calls evenly" or 2 for "As fast as possible.")
attempts	(Optional) Maximum number of retry attempts for each phone number. (Valid values are 1,2,3; default is 1 – meaning no retries.)
retry	(Optional) Number of minutes between retry attempts on a phone number. (Valid values are 5,10,15,30,60,90,120; default is 5.)
simul	(Optional) Number of simultaneous calls to be placed every freq minutes. (Valid values are 1,2,3; default is 1.)
cid	(Optional) Caller ID value that shows up to recipients of the Voice Broadcast. (Valid values are any phone number that matches one of the account's toll free numbers, or is listed in the accounts Registered Phone number list; default is the account's primary toll free number.)
desc	(Optional) Identifier for the Voice Broadcast used on the web site's account management page. (Valid values contain alphanumeric characters and spaces; default is "My SurVo Broadcast"; maximum length is 32 characters.)

Here's an example:

```
https://secure.ifbyphone.com/click_to_xyz.php
?app=cts
&acct=1234
&scheduleonly
&survo_id=890
&key=foofoofoo
&type=1
&sdate=2008-10-30+08%3A48
&edate=2008-10-30+08%3A58
&tz=Central
&phone=7735551212%7C8475551212
&dstime=00%3A00
&detime=23%3A59
```

This URL will schedule a call to two phone numbers and connect those callees to the Click-to-SurVo Smart Form with Building Block ID 890 defined in account 1234. The calls will take place on Oct 30, 2008, starting at 08:48 and ending at 08:58, and the call will be spread out evenly in that time slot. The time zone is set to Central time.

Calls are limited to be between the hours of 00:00 (midnight) and 23:59 (almost midnight), i.e., all day long.

To use a URL with a list of numbers, use (for example)

```
&phone=http%3A%2F%2Fwww.example.com%2Ffiles%2Fnumberstocall.csv
```

In this example, the Smart Form will retrieve the CSV file from "http://www.example.com/files/numberstocall.csv" and schedule calls to the numbers given in the file.

NOTES: If the value of phone is a CSV file, the file must end with ".csv." In addition, the CSV file must have at least one column with the heading "numbers."

For example, a CSV file can contain one column of names and another of telephone numbers:

```
"name", "number"  
"John Q. Public", 7735551212  
"Mary Public", 8475551212
```

6.3.5.2. User Parameters: How to Create Customized Announcements

The parameter "user_parameters," which we mentioned in the Click-to-SurVo table above, lets you perform parameter substitution in a properly configured SurVo. What this means is that if a SurVo Smart Form has text which is read out over the phone using text-to-speech, and that text includes a "user parameter," then the Smart Form will substitute the value you supply for that user parameter and speak that text.

As an example, let's say you have created a SurVo Smart Form for an outbound set of calls and you want to personalize the opening announcement for each person you call. You can create three user parameters and incorporate them into the text:

```
Hello {salutation} {first_name} {last_name}, this is a message from  
the Doctor.
```

The parameters you created are "{salutation}," "{first_name}," and "{last_name}."

To use these parameters, you will need to send the values for these parameters to the SurVo when you make your web request.

Let's say we want to personalize this greeting for "Ms. Donna Noble." We want to send these values:

```
salutation: Ms.  
first_name: Donna  
last_name: Noble
```

This is the list of "user parameters," and we send this list to the API by sending it as the value for the "user_parameters." How do we send a list? As we noted in Chapter 4, to send a list of a parameters send the name and value of a parameter with a "|" between the name and the value and a "||" between each parameter:

```
salutation|Ms.||first_name|Donna||last_name|Noble
```

Before you send the list, you must encode it for transmission (in this case, the "|" separators are sent as %7C because of URL encoding rules). After encoding, we send the list as the value of the user_parameter, and the URL above would have an additional line:

```
&user_parameters=salutation%7CMs.%7C%7Cfirst_name%7CDonna%7C%7Clast_name%7CNoble
```

The SurVo Smart Form will make substitutions and speak this text:

```
Hello Ms. Donna Noble, this is a message from the Doctor.
```

See the "SurVo Advanced Guide" for more information on parameter substitution and how to set up a SurVo Smart Form to use parameter substitution.

6.3.5.3. NetGet: How to Receive Data from a SurVo Smart Form

As we said earlier, another good name for SurVo would be "Questions and Answers" Smart Form. You can use this form to ask questions and get answers (and recordings). Of course this begs the question of how you get that information back from Ifbyphone.

The answer is that you have several methods to get information back. You can store responses on the Ifbyphone web site in a database and retrieve it through the web site. You can ask for the responses to be sent to you via email, including recordings. Or you can ask for the responses to be sent to you immediately over the Internet, via HTTP(S). We'll spend most of our time here discussing this last scenario.

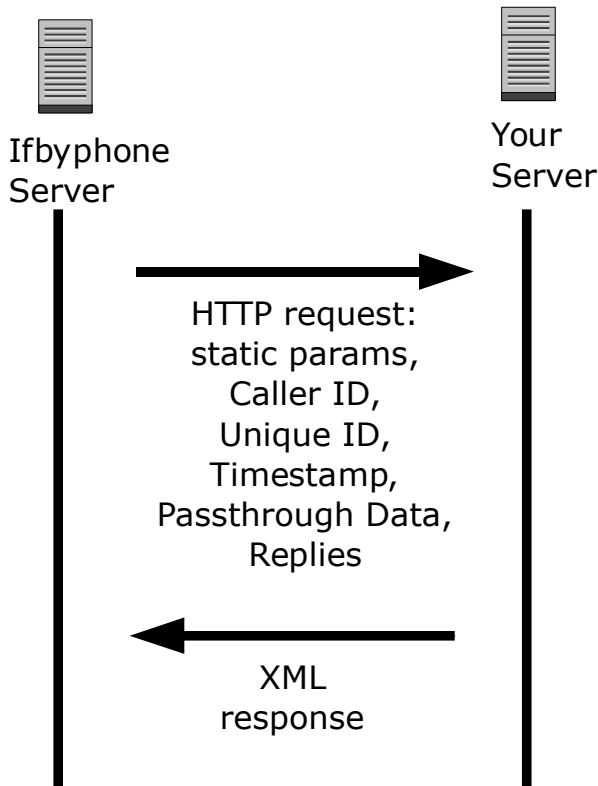
To understand how this works, let's do a quick review of how HTTP-based interactions work for server-side programming:

1. The client issues an HTTP request.
2. The server responds to the HTTP request with a status code and, optionally, some data.

Throughout this chapter, we've shown how to send a web request and the typical response from each Smart Form. Those responses are the immediate responses from the Smart Forms.

SurVo Smart Forms, however, are different because you can configure them to have a second response: a client request from the Smart Form to your server. To send you data, the Smart Form (running on Ifbyphone's servers) sends you an HTTP(S) request, to which you must send a response.

You can configure the SurVo to send quite a bit of information to your server in that HTTP(S) request. Here's a diagram:



Drawing 6.1: NetGet -- HTTP Request from SurVo Smart Form and Your Response

As you can see, the "replies" -- the things that your user said -- are just one part of the overall data.

6.3.5.3.a. Configuration

After you create a SurVo Smart Form online at the web site (Tools -> SurVo), you have a drop-down box that offers you a "Post SurVo Action" to handle the data that the SurVo collects. This drop down box includes several options that begin with "Net Integration." If you choose any of these options, you will will get a screen that looks like this:

Survo NetGet

Configure the URL that will be used to submit SurVo results to your website/server and what action it should take with the results.

NetGet Action:

Submit Type:

Domain:

Page:

Static Parameters:

SurVo Generated Parameters:

Parameter 1: =

Parameter 2: =

Parameter 3: =

Parameter 4: =

To remove a parameter just delete its contents or select the blank option from the drop down and hit update.
 To add another parameter after current ones are filled in, hit Save/Update and a new empty parameter field will be added to the list.

Current URL: `http://www.example.com/killer_app.php?uid=unique_id&time=timestamp&mydata=p_t`

Click Save/Update to see updated URL

Illustration6.1: Configure a SurVo's NetGet Options

In the next sections we'll discuss the options presented here, how they relate to the information that the SurVo Smart Form sends you, and how they affect the overall flow of the application.

6.3.5.3.b. Action

This option describes what the SurVo expects to do after it makes its HTTP(S) request:

Action Name	Description
Retrieve HTML/TEXT and read to user	In response to its HTTP(S) request, the SurVo expects to receive HTML or text, which it will then read to the user using text-to-speech.
Retrieve HTML/TEXT and read to user	In response to its HTTP(S) request, the SurVo expects to receive correctly formatted RSS, which it will then read to the user using text-to-speech. If selected, the "Submit Type" must be

	GET. See Note below.
Transmit data and disconnect	The SurVo doesn't expect any data in response to its HTTP(S) request.
Transmit data and get next action	In response to its HTTP(S) request, the SurVo expects to receive data in XML format. The data will include the ID of the next SurVo Smart Form to execute and input parameters for that SurVo.

NOTES: SurVo Smart Forms accept RSS formats RSS 2.0, RSS 1.0, and Atom X.x.

POST will not work if the page is redirected, that is, if your server redirects the Ifbyphone server.

The following static parameters should not be used in a NetGet URL (they are reserved for system use: A## (where # is a number (e.g. A12)), callerid, unique_id, sid and p_t.

6.3.5.3.c. Submit Type

You may send data via GET or POST. As we've noted many times, POST in combination with HTTPS is the best choice for data security.

You must use GET:

- When you retrieve RSS. RSS servers do not accept POST.
- When the "Domain" will re-direct the SurVos HTTP(S) request.

6.3.5.3.d. Domain

The domain of the server (the URL) of that should receive the SurVo HTTP(S) request. E.g.,

`http://applications.example.com`

6.3.5.3.e. Page

The URI portion of the URL, i.e., the page address. For example,

`phonemashups/great/killer_app.php`

6.3.5.3.f. Static Parameters

These are entirely static parameters, that is, they do not depend on any actions taken by the SurVo. They're handy when you have some data or token you'd like to send along.

For example, if your user will interact with six separate SurVo Smart Forms and you want a plain English token to let you know which one the user just used, include a static parameter such as

```
step=choose_color
```

and that static parameter will be sent along with the other data you request. If you would like to send more than one static parameter, use the "&" between each pair:

```
name1=value1&name2=value2&name3=value3
```

6.3.5.3.g. "SurVo-Generated Parameters": The Dynamic Parameters

This is the heart of the SurVo Smart Form. Dynamic parameters include the answers that users give to your questions. You can also gather important information about the call itself, such as when it happened and the caller ID of the caller.

To use the dynamic parameters, fill out the form. Select a name for the parameter -- your server will receive the parameter, so make it something that's sensible for you -- and then select the value for the parameter from the drop-down list. The values that will appear on the drop-down list are shown in Table 6.1.

Note that when you first get to this page, you only have room for one dynamic parameter. That's a quirk of the web site's interface. Just hit "save changes" and a new line will open up for the next dynamic parameter, as seen in Illustration 6.1.

SurVo Generated Parameter: Value Name	Description
A1, A2, A3,...	User's answer to the SurVos question 1, question 2, question 3, etc.
SurVo ID	Building Block ID of this SurVo.
Unique ID	Unique ID assigned by Smart Form to each interaction a SurVo has with a user.
Caller ID	Caller ID, if known, of the user.
Timestamp	Time stamp of start of user interaction with SurVo.
Number dialed	The number the caller dialed (assuming an inbound call) to reach this SurVo. Also known as "DNIS."
Pass-through data	Data you passed to the SurVo Smart Form and asked to be passed back to you.

Table 6.2: SurVo Generated Parameters

For example, in Illustration 6.1 we have a parameter called "okay," and we assign as its value the answer the user gave to question number 1 of the SurVo, "A1." (The questions are not shown because they're on a different web page, but the question is "Is that okay with you?" and possible answers are "yes" and "no.") The SurVo Smart Form will send data to us, and the parameter "okay" will have a value of "yes" or "no" depending on the response of the caller. We'll also get the unique ID of the call ("uid"), the time and date it happened ("time"), and any pass-through data we sent ("mydata").

The web site's form shows the schematic of the output to expect below the list of dynamic parameters, based on the inputs on the form:

```
http://www.example.com/killer_app.php
?app_name=irving
&uid=unique_id
&time=timestamp
&mydata=p_t
&okay=A1
```

This is just a schematic, not the actual output. In the actual output, the dynamic parameters will be replaced with the actual values from the call -- "A1" will be replaced with a "yes" or "no," "timestamp" with the actual time, etc.

6.3.5.4. Response to the SurVo NetGet from Your Server

As we stated just above, the "NetGet Action" can have one of four values. You can set the SurVo to not expect any data in response to its HTTP(S) request to your server -- easy enough. You can set it to expect HTML or text to read, or an RSS feed to read. Again, easy enough.

If you set it to "Transmit data and expect next action," the SurVo expects to receive an XML-formatted response.

The response will be formatted as follows:

```
<action>
  <app>survo</app>
  <parameters>
    <id>75</id>
    <user_parameters>
      <param1>value1</param1>
      <param2>value2</param2>
    </user_parameters>
    <p_t>mypassthrough data</p_t>
  </parameters>
</action>
```

An "action" element is required, at minimum, or the SurVo will fail to complete.

The "app" element informs the SurVo Smart Form what to do next. Depending on the value of the "app" element, the "parameters" element may contain one or more additional elements to fully define what happens next. This next table gives the possible values of the "app" parameter; the names of any elements in the "parameters" element that are associated with that value of "app," and what it all means.

