In this chapter

- Module 3.1: Custom Script Design
- Module 3.2: Dialer Script Components
- Module 3.3: Scripter Client Tools

Objectives

At the end of this chapter you will be able to:

- Identify key components of successful script design
- Identify key components of an Interaction Dialer script
- Use Interaction Scripter client tools
Module 3.1: Custom Script Design

Topics in this module:

➢ Development Environments
➢ Software Development Process
➢ Six Steps to Scripter Success
➢ Design
➢ Call Flow Diagrams
➢ Flowchart
➢ Pseudocode
➢ Build/Code
➢ Test
➢ Promote Habits of Successful Programmers
Development Environments

Development
This is the environment in which scripts are designed and built. The development or modification of a script does not require a connection to an Dialer system. All the developer needs is a their favorite web development tool. The developer can make the script modifications and copy them to the test server. Now the script is ready to move into the testing environment.

Testing
This is the environment in which scripts and campaigns are tested. It should be as close to the production environment as possible. All of Interaction Dialer’s functionality should be available here. Every possible scenario of the script should be tested. This includes not only proper functionality but error handling also. Only after the script has been thoroughly tested should it be promoted to the production environment.

Production
The production environment is where the Interaction Dialer system interacts with agents and called parties. Be very cautious about promoting scripts to this environment because an improperly working script can have serious consequences on the productivity of the campaigns.

Be careful Make sure that the scripts have been thoroughly tested before loading them on the production server!

This is a cyclical process. As additional needs arise, scripts that are currently in the production environment may require additional customization and go back through the Development and Testing environments.
In order to be successful in script development, you must have a process to follow while working on a project.

The software industry utilizes several different methodologies for managing development projects, primarily divided into the headings of “Waterfall”, “Iterative”, and “Formal”. Waterfall methods have fallen into some disrepute as an Iterative model known as “Agile software development” becomes more popular (for example, Extreme Programming).

For developing scripts, Interactive Intelligence recommends use of a process which essentially combines Iterative and Waterfall methods by going through a series of steps on each iteration. We do not delineate the full set of Waterfall method steps (state requirements, analyze them, design a solution approach, architect a software framework for that solution, develop code, test, deploy, and maintain), but have trimmed them down to the few steps covered later in this chapter.

The key is having a defined process and following it for every project, instead of falling into the “Cowboy” method of programming - where the developers do whatever they think best, without planning, structure, review, or feedback until the “final” product is in the hands of the customer.

First, we will explore the various environments involved in the development process, followed by the process itself.
Six Steps to Scripter Success

This section offers an overview of the entire process of building and editing scripts. You should read this before you begin your script development or customization.

**Design**
Layout the pages you plan to build and establish the flow between them. Additionally, determine what functionality should be on each page and what data and call control the agents should have available to them. In this step, you will design the script on paper: This design may include each of the following forms:

- Call Flow Diagrams
- Flowcharts
- Pseudocode

**Build/Code**
Begin creating and linking steps to perform the new functionality. In this step, you use an HTML editor to build the page.

**Test**
In this step you run the workflow on your test system and ensure the script is functioning correctly.

**Debug (Re-Code)**
Coding is not a one-and-done process. It is a task that is repeated throughout the script development process. Do not forget to plan time for de-bugging and re-coding.

**Re-Test**
Just as coding is not a one-time process, testing is not either. Always retest scripts after debugging. This is a circular process that occasionally even requires going back to the (re)design stage.

**Promote**
In this step, you move the script from the test system to the production environment.
Design

Before diving into coding, every script should begin as a call flow diagram. This will define the flow of the campaign calls and provide direction while coding. The campaign administrator should provide most of this.

Flow from page to page is defined in a call flow diagram. Determine the order that pages should be presented. Also, decide whether certain pages should always be linked.

After determining the layout of each page, do not forget to include call controls that allow the agent to interact more directly with the contact. This includes call control features as well as contact attribute details.

Organize data into a well layed out, easy to use format. The script should provide smooth flow and easy access to pertinent information. The script should make the agent’s job easy.

Determine where/when to write data back to the contact list. This would typically be done at the end of the interaction, right before a reason code is applied.
Call Flow Diagrams

The first step in creating a well-designed Campaign Script is to diagram the call flow in a flowchart format. This should be a joint effort among the Call Center Management team, Web Development team and the Campaign Administrator. The most successful campaigns are thoroughly planned and documented in advance. The Call Center Manager should be able to provide the actual text script that agents will use to sell the product or interact with clients. This script should be broken down into “stages”, such as Open, Close, etc. This will allow the Web Developer to chart the navigation between pages and plan how many web pages will need to be designed.