"app" value	Parameters: name(s)	What it means
survo	id (Required)	SurVo Smart Form will transition call to a different SurVo Smart Form with given "id." See below for other data you can send.
findme	id (Either id or phone_list must be specified)	If "id" is given, SurVo Smart Form will transition to a Find Me Smart Form with given "id."
	nextaction (optional)	Findme configuration option. See Table 6.1 and explanation in section 6.3.2.
	nextactionitem (depends on nextaction)	Findme configuration option. See Table 6.1 and explanation in section 6.3.2.

	phone_list (Either id or phone_list must be specified)	A " " delimited list of phone numbers to try in the order specified.
vr	id	SurVo Smart Form will transition call to a Virtual Receptionist Smart Form with given "id."
voicemail	id	SurVo Smart Form will transition call to a Voicemail Smart Form with given "id."
hangup		Hangs up the call.

If the next action is a SurVo, you can use the parameters element to send the next SurVo any user parameters. In the example of "action" we saw above, the SurVo with ID 75 will receive user parameters of param1 with value1 and param2 with value2. These user parameters can be substituted into the text of outgoing announcements just as explained above. The "p_t" element of the parameters element will contain any pass-through data, as previously discussed.

6.3.5.5. Pseudo-code for the Destination URL

Here's a bit of sample code for how you would receive information from a NetGet and format a correct "action" response. The scenario: the SurVo asks for an account number, and this pseudo-code validates the account number.

- If the account number received from the SurVo via NetGet is correct, have the SurVo transfer control to SurVo with ID 75. Keep the account number with the data by using pass-through parameters.
- If the account number is not correct, check the number of attempts so far -- that number will be in a pass-through parameter send by the SurVo. If we haven't had too many failed attempts, respond with a request that SurVo 76 be the next SurVo (an error-handling SurVo). Keep track of the number of failed attempts by using a pass through parameter.
- If you detect too many failures, ask for control to pass to SurVo with ID 77.

```
//File: phone_interface.php
//Contains the logic for a customer validating an account number,
//triggered by various SurVo NetGets.

// Retrieve the values passed in the URL
$step = $_REQUEST['step'];
$attempt_number = $_REQUEST['attempt_number'];
$account_number = $_REQUEST['acct_number'];

// Take action based on the values.
switch($step){
  case 'authorize': // Validate the account number passed.
    $account_status = valid_account_number($account_number);
    // Test status returned by validation routine
    if ($account_status == 'valid') {
      // Build next action XML structure to pass designated SurVo id
```

```

        // for continued processing
echo "<action>
  <app>survo</app>
  <parameters>
    <id>75</id>
    <p_t>acct|$account_number</p_t>
  </parameters>
</action>";
} else {
// Build next action XML structure to pass designated SurVo id for invalid accounts
// First check how many failed attempts
if ($attempt_number <= 2) {
    // Less than 2 attempts, request another attempt
    //at entering the account id.
$attempt_number++;
echo "<action>
  <app>survo</app>
  <parameters>
    <id>76</id>
    <user_parameters>
      <acct_number>$account_number </acct_number>
    </user_parameters>
    <p_t>attempt|$attempt_number</p_t>
  </parameters>
</action>";
} else {
// Too many failed attempts.
    // Return next action XML structure to execute
    // SurVo which transfers the call to Customer Service

    echo "<action>
  <app>survo</app>
  <parameters>
    <id>77</id>
  </parameters>
</action>";
}
}
break;
default:
;
} // end switch
exit;

```

Note the power of the SurVo and NetGet. The flow of the application is not rigid: your server response dictates which SurVo Smart Form (or other Smart Form) has control of the call next, and you can send parameters to that Smart Form. This dynamic programming under your server's control is the heart of many interesting phone mashups.

7. Reminder and Schedule

These web requests use a different URI than the Click-to-Call web requests.

7.1. Reminder

The URI of a Reminder web request is `vome_widget_add.php`.

Application	Parameter	Value/Notes
Reminder	user_phone	Phone number to call. This must be a registered phone number.
	acct	Account number.
	pin	Personal Identification Number for account.
	key	Public key.
	message_text	Message to be spoken to the recipient of the phone call.
	recipient_name	Name to be spoken to personalize the message.
	caller_name	Name to be spoken to identify the originator of the message.
	M	The month the call should be placed, given as a number between 1 and 12.
	D	The day the call should be placed, given as a number 1-31.
	Y	The year the call should be placed.
	H	The hour of the day that the call should be placed, given as 1-12.
	N	The minute of the hour H that the call should be placed, given as 0-59.
	A	Meridian, values "AM" or "PM".
timezone	Timezone for the time of the call; values: Eastern, Central, Mountain, Pacific, Alaska, Hawaii.	

NOTE: Unlike other web requests, the names of the parameters for this particular web request are case-sensitive. Do not use "PIN"; use "pin."

Example:

```
https://secure.ifbyphone.com/vome_widget_add.php
?&acct=1234
&pin=4321
&scheduleonly
&survo_id=890
&key=foofoofoo
&Y=2009
&M=1
&D=7
&H=8
&N=56
&A=AM
&timezone=Central
&user_phone=7735550000
&caller_name=The+Doctor
&recipient_name=Ms.+Donna+Noble
&message_text=It+is+time+for+your+next+visit.
```

This web request will schedule a reminder call to Ms. Donna Noble at telephone number +1 773 555000 from "The Doctor" on Jan 7, 2009 at 8:56 AM US Central Time. First the system will play "Hello Ms. Donna Noble. This is The Doctor calling with an important message," and ask for consent to play the message. If the callee gives consent, the system will say "It is time for your next appointment."

Note that the time and date formats are very different from the formats used in SurVo web requests.

The return information from a Schedule web requests are very terse on errors: "Denied" for incorrect or missing account information or for invalid telephone numbers; "Invalid" if items are incorrect; "Your Reminder has been scheduled" if successful. If you schedule an event in the past (e.g., yesterday), the system will accept the appointment and make the call immediately.

7.2. Find Schedules & Set Schedule Mode

This web request affects "Schedules," which is a Smart Form that lets you define how other Smart Forms operate. Schedules can be defined by using Tools -> Schedule at the Ifbyphone web site; other Smart Forms can use these Schedules. Each hour of the day can be set to a different "mode." The most common modes are "Open" and "Closed," but you can define modes and give the the modes any name you please.

For example, you can create a Schedule Smart Form called "Marketing Department Schedule" and create the modes "Open," "Closed," and "Lunch Hour." The Schedule Smart Form at the web site lets you assign each hour in the day, for each day in the week, to one of these three modes. A typical arrangement would have the office closed on weekends, open during business hours, but on "Lunch Time" mode from noon to 1 PM. At any particular time of day, the Schedule is "in a mode": at 1 AM it's in the "Closed" mode, and at 12:34 PM it's in the "Lunch Time" mode.

Continuing with this example, we can configure this Schedule to control how other Smart Forms work. When you define a Click-to-Call, for example, you can associate the "Marketing Department Schedule" Schedule Smart Form with the Click-to-Call Smart Form, and the Click-to-Call Smart Form will let you choose what to do when the Schedule is in each mode. When the Schedule is in the mode "Lunch Time," for example, the Click-to-Call Smart Form might be sent to a particular clerk, but when the Schedule is in Closed mode, the calls would be sent directly to voice mail. (When you create a Click-to-Call at the web site and associate it with a schedule other than "24x7," you are asked what to do when the Schedule is in various modes.)

The Schedule web request lets you discover and control the modes of a Schedule. You can:

- 1. "get_current_mode": Determine the current mode of a Schedule.
- 2. "get_avail_modes": Determine what modes are legal for a Schedule.
- 3. "set_mode_hold": Override the Schedule: keep the Schedule in its current mode regardless of the time.

If all or most of your Click-to-Calls and other Smart Forms use the same Schedule, you can easily set all of your company's telephone systems to the mode "Closed" on a holiday or to "Emergency" on a snow day with a single web request.

The URI of a Schedule web request is `ibp_schedule_api.php`.

Application	Parameter	Value/Notes
Schedule	acct_id	Account ID.
	key	Public key.
	func	"get_current_mode", "get_avail_modes", "set_mode_hold".
	usr_schedule_id	The ID number of a Schedule.
	usr_mode_id	The ID number of a Schedule's mode.

NOTE: The ID numbers of Schedules can be seen on the web site at Tools -> Schedules. The ID numbers of the modes for a particular Schedule are returned as a bar-delimited list by this web request when the parameter "func" is set to "get_avail_modes."

To understand how all this works, let's take a set of three examples. First, we'll get a list of the available modes for a particular schedule in our account:

```
https://secure.ifbyphone.com/ibp_schedule_api.php
?acct_id=1234
&key=foofoofoo
&func=get_avail_modes
&usr_schedule_id=630
```

This will fetch the available modes for schedule 630 that belongs to account 1234. The output will be a combination of XML and a bar-delimited list:

```
Open|638||Closed|639||Lunch Time|3751
```

You'll note that this is a bar-delimited list in the familiar format "name1|value1||name2|value2," etc. We see the Schedule has three modes and the ID numbers associated with each mode.

Let's check to see the current mode of this Schedule:

```
https://secure.ifbyphone.com/ibp_schedule_api.php
?acct_id=1234
&key=foofoofoo
&func=get_current_mode
&usr_schedule_id=630
```

The response is a short bar-delimited list:

```
Open|638
```

The Schedule is in "Open" mode, and Smart Forms that use this Schedule will execute the actions specified for mode "Open." Because it's snowing and no one is in the office, we decide change the mode to "Closed." To set mode of the Schedule to "Closed" send "Closed" mode ID, 639:

```
https://secure.ifbyphone.com/ibp_schedule_api.php
?&acct_id=1234
&key=foofoofoo
&func=set_mode_hold
&usr_schedule_id=630
&usr_mode_id=639
```

The response is:

```
success
```

The Schedule is now set to "Closed" and any Smart Form that uses this Schedule will operate in accordance with the actions selected in that Smart Form for "Closed."

NOTE: If you set a Schedule's mode with a web request, that mode will remain in effect until you either (a) set a new mode with another web request or (b) use the web site to access the Schedule and reset the Schedule back to normal, time-driven operation.

8. New Style Web Requests

You've no doubt noticed that as we at Ifbyphone have expanded our list of available web requests, we have also changed and improved the API. The web requests in this chapter and subsequent chapters use an improved version of the API that is a bit easier to understand, easier to develop, and easier to debug. In addition, this version of the API moves away from using account numbers and passwords and relies instead on cryptographic keys to authenticate your web requests.

8.1. Format of New Style Web Requests

New style web requests all use the same URI, `/ibp_api.php`.

To select which Smart Form to access and what action to perform on that Smart Form we use the parameter "action." This parameter uses the format "smartform.method," where "smartform" is the name of the Smart Form we want to access and "method" is some particular action we want to perform with or to that Smart Form.

These web requests also require a key. The particular key that's required is the "API Key" listed on the Building Block ID page -- *not* the "Public Key" used by the Click-to Smart Form APIs. The parameter name is "api_key" (not "key").

A typical web request will appear as follows:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=smartform.method
```

and will also include other relevant parameters.

In another improvement, web requests all return valid XML. At a minimum, the XML will contain a "result" and a "result_description" element.

On success, the response to the web request will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description/>
</response>
```

in addition to other relevant XML elements.

In case of an error, the response will contain a description of the error:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
  <result>failed</result>
  <result_description>Missing Security Key</result_description>
</response>
```

8.2. Voice Mail

The action parameter for the Voice Mail Smart Form uses the format "vmail.method." The Voice Mail Smart Form has the following methods.

8.2.1. Create Voice Mailbox

Method	Parameter	Value/Notes
.createbox	name	Required – Name of the mail box. Values: alphanumeric string.
	pin	Required – PIN to use to access account. Value: 4 digits.
	api_key	Required - API key.
	email_address	Optional – Used if we send voice mail audio via email. Values: system will check for valid email formats.
	send_email	Optional – Send voice mail recordings via email? Values: 0 (false) or 1 (true).
	envelope	Optional – Whether to read back voice mail "envelopes" (time, date, etc.) when calling in to listen to voice mail. Values: 0 (false) or 1 (true).
Returned if success: <pre><response> <result>success</result> <result_description/> <box_id>254</box_id> </response></pre>		

NOTE: You need the "box_id" in the result in order to access the voice mailbox later with other web requests, e.g., .getmessages.

Example:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=vmail.createbox
&name=This+Box+Created+Online
&pin=5454
&send_mail=1
&email_address=speech@example.com
&envelope=0
```

This creates a Voicemail Smart Form named "This Box Created Online" that will send the voicemail messages to the email address "speech@example.com." The response to this request is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description></result_description>
  <box_id>3151</box_id>
</response>
```

Type error messages include "missing API key" if you get the name of the parameter wrong, and "invalid API key" if the key you supply is not valid.

8.2.2. Get Messages

This web request retrieves messages from your voice mailbox. For each message you receive an "envelope" about when the message arrived at the mailbox, and a link to an audio file that contains the message.

.getmessages	api_key	Required - API key.
	box_id	Required - the ID of the vmail box.
	greetings	Optional - In addition to messages, retrieve "greetings" recordings that are played to caller. Value: 1.

Returned if success:

```
<response>
  <result>success</result>
  <result_description/>
  <data>
    <message>
      <url>https://secure.ifbyphone.com/xxx</url>
      <envelope>
        5555551234 on Mon January 14, 2008, 6:23 pm
      </envelope>
    </message>
    <message>
      <url>https://secure.ifbyphone.com/xxx</url>
      <envelope>
        5555551234 on Tue January 15, 2008, 3:45 pm
      </envelope>
    </message>
  </data>
</response>
```

An example to retrieve messages in the previously-created mailbox, which had the ID 3151:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=vmail.getmessage
&box_id=3151
```

The response returns two voicemail messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , data)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT data (message)+>
<!ELEMENT message (url , envelope)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT envelope (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description/>
  <data>
    <message>
      <url>https://secure.ifbyphone.com/user_recordings/1/1234/vmail/3
151/20081028173340_8476266100.wav</url>
      <envelope>20081028173340_8476266100.wav (0 sec)</envelope>
    </message>
    <message>
      <url>https://secure.ifbyphone.com/user_recordings/1/1234/vmail/2
```

```

151/20081028172937_8476266100.wav</url>
  <envelope>20081028172937_8476266100.wav (0 sec)</envelope>
</message>
</data>
</response>

```

Note that instead of <result_description></result_description>, as in the previous "success" response, the system outputs <result_description/>. Standard software packages for parsing XML handle this typical variation without any trouble.

8.2.3. Record Voice Mail Greeting

You can record a personalized voice mailbox greeting by having Ifbyphone's servers call a phone number. The person who responds records a greeting.

.recordgreeting	api_key	Required - API key.
	box_id	Required - the mailbox ID for which the greeting is being recorded.
	recording_type	Required - which recording type. Valid options are: greeting, vacation.
	phone_to_call	Required - the phone number to call to record the greeting.
Returned on success: <pre> <response> <result>Success</result> <result_description>Call Attempted</result_description> </response> </pre>		

To set the greeting in the mailbox we created above, use:

```

https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=vmail.recordgreeting
&box_id=3151
&phone_to_call=7735559999

```

The result is:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>

```

```

]>
<response>
  <result>success</result>
  <result_description>Call Attempted</result_description>
</response>

```

8.3. Report

The "report" web requests let you search for historical data about usage on your account. You can search for information about Click-to Smart Forms and about Broadcasts. This same information, as well as other reports, is also available at the Ifbyphone web site from your account page; select the Reports tab.

NOTE: In a "report" web request, the "start_date" and "end_date" parameters define the date and time limits of the search for data -- **not** when to execute the search! The search executes immediately. The format for these parameters is YYYYMMDD.

Method	Parameter	Value/Notes
.clickto	api_key	Required - API key.
	start_date	Required - Earliest date to search; start time will be midnight of this day.
	end_date	Required - Last date to search; end time will be 23:59 of this day.
	click_id	Optional - a specific Smart Click-to-Call building block ID.
	ref	Optional - a specific value within the "ref" return value (example: for a report on all calls with ref = "home page").

Returned if success:

```

<data>
  <record>
    <Caller_ID>8475551234</Caller_ID>
    <Name>Phone-me-now</Name>
    <Page/>
    <Ref>12345678</Ref>
    <Click_Description>URL based</Click_Description>
    <Click_Date_Time>Mar 03 2008 4:25 PM</Click_Date_Time>
    <Number_Connected>8475554321</Number_Connected>
    <Call_Duration/>
    <Referral_Keywords/>
    <Referral_Domain/>
    <IP_Address>192.168.5.235</IP_Address>

```

```

        <Call_1_Result>Phone Busy</Call_1_Result>
        <Call_2_Result/>
    </record>
</data>

```

To get a list of all click-to actions between Nov 6, 2007 and Nov 4, 2008:

```

https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=report.clickto
&start_date=20071106
&end_date=20081104

```

Again, we emphasize that the search takes place immediately ("start_time" refers to the data we're searching, not the time when we begin our search). The output will be a series of <record> elements similar to the example in the table. If you'd like to see, for the same time period, the reports for just a single Click-to with a particular Building Block ID, use:

```

https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=report.clickto
&start_date=20071106
&end_date=20081104
&click_id=908

```

The output will be series of <record> elements for that particular Click-to Smart Form.

This next web request provides information about Broadcasts that you have performed in the past.

.broadcast	api_key	Required - API key.
	start_date	Required - Earliest date to search; start time will be midnight of this day.
	end_date	Required - Last date to search; end time will be 23:59 of this day.
	broadcast_id	Optional - Search only for Broadcasts with this ID.

```

Returned if success:
<data>
  <record>
    <Scheduled_Time>2008-08-28 14:06:37</Scheduled_Time>
    <Actual_Time>2008-08-28 14:06:37</Actual_Time>
    <Delay_if_Any>00:00:00</Delay_if_Any>
    <Number_Called>8476766624</Number_Called>
  </record>
</data>

```

```

    <Result_of_Call>Answered</Result_of_Call>
    <Response_ID>1234</Response_ID/>
    <Broadcast_ID>556123</Broadcast_ID>
  </record>
</data>

```

Example: To search for all broadcasts between July 14, 2007 and Nov 4, 2008:

```

https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=report.broadcast
&start_date=20070714
&end_date=20081104

```

The output will be a series of <record> elements similar to the example in the table. If you'd like to see, for the same time period, the reports for just a single Broadcast with a particular Building Block ID, use:

```

https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=report.clickto
&start_date=20071106
&end_date=20081104
&broadcast_id=105

```

The output will be series of <record> elements for that particular Broadcast.

8.4. SurVo

This web request lets you retrieve a recording performed by a SurVo Smart Form -- if that recording is stored in the Ifbyphone database.

Method	Parameter	Value/Notes
.get_recording	api_key	Required – API key.
	format	Required – Audio format of stream. Values: wav, mp3.
	survo_id	Required – Building Block ID of SurVo.
	unique_id	Required – ID number of interaction (see Note below).
	question	Required – The question number in that SurVo. Values: 1,2,3,...
	sample_rate	Optional – Audio rate of the streaming data, given in Hz. Values: 8000 (default), 11000 (wav only), 22050, 44100.

NOTES: The response to this web request is a stream of audio information, **not** a file name or a string of characters.

Also note that you can only retrieve recordings that are in the SurVo database. If you've set your SurVo to send the recording to you via email or through a NetGet and you didn't configure the SurVo to keep a copy in the Ifbyphone database, you won't be able to retrieve it.

About the "unique_id" number: Each time Ifbyphone puts a set of responses to a SurVo in its database, that interaction receives an ID number. You can view all ID numbers associated with a SurVo on the website: go to Services->SurVo, select the appropriate SurVo, and you will receive a list of ID numbers. At present there is no web request that accesses the list of ID numbers.

Finally, a SurVo may have multiple recordings associated with a particular ID number. The web request must include the SurVos question number to select the correct recording, even if there is only one recording for that incident or question for that SurVo.

Example: retrieve the audio recording for SurVo number 307, with ID number 93451.

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=survo.get_recording
&survo_id=307
&unique_id=93451
&format=mp3
&audio_rate=22050
```

The result will be an audio stream in the format you requested (MP3) and at the audio rate you requested (22.050 kHz). As a hint to PHP users, the function

```
curl_setopt($curlObject, CURLOPT_FILE, $aFilehandle);
```

will cause PHP to accept the stream from the remote source (in our case, Ifbyphone) and place it into the file with handle \$aFilehandle.

8.5. Verify-Me-Now

This web request initiates a phone call and connects the callee to a Verify-Me-Now Smart Form.

Method	Parameter	Value/Notes
--------	-----------	-------------

.verify	api_key	Required – API key.
	verify_id	Required – Building Block ID of a Verify-Me-Now Smart Form.
	phone_number	Required – the phone number to call.
	pin	Required – the PIN number that the user must enter in order to be verified. Values: Between 1 and 10 digits.
Returned on successful verification:		
<pre><response> <result>verified</result> <data> <sid>0805089531254086</sid> </data> </response></pre>		

To verify someone who gave the phone number +1 773 555 1111, using a pre-defined Verify-Me-Now Smart Form with Building Block ID 808, use:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=verifymenow.verify
&verify_id=808
&phone_number=7735551111
&pin=1234
```

The system will call the number and use the Verify-Me-Now Smart Form to ask the callee to say or enter the PIN number. If they say the correct PIN number -- in this case, 1234 -- the callee will be marked as "verified" by the system.

The web request will return a structure similar to the one given in the table above. The "sid" element ("session ID") provides a unique number that can be used to retrieve recordings, if any, that were made by the Find-Me-Now Smart Form.

NOTE: The only way to know the "sid" for a particular call is to remember it after the verifymenow.verify web request. If you do not make a note of the "sid" then, you will not be able to retrieve any recordings.

This next table shows how to retrieve recordings.

Method	Parameter	Value/Notes
--------	-----------	-------------

.get_recording	api_key	Required - API key.
	format	Required - Audio format of stream. Values: wav, mp3.
	sample_rate	Optional - Audio rate of the streaming data, given in Hz. Values: 8000 (default), 11000 (wav only), 22050, 44100.
	verify_id	Required - Building Block ID of a Verify-Me-Now Smart Form.
	sid	Required - sid (see previous table) for the particular recording requested.

NOTE: The response to this web request is a stream of audio information, **not** a file name or a string of characters.

The "sid" parameter is returned by the verifymenow.verify web request. If you do not record the sid, you will not be able to retrieve the recording.

To retrieve a wav-formatted string at 11000 Hz, for a Verify-Me-Now Smart Form with building block ID of 808 and a sid of 9876543, use:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=barbarbar
&action=verifymenow.get_recording
&verify_id=808
&sid=9876543
&format=wav
&audio_rate=11000
```

The result will be an audio stream in wav format.

8.6. Find Me

Find Me Smart Smart Forms let a caller reach not just a single telephone number, but a list of telephone numbers. The caller is then connected to whichever telephone number on the list is answered (sometimes called "the destination").

In this Guide, we will not explain all the options and the exact meanings of all parameters for the Find Me Smart Form. A complete discussion of how the Find Me list behaves, what options are available, and similar questions can be found in *Find Me Advanced User's Guide*. We also recommend that you spend some time at the Ifbyphone web site to create and test a few Find Me Smart Forms.

Ifbyphone provides a complete set of web requests for Find Me Smart Forms. First, let's see how to create a Find Me Smart Form. Here's the basic command, without all the additional parameters:

.create	api_key	API key.
	name	Name to assign to form. Value: string of 32 characters maximum.
Returned if success: <pre> <response> <result>success</result> <result_description/> <findme_id>111</findme_id> </response></pre>		

NOTE: Although this web request creates a Find Me Smart Form, to use it you will need to add telephone numbers. See the next web request, .add_number, for details.

The Find Me has an impressive number of optional parameters. Here's a list:

Optional Find Me Parameters		
Parameter	Description	Valid Values
usr_findme_type	Specify type of Find Me.	Values: 1, Individual Find Me (default); 2, Customer Service Hunt Group.
loop_count	Number of times to transverse list of destinations. After final transversal the Find Me will perform its end-of-list action.	Values: 1 - 5. Default: 1.
randomize	Telephone numbers in Find Me's list should be selected at random in attempt to connect caller.	Values: 1, randomize; 0, do not randomize (default).
record	Record the conversation after caller connects to one of the Find Me's destinations.	Values: 1, record; 0 or other numeric value, do not record (default).
use_screen	"Screen" caller for information before	Values: 1, screen call; 0 or other numeric value,

	attempting to connect caller.	do not screen (default).
screen_prompt	TTS screening message.	Value: text string. Default: Ifbyphone standard message.
dtmf_only	Person answering at a destination must use DTMF only to accept call (speech recognition disabled).	Values: 1, DTMF only; 0, accept speech and DTMF (default).
holdmusic	Audio file to play to caller as caller waits for destination to answer.	Name of audio file which must be in the account's "holdme" directory. Default: Ifbyphone standard audio file.
holdrepeat	How often the hold message is played to the caller.	Values: 0, only once per Find Me session; 1, only once between each attempt to connect to a destination; 2, repeat continuously between each connection attempt (default).
timeout	Number of seconds to ring each destination before attempting next destination (or end action).	Values: 10 – 60. Default: 30.
whisper_phrase	Whisper phrase, spoken to person who answers phone, used when call screening not enabled.	Value: string. Default: no phrase.
list_type	Type of hunt group.	Optional – Values: 1, individual (default); 2, customer service hunt group.
distribution_type	How calls reach list.	Values: 1, randomize; 2, round robin. Default : traverse list sequentially. (Note: default cannot be set as it has no associated value. To use default, do

		not send this parameter.)
findme_action	Action to take if Find Me reaches end of list.	See next table for values. Default: user told no one available and disconnected.
findme_action _parameter	A Smart Form's Building Block ID, if required by "findme_action" parameter.	See next table.

NOTE: The "findme_action" parameter allows you to specify what to do when none of the Find Me's destinations accept the call. By default, the caller gets a polite announcement and the call ends, but you can specify other treatment. See next table.

Let's create an example Find Me web request, including a few optional parameters:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=findme.create
&name=api-created+find+me+%2312
&list_type=1
&loop_count=1
&record=27
&use_screen=0
&whisper_phrase=Who+knows+what+lives+in+the+hearts+of+men%3F
```

The response is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
  <!ELEMENT response (result,result_description,findme_id)>
  <!ELEMENT result (#PCDATA)>
  <!ELEMENT result_description (#PCDATA)>
  <!ELEMENT findme_id (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description></result_description>
  <findme_id>3831</findme_id>
</response>
```

The result will be a Find Me Smart Form called "api-created find me #12," which the Ifbyphone server assigned Building Block ID 3831. We specified a list type of "individual" which will only be traversed once per call. Because "record" was set to something other than 1, no recording will be made. Calls will not be screened and the whisper phrase is "Who knows what lives in the hearts of men?"

Here's a table of values for the findme_action parameter. Each type of action has an associated numeric value. If the parameter results in the transfer to another Smart Form, then the parameter findme_action_parameter takes the value of the Building Block ID of a particular Smart Form. (If findme_action does not require findme_action_parameter, we note this as "N/A" for "Not Applicable.")

Action to Take	findme_action	findme_action_parameter
Tell user no one is available and disconnect	1	N/A (Default)
Transfer to Voice Mail	2	Voice Mail ID
Automatically accept call on last number tried	3	N/A
Route to a Virtual Receptionist	4	Virtual Receptionist ID
Route to a different Find Me list	5	Find Me ID
Route to a SurVo	6	SurVo ID
Automatically accept call without intro	7	N/A

Let's take the previous example; assuming that no one answers, I want the call to route to a particular Voice Mail Smart Form with Building Block ID 1259:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=findme.create
&name=api-created+find+me+%2312
&list_type=1
&loop_count=1
&record=27
&use_screen=0
&whisper_phrase=Who+knows+what+lives+in+the+hearts+of+men%3F
&findme_action=2
&findme_action_parameter=1259
```

The response to this web request will be the same XML as in the previous example.