If Stages have been created in the Dialer plug-in in Interaction Administrator for this campaign, the relationship between each stage and the appropriate web page must be defined. Detailing the relationship between stages and specific web pages allows the Web Developer to integrate staging into the design of the web pages.

A major element of script design is determining how the agents will move among different stages in the scripts. It is this navigation that triggers the Predictive Dialer algorithm, and thus ensures that the campaign is moving at a correct pace. The more detailed the flowchart, the more successful the script implementation (and campaign) will be when developed. When charting the call flow, make sure to remember some basic design considerations:

- There must be an “idle” index page that agents view when waiting to receive a call from the Dialer system. This will place agents into an available status. Throughout this document, this page will be referred to as the call-waiting page.
- There should be a page that allows the agents to greet the callers. This can incorporate active elements which allow the page to read from the CallList and display the callers’ name and address.
- A campaign may have “static” pages that describe features, meet caller objections, or display an FAQ list. These pages would have HTML text and some navigation features, but may have no active scripter components.
- Reference each page back to a preset stage in Campaign Administrator. Staging will be implemented in the design of the page using Scripter commands.
- Most campaign pages will need to be able to navigate to multiple other pages. For example: agents will need to be able to go directly from the opening page to the features page, the objections page, or the close page. Allow for multidirectional navigation in your flowchart.
- It is possible to implement pages that allow your agents to logoff the campaign for breaks, put the call on hold, and perform other basic call actions from within Interaction Scripter. If this type of interaction with IC is desired, it will use Interaction Scripter and should be noted in the call flow.
The call flow diagram should optimally be done in flowchart form using a program such as Visio. A very basic call flow done in Microsoft PowerPoint could look like the one shown below. This flowchart does not fully implement multi-directional navigation or advanced interactions with Interaction Dialer. We will be using this simple example as our starting point for a detailed discussion of script design and in the early scripter labs.
Before you begin building your new script, always map out the new functionality on a whiteboard or on paper using a flowchart or pseudocode. Try to think in terms of logic and reusable functions. Remember that creating scripts is programming; planning at the outset saves time when you start building the script.

A flowchart is a graphic or diagram which shows how a complex operation takes place. The flowchart breaks that operation down into its smallest, and easiest-to-understand parts. The table below lists some of the more common flowcharting symbols and their purposes. There are many more flowcharting symbols to represent additional functions such as, database operations, printing, calling a subroutine, etc.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Process Symbol" /></td>
<td>Process - represents a process such as, play a prompt, transfer a call, disconnect a call, etc.</td>
</tr>
<tr>
<td><img src="image2" alt="Connector Symbol" /></td>
<td>Connector - used at the intersection of two or more lines.</td>
</tr>
<tr>
<td><img src="image3" alt="Data Symbol" /></td>
<td>Data - represents an input/output operation, such as getting entered digits.</td>
</tr>
<tr>
<td><img src="image4" alt="Decision Symbol" /></td>
<td>Decision - used to evaluate a boolean expression. This symbol has two branches from it, a true path and a false path.</td>
</tr>
<tr>
<td><img src="image5" alt="Manual Operation Symbol" /></td>
<td>Manual operation - represents an activity that requires human intervention, such as picking up a handset.</td>
</tr>
<tr>
<td><img src="image6" alt="Off-Page Connector Symbol" /></td>
<td>Off-page connector - used when the flowchart spans more than a single page. An integer or letter is assigned to the off-page connector and its associated off-page connector on another page to indicate where the logic continues.</td>
</tr>
<tr>
<td><img src="image7" alt="Terminator Symbol" /></td>
<td>Terminator - indicates the end of the logic for this particular program. If this is a function, then control will return to the calling code.</td>
</tr>
</tbody>
</table>

©2008, Interactive Intelligence, Inc.
Pseudocode  After designing a script, but before actually coding the pages, it is a good idea to first describe the page in pseudocode. Pseudocode consists of plain english details of functions and logic without any regard toward syntax. Programmers use pseudocode to “talk out” the logic before attempting to write the program.

The advantage of pseudocode is that it allows the developer quickly jot down details without getting hung up on the specifics of any programming language. A pseudocoded page should reflect all logic and actions of functions. In this manner, anybody should be able to read the pseudocode and know what will happen and the process by which it will happen.
Build/Code

Once you’ve mapped out the functionality and decided to implement the logic, you’re ready to start modifying or building the script.