Once a Find Me Smart Form exists, you must add telephone numbers to it -- a list of numbers for the Find Me to call as it attempts to connect the caller.

.add_number	api_key	API key.
	findme_id	Required – Building Block ID of Find Me.
	phone_number	Required – phone number.
Returned if success: <pre><response> <result>success</result> <result_description>Phone number 5555555555 added to findme 111</result_description> </response></pre>		

For example, to add a phone number to the Find Me Smart Form we created before, we first note that the response gave us a (Building Block) ID of 3831. Then our web request will be:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=findme.add_number
&phone_number=7735559999
&findme_id=3831
```

and the response will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>Phone number 7735559999 added to findme
3831</result_description>
</response>
```

You can modify the settings of a Find Me Smart Form. The basic web request parameters are as follows:

.update_list _settings	api_key	API key.
	findme_id	Required - ID of Find Me Smart Form to modify.
	name	Optional - Name to assign to form.
Returned if success: <pre><response> <result>success</result> <result_description>findme options for findme 259 have been</pre>		

```
updated</result_description>
</response>
```

You can add any of the parameters that you can use in the ".create" web request, as given in the table called "Optional Find Me Parameters."

For example, I'll update the Find Me Smart Form I've been working with, 3831: I will enable recording of all calls and rename the form "The New Form":

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=findme.update_list_settings
&findme_id=3831
&record=1
&&name=The+New+Form
```

The response is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>findme options for findme 3831 have been
updated</result_description>
</response>
```

It's rather useful to get a list of all the Find Me Smart Forms you have created:

.get_findme_list	api_key	API key.
<p>Returned if success:</p> <pre><response> <result>success</result> <result_description/> <data> <findme> <id>999</id> <name>Name for the Findme</name> </findme> </data> </response></pre>		

For example,

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=findme.get_findme_list
```

The response will be in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , data)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT data (findme)+>
<!ELEMENT findme (id , name)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description/>
  <data>
    <findme>
      <id>3831</id>
      <name>api-created find me #12</name>
    </findme>
    <findme>
      <id>1530</id>
      <name>Some other Find Me</name>
    </findme>
  </data>
</response>
```

Note that I found the one I just created and another Find Me Smart Form that I must have created previously through a web request or directly on the Ifbyphone web site.

To get a list of the phone numbers associated with a particular Find Me Smart Form,

.get_phone_list	api_key	API key.
	findme_id	Required – the ID of the Find Me from which to retrieve list of numbers.
<p>Returned if success:</p> <pre><response> <result>success</result> <result_description/> <data> <phone> <findme_number_id>999</findme_number_id> <number>5555555555</number> <priority>1</priority> </phone></pre>		

```
</data>
</response>
```

NOTE: The "priority" of a phone number, seen in the response, is whether it is the first, second, etc. phone number to be tried in order to connect the caller. The first number added through a web request gets priority 1, the second priority 2, and so forth.

Telephone number priorities can be adjusted by logging into the web site and editing the Smart Form.

For example,

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=findme.get_phone_list
&phone_number=7735559999
&findme_id=3831
```

Assuming that I've added two numbers to the Smart Form I created previously (3831), the response will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , data)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT data (phone)+>
<!ELEMENT phone (findme_number_id , number , priority)>
<!ELEMENT findme_number_id (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description/>
  <data>
    <phone>
      <findme_number_id>19512</findme_number_id>
      <number>7735559999</number>
      <priority>1</priority>
    </phone>
    <phone>
      <findme_number_id>19709</findme_number_id>
      <number>7735550000</number>
      <priority>2</priority>
    </phone>
  </data>
</response>
```

If you would like to delete one of these numbers from the list, you will need the "findme_number_id" you just retrieved and the following web request:

.delete_number	api_key	API key.
	findme_id	Required – ID of the Find Me from which to remove the phone number.
	findme_number_id	Required – ID of the phone number to remove.
<p>Returned if success:</p> <pre><response> <result>success</result> <result_description> Findme number id 999 removed from findme 111 </result_description> </response></pre>		

For example, in the previous example, we examined the list of numbers for Find Me Smart Form with ID 3831; it has two telephone numbers. The second one is telephone number +1 773 555 0000 with findme_number_id 19709. Let's remove that telephone number from the list:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=findme.delete_number
  &findme_id=3831
  &findme_number_id= 19709
```

Note that we don't use the phone number itself, just the ID number. This is an important distinction, especially if you've managed to enter the same number on the list twice -- you can choose which copy to delete. The result of this web request is

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>
    Findme number id 19709 removed from findme 3831
  </result_description>
</response>
```

Successful calls can be recorded. If you elect to do so, you can retrieve a list of all -- or specific -- recorded calls:

.	api_key	API key.
---	---------	----------

get_recorded_calls_list	findme_id	Optional – Return list for Find Me Smart Form with this Building Block ID.
Returned if success: <pre><response> <result>success</result> <result_description/> <data> <findme> <findme_id>111</findme_id> <recorded_call> <url> (path to file) </url> <date>04/02/2008 09:23:43</date> <callerid>5555555555</callerid> <connected_number>5555555555</connected_number> </recorded_call> </findme> </data> </response></pre>		

For example,

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=findme.get_phone_list
```

will return, assuming that I've just one recording,

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , data)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT data (findme)+>
<!ELEMENT findme (findme_id , recorded_call+)>
<!ELEMENT findme_id (#PCDATA)>
<!ELEMENT recorded_call (url , date , callerid , connected_number)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT callerid (#PCDATA)>
<!ELEMENT connected_number (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description/>
  <data>
```

```

<findme>
  <findme_id>3831</findme_id>
  <recorded_call>
    <url>https://secure.ifbyphone.com/user_recordings/1/1234/findme/3831/20081110153526_77355599999_7735550000.wav</url>
    <date>11/10/2008 15:35:26</date>
    <callerid>8479991234</callerid>
    <connected_number>7735550000</connected_number>
  </recorded_call>
</findme>
</data>

```

The URL lets me listen to a recording of the call. The "callerid" element is the caller ID of the person who called, and the "connected_number" is the telephone number to which the caller was connected by the Find Me.

Once you have listened to the recording, you may wish to erase it (Ifbyphone may charge you for storage of recorded calls; ask your account manager for details). To erase a stored call, use

.delete_recorded_call	api_key	API key.
	findme_id	Required – ID of Find Me of containing the call to delete
	recording_name	Required – file name of the recording to delete. See Note, below.
<p>Returned if success:</p> <pre> <response> <result>success</result> <result_description>Findme recording has been deleted</result_description> </response> </pre>		

NOTE: The "file name" of a recording is the final part of the URL of the call; see the example below. If you give the wrong file name, the error message will be "Findme delete recording failed - reason undefined."

Example: in the previous example we asked for a list of all recordings and the response indicated that a recording existed at the URL:

```

https://secure.ifbyphone.com/user_recordings/1/1234/findme/3831/20081110153526_77355599999_7735550000.wav

```

The "file name" associated with this recording is the very last part of the URL:

```

20081110153526_77355599999_7735550000.wav

```

To delete this recording, we send:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=findme.delete_recorded_call
  &findme_id=3831
  &recording_name=20081110153526_77355599999_7735550000.wav
```

and the response will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>Findme recording has been
deleted</result_description>
</response>
```

To delete a Find Me Smart Form entirely, use the delete web request.

.delete	api_key	API key.
	findme_id	Required – the ID of the Find Me to delete.
<p>Returned if success:</p> <pre><response> <result>success</result> <result_description> findme 111 has been deleted </result_description> </response></pre>		

As an example, let's get rid of that pesky Find Me Smart Form ID that we created with ID 3831:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=findme.delete
  &findme_id=3831
```

And the response is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>
```

```
<result>success</result>  
<result_description>  
  findme 3831 has been deleted  
</result_description>  
</response>
```

It's up to you, as the person who deleted the Find Me Smart Form, to make certain that the rest of your applications hosted at Ifbyphone that depended on the Find Me behave correctly. For example, if you had a phone number that routed to this Find Me, you can delete the Find Me -- the web request will be honored -- and the phone number will simply route to one of your other Find Me Smart Forms (the one with the lowest Building Block ID number).

8.7. Voice Broadcast

These web requests let you create, schedule, and stop a Voice Broadcast Smart Form, as well as retrieve information about a Voice Broadcast.

The Smart Form really has two parts. First there's the skeleton of the Smart Form itself, which consists of a name, an ID, an associated recording or two, and some configuration information. This Smart Form can have several "campaigns." A campaign consists of a set of phone numbers to call, a schedule of when to make the calls, and perhaps some configuration information.

First, here's how to create a Voice Broadcast Smart Form and, at the same time, optionally create a campaign for that Voice Broadcast:

Method	Parameter	Value/Notes
--------	-----------	-------------

.create	api_key	API key.
	name	Required – name of Voice Broadcast. Maximum of 32 characters.
	recording_phone_number	Required – record announcement to broadcast by calling this number. Call to this number will be made immediately.
	machine_detection	Optional – Values: 1, system uses different recording when a broadcast call reaches voicemail or answering machine; 0, no separate recording (default).
	use_transfer	Optional – Values: 1, callee can request transfer after receiving call; 0, no transfer (default).
	transfer_number	Optional – If use_transfer=1, telephone number to which to route callees on transfer request.
	schedule	Optional – Values: 1, schedule a campaign; 0, do not schedule (default).
	The following parameter is required if schedule=1	
	phone	Required. Defined in table "Voice Broadcast Campaign Parameters," below.
	The following parameters are available when schedule=1	
timestamp, attempts, retry, simul, cid, desc	Optional – Defined in table "Voice Broadcast Campaign Parameters," below.	
<p>Returned if success:</p> <pre> <response> <result>success</result> <result_description> Basic Audio Dialog Created </result_description> <audio_dialog_id>372</audio_dialog_id> </response> </pre>		

NOTE: If you create a Voice Broadcast and set "schedule" to "1" to also create a campaign, the schedule for the campaign is not very flexible. You may specify a

starting time in the future, but you may not specify an end time or hours during which calls can be made.

Example:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=broadcast.create
&name=Voice+Broadcast+%2312+at+2008-11-11+07%3A13
&phone_number_list=7735551212
&recording_phone_number=7735550000
```

This creates a Voice Broadcast named "Voice Broadcast #12 at 2008-11-11 07:13" that will attempt to call +1 773 555 1212. The Ifbyphone servers will immediately call +1 773 555 0000 to record the message of he broadcast. There is no campaign associated with this Voice Broadcast just yet.

NOTE: The announcement heard by the person who records the message is "Your Broadcast was scheduled successfully," even if you did not schedule a campaign but only created the Voice Broadcast.

The XML result will be

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , audio_dialog_id)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT audio_dialog_id (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>Basic Audio Dialog Created</result_description>
  <audio_dialog_id>20901</audio_dialog_id>
</response>
```

NOTE: You may need the "audio_dialog_id" for later use, and the XML response to the create web request is (at present) the only way to obtain it.

Another example is when you create a Voice Broadcast and also schedule a campaign:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=broadcast.create
&name=Broadcast%20%2312%20created%202008-11-13%2007%3A55
&recording_phone_number=7735551212
&schedule=1
&phone=7735550000
&desc=Scheduled%20at%20creation
&cid=7735559999
```

```
&timestamp=2008-11-14%2007%3A55%20-0600
```

This creates a Voice Broadcast named "Voice Broadcast created 2008-11-13 07:55" that will attempt to call +1 773 555 1212. The Ifbyphone servers will immediately call +1 773 555 0000 to record the message of the broadcast. The campaign associated with this broadcast is called "Scheduled at creation" and will commence at 2008-11-14 at 07:55, Central Standard Time. The caller ID displayed to people who receive the calls will be +1 773 555 9999.

The XML response will contain both the Voice Broadcast ID ("audio_dialog_id") and the campaign's ID (basic_broadcast_id):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , basic_broadcast_id ,
audio_dialog_id)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT basic_broadcast_id (#PCDATA)>
<!ELEMENT audio_dialog_id (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>
    Basic Audio Broadcast Created
  </result_description>
  <basic_broadcast_id>61281</basic_broadcast_id>
  <audio_dialog_id>22851</audio_dialog_id>
</response>
```

Note that the XML response is different than the one in the previous example, with its focus on the campaign ("Basic Audio Broadcast Created") instead of on the Voice Broadcast Smart Form ("Basic Audio Dialog Created"). You must remember the Voice Broadcast ID and the campaign ID if you want to manipulate them via the API at some future point.

To schedule a campaign for a previously created Voice Broadcast, either because the campaign was never scheduled in the first place or because you wish to broadcast an additional campaign, use the .schedule web request.

.schedule	api_key	API key.
	audio_dialog_id	Required – Basic Broadcast Dialog to schedule.
	phone_number_list	Required – list of phone numbers, as defined in table "Voice Broadcast Campaign Parameters," below.
	dstime, detime, timestamp, edate, desc, simul, attempts, type, retry, cid	As defined in table "Voice Broadcast Campaign Parameters," below.
<p>Returned if success:</p> <pre> <response> <result>success</result> <result_description> 1 Voice Broadcast(s) scheduled </result_description> <basic_broadcast_id>150</basic_broadcast_id> </response> </pre>		

As in the create request, the schedule web request uses parameters from the table below.

Voice Broadcast Campaign Parameters	
phone, phone_number_list	<p>The phone number(s) scheduled to receive the Broadcast. For historical reasons, the ".create" web request uses the name "phone" for this parameter while the ".schedule" web request uses the name "phone_number_list" for the same parameter. Regardless of the name, the values for the parameter are the same for both web requests.</p> <p>May be a single phone number, a " " delimited list of phone numbers, or a fully qualified URL to a web-accessible and valid CSV file. See Notes, below.</p>
timestamp	<p>Optional – Start time; for format, see Note below. Default: begin campaign immediately (or after broadcast message recorded, if not already recorded).</p>
edate	<p>Required to schedule: end of campaign at date/time. Format YYYY-MM-DD HH:MM (24 hour</p>

	format). Must be chronologically later than "timestamp." Note that this parameter is ignored if the "type" parameter is set to "2" (As fast as possible).
dstime	(Optional) The "Daily Call Range" start time of each day when campaign calls can be made. Values: hours and minutes in HH:MM 24 hour format. Default: 00:01.
detime	(Optional) The "Daily Call Range" last time of each day when campaign calls can be made. Values: hours and minutes in HH:MM 24 hour format. Value must chronologically later in the day than the value for "dstime." Default: 23:59.
type	(Optional) Scheduling algorithm for this campaign. Values: 1, "Spread calls evenly"; 2, "As fast as possible" (default).
attempts	(Optional) Maximum number of retry attempts for each phone number. Values: 1 (default, no retries), 2, 3.
retry	(Optional) Number of minutes between retry attempts on a phone number. Values: 5 (default), 10, 15, 30, 60, 90, 120.
simul	(Optional) Number of simultaneous calls to be placed each interval, that is, during "retry" time. Values: 1 (default), 2, 3.
cid	(Optional) Caller ID displayed to recipients of the Voice Broadcast. Values: any account toll-free number or account registered number. Default: account primary toll free number.
desc	(Optional) Identifier for the Voice Broadcast used for GUI management. Values: string of alphanumeric characters and spaces, maximum 32 characters. Default: none.

NOTE: If the value of phone/phone_number_list is a CSV file, the file must end with ".csv." In addition, the CSV file must have at least one column with the heading "numbers" that contains the telephone numbers.

NOTE: The format of timestamp is "YYYY-mm-dd HH:MM <offset from UTC/GMT>." For example, November 10, 2008 at 8:32 AM, Central Standard time, is written as "2008-11-10 08:30 -0600." During summer Daylight Saving Time, the offset in the Central Time Zone is -0500 from UTC/GMT.

Example: we will create a campaign for Voice Broadcast 20901. The start time will be 2008-11-13 at 16:08, Central Standard Time. The end date will be 2008-11-22 at 16:08, and calls will be permitted all day. We'll grab the calls from a CSV file online at <http://www.example.com/files/numberstocall.csv>. To make it easy to find on the web site, we will use the "desc" parameter and name this "campaign 12." We'll set "type" to 2 to make calls as quickly as possible.

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=2c9e37ac599442f3e897c4513fd50f20901ea1a1
&action=broadcast.schedule
&audio_dialog_id=20901
&phone_number_list=http%3A%2F%2Fwww.example.com%2Ffiles
%2Fnumberstocall.csv
&timestamp=2008-11-13%2016%3A08%20-0600
&edate=2008-11-22%2016%3A08
&desc=campaign%2012
&type=2
&dstime=00%3A00
&detime=23%3A59
```

The result will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , basic_broadcast_id ,
audio_dialog_id)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT basic_broadcast_id (#PCDATA)>
<!ELEMENT audio_dialog_id (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>
    Basic Audio Broadcast Created
  </result_description>
  <basic_broadcast_id>60451</basic_broadcast_id>
  <audio_dialog_id>20901</audio_dialog_id>
</response>
```

Note the "basic_broadcast_id" element. This ID identifies the campaign, and at present there's no other way to discover this ID.

To find the status:

.status	basic_broadcast_id	Required – ID of a particular campaign, not of a Voice Broadcast. ID is returned as "basic_broadcast_id" parameter when a campaign is created.
----------------	--------------------	---

	gmt	Optional – Returns times in given time zone. Values: "Eastern" (default), "Central", "Mountain", "Pacific", "Alaska", "Hawaii".
<p>Returned if success:</p> <pre><response> <result>success</result> <call> <call_time_scheduled>2008-05-29 13:05:03</call_time_scheduled> <call_time_actual>2008-05-29 13:05:03</call_time_actual> <call_delay>00:00:00</call_delay> <number_called>8476766624</number_called> <call_result>Scheduled</call_result> </call> </response></pre> <p>"call_result" element values: Scheduled, Live Person, Answering Machine, HANGUP, or NOTSURE.</p>		

Example: In the previous example, we created a campaign, and the basic_broadcast_id was 60451. To get the status of that campaign:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=broadcast.status
  &basic_broadcast_id=20921
```

The response will be similar to the one given in the table:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , call+)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT call (call_time_scheduled , call_time_actual , call_delay ,
number_called , call_result)>
<!ELEMENT call_time_scheduled (#PCDATA)>
<!ELEMENT call_time_actual (#PCDATA)>
<!ELEMENT call_delay (#PCDATA)>
<!ELEMENT number_called (#PCDATA)>
<!ELEMENT call_result (#PCDATA)>
]>
<response>
  <result>success</result>
  <call>
    <call_time_scheduled>2008-11-15 17:34</call_time_scheduled>
    <call_time_actual>2008-11-12 00:00</call_time_actual>
    <call_delay/>
    <number_called>7735551111</number_called>
```

```

    <call_result>Scheduled</call_result>
  </call>
<call>
  <call_time_scheduled>2008-11-19 18:30</call_time_scheduled>
  <call_time_actual>2008-11-12 00:00</call_time_actual>
  <call_delay/>
  <number_called>7735552222</number_called>
  <call_result>Scheduled</call_result>
</call>
</response>

```

There were two numbers in the CSV file, and the system didn't make any outbound calls. The "call_result" of both is "Scheduled" and the "call_time_actual" isn't meaningful.

NOTE: If you make a .status web request when no campaigns are scheduled, you will receive a result of "failed" with "result_description" of "No Basic Broadcasts Stopped," which is the same response as when you attempt to get status for a non-existent Voice Broadcast.

To stop all scheduled series of calls for a particular Voice Broadcast, use the .stop web request.

.stop	api_key	API key.
	basic_broadcast_id	Required – broadcast to stop.
<p>Returned if success:</p> <pre> <response> <result>success</result> <result_description> 2 Basic Broadcast(s) Stopped </result_description> </response> </pre>		

Example:

```

https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=broadcast.stop
  &basic_broadcast_id=20921

```

The result will be similar to the one given in the table:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
]>
<response>

```

```
<result>success</result>  
<result_description>  
  1 Basic Broadcast(s) Stopped  
</result_description>  
</response>
```

NOTE: If you make a .stop web request when no more calls for a campaign are scheduled, you will receive a result of "failed" with "result_description" of "No Basic Broadcasts Stopped." This is the same response as when you attempt to stop a non-existent campaign.

8.8. Conference

These web requests allow you create a Conference Smart Form, schedule it, and manage attendees.

Method	Parameter	Value/Notes
.schedule	api_key	Required – the account’s API key.
	attendee_list	Required – list of attendees. See note, below.
	scheduled_time	Optional – Scheduled start time for the conference. See note for format, below. Time will be rounded up to the nearest 5-minute interval. Default: Immediate start.
	conference_length	Optional – Length of the conference in minutes. Values: 10-240. Time will be rounded up to the nearest 5-minute interval. Default: 30.
	conference_name	Optional – Name for the conference. Value: String, max length 32 characters.
	pin	Optional – PIN that allows a delegate to take control of conference. Value: 4 digits.
	allow_ouerrun	Optional – If specified (value is irrelevant), active conference participants will not be forcefully

		disconnected when the conference has expired, but new inbound callers will still be denied from further participation and the dashboard call out feature will be disabled.
	inboundonly	Optional – Conference will be available for inbound calls beginning 5 minutes before its scheduled time, but attendees will not be called. Value: any value.
	invitations	Optional – Send email to all attendees with email addresses with participation information. Value: any value.

Returned if success:

```
<response>
  <result>success</result>
  <result_description>Successfully scheduled the conference with 2
  requested attendee(s)</result_description>
  <usr_conference_id>112</usr_conference_id>
  <conference_name></conference_name>
  <scheduled_time>2008-09-03 16:30:00 -0600</scheduled_time>
  <conference_length>60</conference_length>
  <pin></pin>
  <status>0</status>
  <attendees_added>2</attendees_added>
  <sys_conference_server_id>2</sys_conference_server_id>
  <dnis>8003415801</dnis>
</response>
```

NOTE: The parameter "attendee_list" can have one of these three formats:

(1) A list of participants. Each participant is described by a list: a phone number; a name; and optionally an email address. As usual, to send a list of lists, separate each list from the next list by two bars.

Example:

Donna Noble|7735551212|dnoble@example.com||Martha Jones|7735552323

This participant list contains contact information for two participants. Note that for one participant we had the email address and for the other we did not.

(2) CSV Internet-accessible file. The value of attendee_list can be a fully-qualified HTTP(S) or FTP(S) URL to a CSV file. The file must end in ".csv." The file's structure: a required column with the header "phone_number"; a required column

with the header "attendee_name"; an optional column with the header "email_address."

(3) Uploaded file. The value can be a file upload, 512KB maximum. Structure: a required column "phone_number"; required column "attendee_name"; optional column "email_address".

NOTE: The format of "scheduled_time" is "YYYY-mm-dd HH:MM:SS <optional: offset from UTC/GMT>." For example, November 10, 2008 at 8:32 AM, Central Standard time, is written as "2008-11-10 08:30:00 -0600." During summer Daylight Saving Time, the offset in the Central Time Zone is -0500 from UTC/GMT. The default offset is -0500, namely Eastern Standard Time.

NOTE: Invalid scheduled times will report back an error.

Example 1: provide list of attendees in the value of "attendee_list".

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.schedule
  &conference_name=Conference%20%2317
  &scheduled_time=2008-11-18%2014%3A18%3A00%20-0600
  &conference_length=10
  &inboundonly=1
  &invitations=1
  &attendee_list=7735551212%7CJohn%20Smith%7Cjsmith%40example.com%7C
%7C7735552222%7CMary%20Cellphone
```

This will result in a Conference Smart Form to support a conference call between John Smith and "Mary Cellphone." The participants must call in to the conference (rather than being called), and John Smith will be notified by email with details about the call. The XML output of this web request is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , usr_conference_id ,
conference_name , scheduled_time , conference_length , pin ,
allow_overrun , status , attendees_added , sys_conference_server_id ,
dnis)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT conference_name (#PCDATA)>
<!ELEMENT scheduled_time (#PCDATA)>
<!ELEMENT conference_length (#PCDATA)>
<!ELEMENT pin (#PCDATA)>
<!ELEMENT allow_overrun (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT attendees_added (#PCDATA)>
```

```
<!ELEMENT sys_conference_server_id (#PCDATA)>
<!ELEMENT dnis (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>
    Successfully scheduled the Smart Conference with 2 out of 2 valid
attendee(s)
  </result_description>
  <usr_conference_id>3651</usr_conference_id>
  <conference_name>
    Conference #17
  </conference_name>
  <scheduled_time>2008-11-18 14:20:00 -0600</scheduled_time>
  <conference_length>10</conference_length>
  <pin/>
  <allow_overrun>1</allow_overrun>
  <status>1</status>
  <attendees_added>2</attendees_added>
  <sys_conference_server_id>2</sys_conference_server_id>
  <dnis>8005551212</dnis>
</response>
```

Note that the output contains the conference ID ("usr_conference_id"), which you will need later if you attempt to access this conference using the API, as well as the "DNIS" -- the number that attendees should call in order to participate in the conference.

Here's another example, when the list of attendees is accessible via the web. Instead of sending a list of phone numbers, names, and email addresses, send the URL of a CSV file that contains the "attendee_list".

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.schedule
  &conference_length=10
  &inboundonly=1
  &invitations=1
  &scheduled_time=2008-11-18%2014%3A18%3A00%20-0600
  &attendee_list=http%3A%2F%2Fwww.example.com%2Ffiles
%2Fconf_numberstocall.csv
```

The system retrieves the CSV file at example.com and schedules the call. The output is similar to the example above.