Turn the script into usable pages by translating pseudocode into javascript. At this point in time, all functions and statements should be valid javascript code. Insert Scripter API calls and peripheral functions as appropriate. Functions have effectively been defined already. This step is just a matter of turning your logic into executable code.

Do not blindly delete the pseudocode. It will likely make effective comments. Mark this pseudocode as comments to describe the actions within each function.

Always include plenty of comments in code. Far too often, a developer will return to code later, whether after days, months, or years, and not recall the intentions of certain segments. Comments provide an opportunity to describe those intentions. Especially when coding workarounds or atypical processes, include comments to detail why code might be written the way it is. This will save you and your colleagues from many headaches.

Coding should actually be one of the quickest phases of script development.
Testing is an all too often underestimated step in script development. Provide plenty of time to test scripts and ensure they behave as expected.

Validate page flow, call control, and database updates. Use Interaction Supervisor to validate reason/finish codes and stages. Make certain that no agent can attempt to take another call before the current call is dispositioned.

A development Dialer/IC server is always recommended. Like other critical components, scripts should always be tested through a test campaign, preferably on a test server, prior to going live.

Ideally, new scripts will first be introduced on a test server. In reality, actual call center agents often provide the best testbed for new scripts. The logic of a developer might differ from the logic of an agent. As practical, rollout a new script to a limited number of agents until confident in its functionality.

Recommended method for limited script rollout:

■ Copy the following components:
  ➢ Workflow
  ➢ Campaign
  ➢ Workgroup

■ Assign limited number of agents to limited release workgroup

After testing, correct errors and retest. If changes are made to the design of a page, be certain to update call flow diagrams so that documentation will remain up to date.

All scripts should be tested before being released into production.

Be careful This is a critical step and must not be overlooked.
Promote Only after the script has been thoroughly tested, should it be rolled out on the production campaigns. Following proper project lifecycle management will help ensure maximum productivity and minimal downtime.
Habits of Successful Programmers

- Always backup before making changes
- Always test your changes before applying them to a production server. Verify that pages function even if incorrect data or values are entered. All changes are significant. Even a small change can bring a campaign down.
- Reuse code across projects. Avoid naming js include files, variables, and other items after clients or projects so that you can easily reuse the code for other clients or projects. This is especially true with functions.
- Follow good project management practices. Plan out scripts using call flow diagrams, created using Visio or a similar application. Make sure all paths for all options in the call flow end with specific requirements. Make sure the diagrams are complete, including even minor details. Get a sign off on the call flow scope of work before starting.
- Document! Document! Document!

Best Practice

Project documentation that has been signed by both the script developer and the campaign administrator serves to protect both parties. From the script developer perspective, this is something to refer to when developing to be certain that requirements are understood and later to serve as validation that requirements are delivered.
Module 3.2: Dialer Script Components

Topics in this module:

➢ Scripting and Staging
➢ Dispositions
Correct implementation of staging is absolutely vital to creating a script that maximizes the potential of the Predictive Dialer algorithm. In previous chapters, we looked at how to create stages in the Dialer plug-in for Interaction Administrator. These stages are used in Scripter pages to tell Interaction Dialer that an agent has moved from one phase of a call to another phase. The Interaction Scripter command that moves a call to the next stage is IS_Action_Stage. Without this command in the script pages, the Predictive Dialer will be unable to effectively predict when an agent will be available for a new call. Normally, IS_Action_Stage is included as part of your web page navigation. For example when an agent clicks on a navigation button to move to the “Close” page in the Interaction Scripter Client, IS_Action_Stage would tell Interaction Dialer to move that agent and that call to the “Close” stage.

Stages should be determined before charting the call flow and should be implemented by the Campaign Administrator. Staging plays a direct role in the effectiveness of campaign pacing and, when set up correctly, can greatly increase the success of a campaign. When troubleshooting problems with pace (abandoned calls, idle agents), check the staging to make sure that stages have been correctly defined for the current campaign and script.

Stages should be assigned automatically when a page loads.

Be careful Only terminal stages should be defined as predictive.
Dispositions

Scripts provide an opportunity to assign reason codes and finish codes to interactions. Finish codes should be descriptive so that campaign managers can get detailed information from their Dialer reports.

As a final step for any campaign call, the script should assign a reason code and finish code to an interaction. This can be done through a button that the agent will push or even assigned automatically when a certain page is loaded. Assigning a reason code indicates to Dialer that the current call record is complete and should be released.