NOTE: This CSV file is read when the Conference Smart Form is first created, **not** (for example) just before the call begins. You cannot use URL-based attendee lists to create a Conference Smart Form in advance and decide, at the last moment, who will be called by changing the contents of the CSV file. If you want to modify the list of attendees, use the web requests described below.

Finally, an example of uploading the CSV file:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=conference.schedule
&conference_length=10
&inboundonly=1
&invitations=1
&scheduled_time=2008-11-18%2014%3A18%3A00%20-0600
```

Note that we didn't show "attendee_list" in the list of parameters above. That's because each server-side language has its own method of uploading files. In the version of PHP that we use for much of our testing, the way to upload a file is to give the name of the parameter, followed by an =, followed by a "@" character, and then the complete path to the file name. If the file is at /tmp/conf_nummberstocall.csv, then we'd use:

```
attendee_list=@/tmp/conf_numberstocall.csv
```

which translates to:

```
&attendee_list=%40%2Ftmp%2Fconf_numberstocall.csv
```

inside the PHP programming after URL encoding. Python would use a different method to send the CSV file. Each server side language also has its quirks about what files the server can access.

To get a list of conferences,

Method	Parameter	Value/Notes
.list	api_key	Required – the account’s API key.
<p>Returned if success:</p> <pre><response> <result>success</result> <result_description></result_description> <conference> <usr_conference_id>113</usr_conference_id> <conference_name></conference_name> <scheduled_time>2008-09-03 16:30:00 -0600</scheduled_time> <conference_length>60</conference_length> <pin></pin> <status>0</status> <attendees>2</attendees> </conference> </response></pre>		

For example,

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=conference.schedule
```

The output will be similar to what appears in the table, above:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , conferences+)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT conferences (usr_conference_id , conference_name ,
scheduled_time , conference_length , pin , allow_overrun , status ,
attendees , participants , sys_conference_server_id , dnis)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT conference_name (#PCDATA)>
<!ELEMENT scheduled_time (#PCDATA)>
<!ELEMENT conference_length (#PCDATA)>
<!ELEMENT pin (#PCDATA)>
<!ELEMENT allow_overrun (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT attendees (#PCDATA)>
<!ELEMENT participants (#PCDATA)>
<!ELEMENT sys_conference_server_id (#PCDATA)>
<!ELEMENT dnis (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description/>
  <conference>
    <usr_conference_id>3651</usr_conference_id>
    <conference_name>Conference #17 at 2008-11-17
12:</conference_name>
    <scheduled_time>2008-11-17 13:55:00 -0500</scheduled_time>
    <conference_length>10</conference_length>
    <pin/>
    <allow_overrun>1</allow_overrun>
    <status>2</status>
    <attendees>2</attendees>
    <participants>0</participants>
    <sys_conference_server_id>2</sys_conference_server_id>
    <dnis>8003415801</dnis>
  </conference>
</response>
```

If more than one Smart Conference has been scheduled, completed, or is active, you will receive multiple "conference" elements.

NOTE: The "status" element in the response has three values: 0 - conference scheduled for the future; 1 - conference active, i.e., in progress; 2 - conference completed.

For the same information, but only for a specific conference, use the following web request. You'll need the "usr_conference_id" which you received when you created the conference, or when you use the "list" web request.

Method	Parameter	Value/Notes
.details	api_key	Required – the account’s API key.
	usr_conference_id	Required – ID of specific Conference.
Returned if success: <pre> <response> <result>success</result> <result_description></result_description> <usr_conference_id>113</usr_conference_id> <conference_name></conference_name> <scheduled_time>2008-09-03 16:30:00 -0600</scheduled_time> <conference_length>60</conference_length> <pin></pin> <status>0</status> <participants>0</participants> <sys_conference_server_id>2</sys_conference_server_id> <dnis>8003415801</dnis> </response> </pre>		

For example:

```

https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.schedule
  &usr_conference_id=3731
  
```

requests information only for conference 3731. The response will be:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , usr_conference_id ,
conference_name , scheduled_time , conference_length , pin ,
allow_overrun , status , attendees , participants ,
sys_conference_server_id , dnis)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT conference_name (#PCDATA)>
<!ELEMENT scheduled_time (#PCDATA)>
<!ELEMENT conference_length (#PCDATA)>
<!ELEMENT pin (#PCDATA)>
<!ELEMENT allow_overrun (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT attendees (#PCDATA)>
<!ELEMENT participants (#PCDATA)>
<!ELEMENT sys_conference_server_id (#PCDATA)>
<!ELEMENT dnis (#PCDATA)>
]>
<response>
  
```

```
<result>success</result>
<result_description/>
<usr_conference_id>3731</usr_conference_id>
<conference_name>Smart Conference 11-17-2008</conference_name>
<scheduled_time>2008-11-18 15:20:00 -0600</scheduled_time>
<conference_length>10</conference_length>
<pin/>
<allow_overrun>1</allow_overrun>
<status>0</status>
<attendees>2</attendees>
<participants>0</participants>
<sys_conference_server_id>2</sys_conference_server_id>
<dnis>8003415801</dnis>
</response>
```

In other words, this response provides the same details as one returned by the "list" web request, but for a specific Conference Smart Form.

To find out who the participants are, you have two different web requests. First, you can ask for a list of all participants.

Method	Parameter	Value/Notes
.attendee.list	api_key	Required – the account’s API key.
	usr_conference_id	Required – ID of specific Conference.

```
Returned if success:
<response>
  <result>success</result>
  <result_description>Conference attendee list successfully
generated</result_description>
  <attendee>
    <usr_conference_attendee_id>289</usr_conference_attendee_id>
    <attendee_name>Joe</attendee_name>
    <phone_number>1112223333</phone_number>
    <email_address></email_address>
    <participation>
      <usr_conference_participation_id></usr_conference_participat
ion_id>
      <sid></sid>
      <caller_id></caller_id>
      <status>0</status>
      <user_number></user_number>
      <muted>0</muted>
      <active_minutes>0</active_minutes>
      <elapsed_minutes></elapsed_minutes>
    </participation>
  </attendee>
</attendee>
```

Method	Parameter	Value/Notes
		<pre> <usr_conference_attendee_id>290</usr_conference_attendee_id> <attendee_name>Fred</attendee_name> <phone_number>2223334444</phone_number> <email_address>fred@example.com</email_address> <participation> <sid></sid> <caller_id></caller_id> <status>0</status> <user_number></user_number> <muted>0</muted> <active_minutes>0</active_minutes> <elapsed_minutes></elapsed_minutes> </participation> </attendee> </response> </pre>

NOTE: The "status" element in the response has three values: 0 - did not participate; 1 - currently active; 2 - currently disconnected.

For example:

```

https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.attendee.list
  &usr_conference_id=3731

```

requests a list of attendees for a Conference Smart Form with ID 3731. The response will be:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , attendee+)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT attendee (usr_conference_attendee_id , attendee_name ,
phone_number , email_address , participation+)>
<!ELEMENT usr_conference_attendee_id (#PCDATA)>
<!ELEMENT attendee_name (#PCDATA)>
<!ELEMENT phone_number (#PCDATA)>
<!ELEMENT participation (usr_conference_participation_id , sid ,
caller_id , status , user_number , muted , active_minutes ,
elapsed_minutes)>
<!ELEMENT usr_conference_participation_id (#PCDATA)>
<!ELEMENT sid (#PCDATA)>
<!ELEMENT caller_id (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT user_number (#PCDATA)>
<!ELEMENT muted (#PCDATA)>
<!ELEMENT active_minutes (#PCDATA)>
<!ELEMENT elapsed_minutes (#PCDATA)>

```

```

]>
<response>
  <result>success</result>
  <result_description>Smart Conference attendee list successfully
generated</result_description>
  <attendee>
    <usr_conference_attendee_id>11911</usr_conference_attendee_id>
    <attendee_name>Rose Tyler</attendee_name>
    <phone_number>7735552222</phone_number>
    <email_address/>
    <participation>
      <usr_conference_participation_id/>
      <sid/>
      <caller_id/>
      <status>0</status>
      <user_number/>
      <muted>0</muted>
      <active_minutes>0</active_minutes>
      <elapsed_minutes/>
    </participation>
  </attendee>
  <attendee>
    <usr_conference_attendee_id>12181</usr_conference_attendee_id>
    <attendee_name>Harriet Jones</attendee_name>
    <phone_number>7735551212</phone_number>
    <email_address>formerminister@example.com</email_address>
    <participation>
      <usr_conference_participation_id>8921</usr_conference_participat
ion_id>
      <sid>0811188727015299</sid>
      <caller_id>7735559999</caller_id>
      <status>1</status>
      <user_number/>
      <muted>0</muted>
      <active_minutes>10</active_minutes>
      <elapsed_minutes>12.2</elapsed_minutes>
    </participation>
  </attendee>
</response>

```

The conference has two possible attendees, Rose Tyler and Harriet Jones. Rose didn't participate in the conference. Harriet Jones did call in, is currently active, and has been for the past 10 minutes. Harriet Jones had a phone number of +1 773 555 1212 listed, but she actually called in from +1 773 555 9999 (the "caller_id" element).

The next web request returns information about a specific attendee.

Method	Parameter	Value/Notes
.attendee.details	api_key	Required – the account's API key.

Method	Parameter	Value/Notes
	usr_conference_id	Required – ID of specific Conference.
	usr_conference_attendee_id	Required - ID of specific attendee.

For example:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=conference.attendee.list
&usr_conference_id=3731
&usr_conference_attendee_id=12181
```

requests information for an attendee with ID 12181 who is participating in the Conference Smart Form with ID 3731. The information returned is the same as that for the previous "list" web request, but for only a single participant:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description ,
usr_conference_attendee_id , attendee_name , phone_number ,
email_address , participation+)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT usr_conference_attendee_id (#PCDATA)>
<!ELEMENT attendee_name (#PCDATA)>
<!ELEMENT phone_number (#PCDATA)>
<!ELEMENT email_address (#PCDATA)>
<!ELEMENT participation (usr_conference_participation_id , sid ,
caller_id , status , user_number , muted , active_minutes ,
elapsed_minutes)>
<!ELEMENT usr_conference_participation_id (#PCDATA)>
<!ELEMENT sid (#PCDATA)>
<!ELEMENT caller_id (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT user_number (#PCDATA)>
<!ELEMENT muted (#PCDATA)>
<!ELEMENT active_minutes (#PCDATA)>
<!ELEMENT elapsed_minutes (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>Smart Conference attendee details successfully
generated</result_description>
  <usr_conference_attendee_id>12181</usr_conference_attendee_id>
  <attendee_name>Harriet Jones</attendee_name>
  <phone_number>773551212</phone_number>
  <email_address>formerminister@example.com</email_address>
```

```
<participation>
  <usr_conference_participation_id>8921</usr_conference_participatio
n_id>
  <sid>0811188727015299</sid>
  <caller_id>7735559999</caller_id>
  <status>2</status>
  <user_number/>
  <muted>0</muted>
  <active_minutes>10</active_minutes>
  <elapsed_minutes>12.2</elapsed_minutes>
</participation>
</response>
```

In other words, this response contains same information as the one for the "list" web request.

To add one or more attendees to an existing Conference Smart Form, use the following web request. If the conference call is already in progress, the Conference Smart Form can make outbound calls to the new attendees to add them to the conference call.

Method	Parameter	Value/Notes
.attendee.add	api_key	Required – the account’s API key.
	usr_conference_id	Required – conference ID to which to add attendee(s).
	attendee_list	Required – See note, above, for format of attendee_list.
	invitations	Optional – Send email to all attendees with email addresses with participation information. Value: any value.
	immediate	Optional – Call attendee immediately to add attendee to conference. Value: any value.

Returned if success:

```
<response>
  <result>success</result>
  <result_description>Successfully added the attendee(s) to the
conference</result_description>
  <usr_conference_id>113</usr_conference_id>
  <attendees_added>1</attendees_added>
  <attendee>
    <usr_conference_attendee_id>285</usr_conference_attendee_id>
    <attendee_name>Jeff</attendee_name>
    <phone_number>3334445555</phone_number>
    <email_address>jeff@example.com</email_address>
```

Method	Parameter	Value/Notes
</attendee> </response>		

For example, add a single individual to a conference:

```
https://secure.ifbyphone.com/ibp_api.php
?api_key=foofoofoo
&action=conference.attendee.add
&usr_conference_id=3731
&attendee_list=7737648727%7CRose%20Tyler%7Crtyler%40example.com
&immediate=1
&invitations=1
```

This adds "Rose Tyler" to the Conference Smart Form with ID 3731. Rose will receive an email notification, and the Conference Smart Form will immediately call her to add her to call. The response is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , usr_conference_id ,
attendees_added , attendee+)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT attendees_added (#PCDATA)>
<!ELEMENT attendee (usr_conference_attendee_id , attendee_name ,
phone_number , email_address)>
<!ELEMENT usr_conference_attendee_id (#PCDATA)>
<!ELEMENT attendee_name (#PCDATA)>
<!ELEMENT phone_number (#PCDATA)>
<!ELEMENT email_address (#PCDATA)>
]>
<response>
  <result/>
  <result_description>Successfully added 1 out of 1 valid attendee(s)
to the Smart Conference</result_description>
  <usr_conference_id>3731</usr_conference_id>
  <attendees_added>1</attendees_added>
  <attendee>
    <usr_conference_attendee_id>12181</usr_conference_attendee_id>
    <attendee_name>Rose Tyler</attendee_name>
    <phone_number>7735552222</phone_number>
    <email_address>rtyler@example.com</email_address>
  </attendee>
</response>
```

To remove one or more attendees:

Method	Parameter	Value/Notes
.attendee.remove	api_key	Required – the account’s API key.
	usr_conference_id	Required – conference ID from which to remove attendee(s).
	usr_conference_attendee_id	Required - List of attendees to remove from Conference Smart Form. Value: list of one or more ID's. See Note below.

Returned if success:

```
<response>
  <result>success</result>
  <result_description>Successfully removed the attendee(s) from the conference</result_description>
  <usr_conference_id>113</usr_conference_id>
  <attendees_removed>2</attendees_removed>
</response>
```

NOTE: Attendees can only be removed by this method before they participate in the call; once they've participated they cannot be removed, even if they are not currently connected to the conference call. To remove an attendee from a call in progress, use the "attendee.kick" web request.

To remove an attendee with ID 12181 from Conference Smart Form with ID 3731:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.attendee.remove
  &usr_conference_id=3731
  &usr_conference_attendee_id=12181
```

and the result will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , usr_conference_id ,
attendees_removed)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT attendees_removed (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>Successfully removed 1 attendee(s) from the
Smart Conference</result_description>
  <usr_conference_id>3731</usr_conference_id>
  <attendees_removed>1</attendees_removed>
```

</response>

Method	Parameter	Value/Notes
.attendee.mute	api_key	Required – the account’s API key.
	usr_conference_id	Required – conference ID.
	user_number	Required – User number of attendee to mute. See Note below.
Returned if success: <pre> <response> <result>success</result> <result_description>Conference participant mute successfully performed</result_description> <result_action>mute</result_action> <usr_conference_id>113</usr_conference_id> <user_number>01</user_number> </response> </pre>		

NOTE: The user_number may be found by using the "attendee.list" or "attendee.details" web requests. The response for each attendee includes a participation element, which has a user_number element if the attendee is currently on a call.

For example:

```

https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.attendee.mute
  &usr_conference_id=3731
  &user_number=28971

```

will mute an attendee with user_number 28971 in a Conference Smart Form with ID 3731. The response will be:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , result_action ,
usr_conference_id , user_number)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT result_action (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT user_number (#PCDATA)>
]>

```

```
<response>
  <result>success</result>
  <result_description>Smart Conference participant mute successfully
  performed</result_description>
  <result_action>mute</result_action>
  <usr_conference_id>3731</usr_conference_id>
  <user_number>28971</user_number>
</response>
```

To unmute:

Method	Parameter	Value/Notes
.attendee.unmute	api_key	Required – the account’s API key.
	usr_conference_id	Required – conference ID.
	user_number	Required – User number of attendee to mute. See Note above.
Returned if success: <pre><response> <result>success</result> <result_description>Conference participant unmute successfully performed</result_description> <result_action>unmute</result_action> <usr_conference_id>113</usr_conference_id> <user_number>01</user_number> </response></pre>		

For example:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.attendee.unmute
  &usr_conference_id=3731
  &user_number=28971
```

will unmute an attendee with user_number 28971 in a Conference Smart Form with ID 3731. The response will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , result_action ,
usr_conference_id , user_number)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT result_action (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT user_number (#PCDATA)>
]>
```

```
<response>
  <result>success</result>
  <result_description>Smart Conference participant unmute successfully
  performed</result_description>
  <result_action>unmute</result_action>
  <usr_conference_id>3731</usr_conference_id>
  <user_number>28971</user_number>
</response>
```

You can kick someone off the call:

Method	Parameter	Value/Notes
.attendee.kick	api_key	Required – the account’s API key.
	usr_conference_id	Required – conference ID.
	user_number	Required – User number of attendee to mute. See note, above.

Returned if success:

```
<response>
  <result>success</result>
  <result_description>Conference participant kick successfully
  performed</result_description>
  <result_action>kick</result_action>
  <usr_conference_id>113</usr_conference_id>
  <user_number>01</user_number>
</response>
```

For example:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.attendee.unmute
  &usr_conference_id=3731
  &user_number=28971
```

will result in an attendee with ID 28971, in a conference call with ID 3731, being kicked off the call:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , result_action ,
usr_conference_id , user_number)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT result_action (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT user_number (#PCDATA)>
]>
```

```
<response>
  <result>success</result>
  <result_description>Smart Conference participant kick successfully
performed</result_description>
  <result_action>kick</result_action>
  <usr_conference_id>3731</usr_conference_id>
  <user_number>999</user_number>
</response>
```

This next web request lets you call an attendee and add them to the conference call; this is different than the "attendee.add" web request, because that one fails if you try to add the same person more than once. This web request allows you to call an attendee who is already on the list of participants.

Because this method is meant, in part, to re-connect attendees who may have disconnected by accident, the web request lets you specify that the call go to the actual phone number that the attendee used to reach the call (and not just the phone number originally entered for that attendee). For example, this might be useful if we enter the attendee's office number when we schedule the conference call, but the attendee actually joins the conference from a cell phone.

Method	Parameter	Value/Notes
.attendee.call	api_key	Required – the account’s API key.
	usr_conference_id	Required – conference ID.
	usr_conference_atten dee_id	Required – conference attendee to call. Value: ID.
	usr_conference_partici pation_id	Optional – Place call to phone number used by attendee. Value: ID. See Note below.

Returned if success:

```
<response>
  <result>success</result>
  <result_description>Conference participant call successfully
performed</result_description>
  <result_action>call</result_action>
  <usr_conference_id>113</usr_conference_id>
  <usr_conference_attendee_id >289</usr_conference_attendee_id>
  <usr_conference_participation_id></usr_conference_participation_id
>
</response>
```

NOTE: Attendees may call into a conference call from any phone. Responses to the "list" and "details" web requests include one or more "participation" elements which contain a "user_conference_participation_id" element. If that ID is sent, the call is placed to the phone number used by the attendee for in the conference.

An attendee may participate several times during a single conference, and each separate participation may come from a different telephone number. As such we need the ID of the participation.

In the example we used to illustrate the attendee.details web request, we saw that Harriet Jones had called into the conference from a different number than the one we'd entered when we added her to the list of attendees (+1 773 555 9999, as given in the caller_id element, instead of the +1 773 555 1212 we have listed in the phone_number element). To have the Conference Smart Form call her to put Harriet Jones back into the call -- she's listed as disconnected -- we send the conference information, her conference ID, and her conference participation ID.

To call Harriet Jones, we use:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.attendee.call
  &usr_conference_id=3731
  &usr_conference_attendee_id=12181
  &usr_conference_participation_id=8921
```

The response is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE response [
<!ELEMENT response (result , result_description , result_action ,
usr_conference_id , usr_conference_attendee_id ,
usr_conference_participation_id)>
<!ELEMENT result (#PCDATA)>
<!ELEMENT result_description (#PCDATA)>
<!ELEMENT result_action (#PCDATA)>
<!ELEMENT usr_conference_id (#PCDATA)>
<!ELEMENT usr_conference_attendee_id (#PCDATA)>
<!ELEMENT usr_conference_participation_id (#PCDATA)>
]>
<response>
  <result>success</result>
  <result_description>Smart Conference participant call successfully
performed</result_description>
  <result_action>call</result_action>
  <usr_conference_id>3731</usr_conference_id>
  <usr_conference_attendee_id>12181</usr_conference_attendee_id>
  <usr_conference_participation_id>8921</usr_conference_participation_
id>
</response>
```

Harriet Jones receives a call and is placed into the conference call.

Finally, we may just want to remove a scheduled Conference Smart Form entirely. This web request will not work when a conference is active.

Method	Parameter	Value/Notes
.remove	api_key	Required – the account’s API key.
	usr_conference_id	Required – conference ID.
Returned if success: <pre><response> <result>success</result> <result_description>The conference was successfully removed</result_description> <usr_conference_id>112</usr_conference_id> </response></pre>		

Example:

```
https://secure.ifbyphone.com/ibp_api.php
  ?api_key=foofoofoo
  &action=conference.remove
  &usr_conference_id=3691
```

This web request results in the removal of conference 3691, with output similar to that given in the table.

9. Appendix: Web Service Code Samples

This section includes examples of how to send web requests and receive results using server-side programming. We include examples from ASP, PHP, and Python.

9.1. ASP Example

```
<%@ Page Language="VB"%>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    Sub submit(ByVal sender As Object, ByVal e As EventArgs)
        Dim url As String

        url = "https://secure.ifbyphone.com/click_to_xyz.php?key=XXXX"
        url = url + "&phone_to_call=" + txtNPA.Text + txtNNX.Text +
txtLine.Text
        url = url + "&click_id=" + txtBuildingBlockID.Text

        Dim req As HttpWebRequest = WebRequest.Create(url)

        Try
            'Get the data as an HttpWebResponse object
            Dim resp As HttpWebResponse = req.GetResponse()

            Dim sr As New StreamReader(resp.GetResponseStream())
            Dim results As String = sr.ReadToEnd()
            sr.Close()

            lblCallStatus.Text = results
        Catch wex As WebException
            Response.Write("<font color=red>Error<br />Status: " &
wex.Status & "Message: " & wex.Message & "</font>")
        End Try
    End Sub
</script>

<html>
<body>
```

9.2. Example: Basic PHP Sample

The following is a simple example of triggering a click to call using cURL.

```
<?php

$phone="8475551234";
$click_id="XXX";

$url = "https://secure.ifbyphone.com/click_to_xyz.php?key=XXXX
      &phone_to_call=$phone&click_id=$click_id";

$session = curl_init($url);

// setup curl options
curl_setopt($session, CURLOPT_HEADER, false );
curl_setopt($session, CURLOPT_RETURNTRANSFER, true);

$result = curl_exec($session);
curl_close($session);

echo $result;

?>
```

9.3. Example: Advanced PHP Sample

To make programming with PHP easier, Ifbyphone provides two files at the Phone Mashup web site (www.phonemashup.com) in the package called "API Testing Scripts" These files are "utilities.php" and "webrequest.php." You can use these files to create very simple programs to send web requests.

Here's an example of how simple it is if you use the webrequest.php file. We'll ask for a list of all attendees for a Conference Smart Object with ID 3171. Note that the entire program has just 9 lines of code:

```
<?php
require ("utilities.php") ;
require ("webrequest.php") ;

// create web request object
$test = new webrequest( "secure.ifbyphone.com/ibp_api.php" ) ;

// add parameters
$test->addParameter("api_key","foofoofoo"); // "API key"
$test->addParameter("action", "conference.attendee.list") ;
$test->addParameter("usr_conference_id", 3171) ;

// send to Ifbyphone
$test->send() ;
```

```
// print out response from Smart Form
print "\n==response==\n" . $test->read() . "\n" ;
$test->close() ;
?>
```

Very straightforward, right? Let's review the contents of `webrequest.php` to see how it's done. We'll interpolate the explanations right into the text.

We start with the headers:

```
<?php
// Send web request (API call) to Ifbyphone.
// By default, use secure channels.

class webrequest {
    //a web request sends and receives data.
    // This is a REST transaction
    // we use POST exclusively to send the data
```

This first section contains a few variables:

```
    public $data = array() ;    // data to send with call
    public $url = "" ;
    public $remote = NULL ;
    public $service = "https://" ;
    public $encodedData = array() ;
    public $received = "" ;
    public $returnstring = TRUE;
    public $verifyPeer = FALSE ;
    public $verifyHost = FALSE ;

    // file-related
    public $filemode = "w" ;
    public $filehandle = NULL ;
```

This first function is invoked whenever we use "new." We collect the URL (server and web page) for the web request:

```
function __construct($url) {
    $this->url = $url ;
}
```

Rarely used, `nonsecure()` sets the system to use HTTP instead of HTTPS:

```
function nonsecure() {
    // set service to non-secure mode, http
    $this->service = "http://" ;
}
```

```
function secure() {
    // set service to HTTPS, i.e., secure
    $this->service = "https://" ;
}
```

In case we use FTP or some other service, we'd set this string. Again, not used in practice:

```
function setservice($service) {
    // set service to user-supplied string
    $this->service = $service ;
}

function setVerifyPeer($value) {
    // set SSL CA verification on or off
    $this->verifyPeer = $value ;
}

function setVerifyHost($value) {
    // set SSL Host verification on or off
    $this->verifyHost = $value ;
}
```

`addParameter()` is the workhorse function. Use `addParameter()` to send a parameter in the web request. While numbers will be transformed correctly, it's probably best to put quotes around large numbers, such as phone numbers.

You may add as many parameters as you need. To send a file from your server, use a "@" in front of the full path name of the file. (The URL of a file, on the other hand, is just an ordinary string.)