In addition to assigning reason/finish codes, be careful to avoid ambiguous entries. Ambiguous entries indicate that Interaction Dialer does not know the result of a call.

Finish Code Implementation — Finish codes are user-defined strings that indicate the completion status of a campaign call. Finish codes add granularity to statistics and reports, allowing you to see not just that the call failed, but why the call failed, for example. If finish codes are not defined, an appropriate finish code will be assigned by Interaction Dialer.

Let’s look at an example: If a call fails, you might have the agent choose a reason for the failure from a list. In the script, each choice would log a different finish code, such as, “Doesn’t have a computer” or “Already has Internet service” for the reason code of “Failure”.

Ambiguous Entries — If no return code is returned by the client application, the reason code and finish code will be assigned by Dialer as “Ambiguous”. This could happen if:

- The script is written in such a way that the agent is placed in an available status before the call is dispositioned.
- The agent exits out of the client application improperly (the agent doesn’t logout, but clicks on the close button).
- Interaction Scripter crashes.

Be careful

After a reason code is assigned, the call record is released. This means that Scripter commands will no longer apply to that record. It will be impossible to use IS_Attr commands to retrieve or update call list attributes.
Module 3.3: Scripter Client Tools

Topics in this module:

➢ Script Debugger
➢ Show Ambiguous
Interaction Scripter offers a debugging feature that helps Web Page Designers detect and resolve problems with custom campaign scripts. The debugger is available only when Interaction Scripter is started with a /debug command-line parameter. The syntax is:

```
interactionscripter.exe /debug
```

When the debugger is active, you can open the Debug Dialog by selecting Show Dialog from the Debug menu:

Debug mode analyzes events in the current session to identify potential problems with scripts. When Interaction Scripter executes a script it looks for common types of script errors. When a problem is detected, Scripter sends messages to its debugger. These messages are analogous to warning and error messages that a compiler might generate. Interaction Scripter is not a compiler, however. The debug window opens automatically when scenarios like these are detected:

- An Agent sets status to Available (in the script) without dispositioning the current call, or the Agent's Interaction Client status changes to Available while the Agent is on a call.
- An Agent receives datapop when a disposition is pending.
- A page unload event is called without a corresponding onload within 5 seconds.
- Multiple page elements are assigned to the same 'name' attribute.
- Scripter detects COM errors related to invalid Call Ids, Conference Ids etc.
- The user is warned if SetAvailability calls are made when on a call or in 'Awaiting Callback' status.
- If the Agent's Workgroup activation changes, the user is warned that the Agent will no longer receive ACD calls.
- The user is warned if the call was not been dispositioned prior to transfer.
The debug dialog error list displays the time of the error, the script page that contains the error, and the error message. The script URL is displayed as well. The four tabs that are displayed are defined below:

**Error** — Displays the text of the selected error message. Since the window is not resizable, it is much easier to view error messages here than in the error list.

**Suggestion** — May display suggestions for resolving the error.

**Example** — May contain a script snippet that shows how to implement the suggestion. Text on this page can be copied by right clicking on the selected text and choosing copy from the context menu that appears.

**Stack** — May provide contextual information regarding a situation. For example, if the script attempts to disconnect when there is no active call object, that context would be described on the stack page.
Show Ambiguous  Ambiguous reason codes devalue reports because they do not provide accurate information regarding the result of a call. For this reason, ambiguous reason codes should be eliminated. Most ambiguous reason codes arise from poor scripting.

Interaction Scripter provides a command line switch to notify the agent, typically during testing of a script, that an ambiguous entry has been logged. Simply use the following command line parameter when running Interaction Scripter client:

/showambiguous

Use showambiguous to display an error dialog when calls are not dispositioned. When adding this command line option to Interaction Scripter, the format is a forward slash followed by the word showambiguous. With this option a dialog is presented to the user that indicates that the user has gone to an available status without dispositioning the call they were on and this resulted in an ambiguous result. This option is useful to use in conjunction with the debug command line option.
Labs
There are no labs associated with this chapter.

Summary
In this chapter we discussed:

➢ Development Environments
➢ Software Development Process
➢ Six Steps to Scripter Success
➢ Design
➢ Call Flow Diagrams
➢ Flowchart
➢ Pseudocode
➢ Build/Code
➢ Test
➢ Promote Habits of Successful Programmers
➢ Scripting and Staging
➢ Dispositions
➢ Script Debugger
➢ Show Ambiguous