```
function addParameter( $name, $value) {
    // add a single item to list of data to send. Keep encoded
    format current.
    $this->data[$name] = $value ;
    $this->encodedData[rawurlencode($name)] =
    rawurlencode($value) ;
}
```

Ignored in ordinary practice, the next few functions let you get the response from the Smart Form as a file instead of as a string.

```
function setfilemode($mode) {
    // mode in which to open file, e.g., w, w+, etc.
    $this->filemode = $mode ;
}

function setfilename($name) {
    // data should be returned as file; used primarily for
    streams
    $this->returnstring = FALSE ;
    try {
```

```

        $this->filehandle = fopen($name, $this->filemode ) ;
    } catch (Exception $e) {
        print "Unable to open file:" . var_dump ($e) ;
        exit("Exiting...") ;
    }
}
function getfilehandle() {
    // for those who want to manipulate the data themselves
    return $this->filehandle ;
}

//
// Set up and then send web request
//

```

This next function sends all the information to the Smart Form. The function uses some CURL options to make certain that data are sent via POST, that the response from the Smart Form is collected, and so on:

```

function send() {
    // send data to far side, wait for response
    // send data to far end via POST

    $this->remote = curl_init ($this->service . $this->url) ;

    // set options, which include read back of far end
    curl_setopt($this->remote, CURLOPT_POST, TRUE ) ; // POST
    curl_setopt($this->remote, CURLOPT_RETURNTRANSFER, $this-
>returnstring) ; // returns string
    curl_setopt($this->remote, CURLOPT_FOLLOWLOCATION , TRUE) ;
    // follow redirects
    curl_setopt($this->remote, CURLOPT_HEADER, FALSE) ; //
results only in output
    curl_setopt($this->remote, CURLOPT_SSL_VERIFYPEER, $this-
>verifyPeer) ; // SSL: verify CA?
    curl_setopt($this->remote, CURLOPT_SSL_VERIFYHOST, $this-
>verifyHost) ; // SSL: verify peer?

    // if this is a file transfer
    if ($this->returnstring == FALSE) curl_setopt($this-
>remote, CURLOPT_FILE, $this->filehandle) ;

    // include data
    curl_setopt($this->remote, CURLOPT_POSTFIELDS, $this-
>data) ;

    $this->received = curl_exec ($this->remote) ;

    if (curl_errno($this->remote)) {
        print ("Error send/receive web request: number " .
curl_errno($this->remote) .

```

```

        ", explanation " . curl_error($this->remote) ) ;
    }
}

```

After the send is over, read the results as a string:

```

function read() {
    // read results from far end
    if ($this->returnstring == TRUE) {
        return $this->received;    // return the string
    } else {
        // nothing to return; it went into a file
        return NULL ;
    }
}

```

This function returns information about the HTTP transaction, useful for debugging:

```

function info() {
    // read any info from far end
    return strval(curl_getinfo($this->remote), 1) ;
}

function getURL() {
    return $this->service . $this->url ;
}

```

Function used internally to debug:

```

function dataEncoded() {
    // show the data we sent, used for debugging
    // make it look like a URL query
    // for cut/paste into documents
    $output = " ?" ;
    $items = count($this->encodedData ) ;
    $current = 0 ;
    foreach ( $this->encodedData as $key => $value ) {
        if ($current != 0 ) $output .= " &" ;
        $current += 1 ;
        $output .= "$key=$value\n" ;
    }
    return $output ;
}

```

More debugging. The raw parameters, as name/value pairs:

```

function dataRaw() {
    // show raw data we sent, used for debugging
    $output = "" ;
    $items = count($this->data ) ;
    $current = 0 ;

```

```

foreach ( $this->data as $key => $value ) {
    if ( $current != 0 ) $output .= " " ;
    $current += 1 ;
    $output .= "\"$key\" = \"$value\"\n" ;
}
return $output ;
}

```

For debugging, what the data would look like if they were URL-encoded instead of being sent via POST. (This is the function that generated all the examples you see in this document.)

```

function dataSent() {
    // show the data we sent, used for debugging
    // decoded
    $output = $this->dataEncoded() . "\n" ;
    foreach ( $this->encodedData as $key => $value ) {
        $k = "'" . rawurldecode($key) . "'" ;
        $output .= sprintf('%-22s', $k ) . "= \"".
            rawurldecode($value) . "\"\n" ;
    }
    return $output ;
}

```

This function closes the connection between your server and Ifbyphone's servers. It's not strictly necessary because the connection will close automatically when PHP ends, but it's a really good idea to explicitly close things as a matter of habit.

```

function close() {
    // finished
    try {
        curl_close($this->remote) ;
        if ($this->filehandle != NULL) fclose($this-
>filehandle) ;
    } catch (Exception $e) {
        // ignore
    }
}
}
?>

```

The file utilities.php includes a few "helper" functions that help format a bar-delimited list and print out times in the correct format. For example, formatTimestamp() prints out a time in the correct format for the "timestamp" parameter used by the Voice Broadcast Smart Form.

As a matter of good programming practice, we recommend that you maintain separate files to format dates and times, etc., rather than copying the code into your files. You'll find that separate files are far easier to maintain, to improve, and to keep up with any modifications. As an example of this in practice, when the

Conference API was in beta, the format of the "scheduled_time" changed. Because the function that formatted the time was in a single file, we made a change to the format in that single file -- instead of making a change in a dozen different files that all used the "scheduled_time" parameter.

9.4. Example: Python

These Python programs aren't really server-side programming; they can be run from the command line. Of course, Python is often used in server-side programming.

Python uses the same basic idea of a "utilities.py" and a "webrequest.py" file to keep all the helper functions in a separate module, and they're available for download at the Phone Mashup web site (www.phonemashup.com) inside the "API Testing Scripts" package.

Let's present the same example that we did for PHP: we'll ask for a list of all attendees for a Conference Smart Object with ID 3171. The entire program has only 9 lines of code, and appears remarkably similar:

```
import webrequest, utilities

# create web request object
URL = "secure.ifbyphone.com/ibp_api.php"
test = webrequest.webrequest( URL )

test.addParameter("api_key", "foofoofoo")
test.addParameter("action", "conference.attendee.list")
test.addParameter("usr_conference_id", 3171)

# send the web request
test.send()

# print response from Ifbyphone
print "\n==response==\n", test.read()

test.close()
```

Here are the contents of the Python webrequest.py file. If you use Python, you probably don't need much in the way of comments beyond the "__doc__"-accessible ones embedded in the file. And you probably can spot many ways to improve the webrequest class or replace it with a more versatile class.

Python's "urllib" methods to send a web request and receive the data also require far fewer options than the CURL package in PHP. In fact, "urllib" requires no options at all to use HTTPS/POST to connect securely to the Smart Forms.

```
''' send web requests to Ifbyphone, securely'''

import urllib, sys
```

```
class webrequest:
    ''' a web request sends and receives data. This is a REST
    transaction'''

    data = {} # dictionary of data to send with call
    url = ""
    fileobject = None
    service = "https://"
    encodedData = None

    def __init__(self, url):
        self.url = url

    def nonsecure(self):
        '''set service to non-secure mode, http'''
        self.service = "http://"

    def secure(self):
        '''set service to HTTPS, i.e., secure'''
        self.service = "https://"

    def setservice(self, service):
        '''set service to user-supplied string'''
        self.service = service

    def addParameter(self, name, value):
        ''' add a single item to list of data to send. Keep encoded
        format current.'''
        self.data[name] = value
        self.encodedData = urllib.urlencode(self.data)

    def send(self):
        ''' send data to far side, wait for response '''
        # send data to far end via POST
        try:
            self.fileobject = urllib.urlopen(self.service +
self.url, self.encodedData)
        except:
            # exceptions not handled in this code!
            print "error, exiting"
            self.fileobject.info()
            sys.exit()

    def read(self):
        '''read results from far end'''
        retval = ""
        temp = self.fileobject.read(1000)
        while temp.count('\n') > 1:
```

```
        retval += temp # what we have so far
        # try for more
        temp = self.fileobject.read(1000)
    return retval
    # return self.fileobject.read() # read() without size is
not entirely reliable

def info(self):
    '''read any info from far end'''
    return self.fileobject.info()

def fileno(self):
    '''give file descriptor for anyone who wants finer
control'''
    return self.fileobject.fileno()

def fileobject(self):
    '''file-like object for interested parties'''
    return self.fileobject

def dataSent(self):
    '''show the data we sent, used for debugging'''
    return str(self.encodedData)

def close(self):
    '''finished'''
    return self.fileobject.close()
```

10. Appendix: Ifbyphone Glossary

The purpose of the glossary is to assist you by defining commonly-used Ifbyphone terms.

Auto attendant

Another term for Interactive Voice Response (IVR). Essentially refers to an automated telephone answering system that routes calls.

Building Block

Ifbyphone proprietary code that allows you to build a complex voice application that is accessible via the Web, Email or telephone. Smart Click-to-Call and all destinations are building blocks.

Click-to-Call (CTC)

Immediately connect a user to a registered business telephone number. Click-to-Call can be easily set up by registering phone numbers—in addition to the toll free number given to you at sign up—within the Registered Numbers utility. Click-to-Call can then be configured as a destination for Smart Click-to-Call, enabling Web site visitors to click a button and immediately speak with a customer service representative or other call recipient.

Destinations

Destinations are Ifbyphone services that can be configured for Smart Click-to-Call access. Through Smart Click-to-Call, customers can click a link, type in their phone number, and connect immediately to a destination.

Destination Configurations

Destination Configurations allow you to setup a destination for use with a specific Smart Click-to-Call.

Interactive Voice Response (IVR)

A computerized system that allows a telephone caller to select options from a voice menu and interact with the computer phone system. IVR systems use Dual Tone, Multi-Frequency (DTMF) signals (entered from the telephone keypad) and natural language speech recognition to interpret the caller's response to IVR prompts.

Find Me

An Ifbyphone business service and destination, Find Me can be configured to maintain a prioritized list of phone numbers where a call recipient may be reached (e.g. desk or cell phone, pager, etc.). If a call recipient cannot be located at any of the phone numbers listed, the call can be redirected to their voice mailbox.

Modes

Blocks of time established within a specific day when configuring a schedule. Modes are fully customizable, but are commonly used to represent a workplace's open, closed, lunch, and after hours. Once modes have been set within a schedule, Smart Click-to-Call and Virtual Receptionist can be configured to exhibit different IVR behavior for a specific mode.

Net Integration

Communication between a SurVo and Web-based server. Net integration may be carried out to retrieve or post information and to perform a function like user account validation. Typically net integration is performed when a company is utilizing Ifbyphone business services to add voice to an existing system.

Schedule

An Ifbyphone tool that allows a user to specify the day-to-day operation of a workplace with start and finish times. Schedules are configured for Ifbyphone business services, including Smart Click-to-Call and Virtual Receptionist. Within a schedule, it is possible to specify a desired time zone, as well as modes.

Software as a Service (SaaS)

A software model employed by vendors who both develop and operate network-based applications for use by customers over the internet. SaaS is typically a low-cost way for businesses to obtain the same benefits of commercially licensed, internally-operated software, without the associated complexity and high initial cost.

Smart Click-to-Call

An Ifbyphone service that is highly flexible in customization and scalability, Smart Click-to-Call enables routed communication between customers, business services and contacts via the Web or Email. Through Smart Click-to-Call, customers can click a link, type in their phone number, and connect immediately to a company's configured services (commonly referred to as "destinations" in this documentation). Ifbyphone offers the following services as Smart Click-to-

Call destinations: a custom Virtual Receptionist, voice survey, voice mailbox, a Find Me auto-dial telephone list to locate an individual, and Click-to-Call for immediate telephony connection. Smart Click-to-Call can be configured based on the day and/or time of day.

SurVo

See Voice Survey Form (SurVo)

Virtual Receptionist

An Ifbyphone business service and destination, Virtual Receptionist can be configured to provide menus for callers to route incoming calls. An automated phone environment, the virtual receptionist can transfer callers to other numbers/extensions, forward callers to voicemail boxes, initiate a Find Me list or surveys, play pre-recorded messages, etc.

Voice Form

Similar to a Web form—with voice recognition capabilities, a voice form is a dialog between a user and the system configured to provide or collect information. At the completion of a voice form, the information collected can be emailed, saved to a web site or transmitted for further processing.

Voice Survey Form (SurVo)

An Ifbyphone business service and destination, Voice Survey Forms (SurVo) are used to create dialogs over the telephone between callers and the system, much like a Web form with voice recognition capabilities (see Voice Form). A SurVo can be used to conduct employment screens, provide a customer satisfaction survey, route a call based on a user's answers to a series of questions, etc. SurVos have the potential to be invoked in response to a Smart Click-to-Call, an inbound call, an outbound call, a scheduled call, or an API request from a Web site.

VoiceXML

W3C's standard XML format for specifying interactive voice dialogues between a human and a computer. VoiceXML allows voice applications to be developed and deployed for visual applications. Analogous to HTML interpretation by a visual web browser, VoiceXML documents are interpreted by a voice browser. VoiceXML has tags that instruct the voice browser to provide speech synthesis, automatic speech recognition, dialog management and sound file playback.

VUI (Voice User Integration)

Describes the interaction with computers through a voice and/or speech platform to initiate an automated service or process.

World Wide Web Consortium (W3C)

An international consortium, hosted by MIT, which develops interoperable technologies (standards, protocols, software and tools) for the Web. W3C's mission is to promote the evolution of the Web in a single direction, rather than into splintered factions.