

Wählplan-Grundlagen

*Everything should be made as simple as possible,
but not simpler.*

Albert Einstein (1879–1955)

Der Wählplan ist wirklich das Herz jedes Asterisk-Systems, da er definiert, wie Asterisk ein- und ausgehende Anrufe verarbeitet. Kurz zusammengefasst, lässt sich sagen, dass er aus einer Liste von Anweisungen oder Schritten besteht, denen Asterisk folgt. Im Gegensatz zu traditionellen Telefonsystemen ist der Asterisk-Wählplan vollständig konfigurierbar. Um Ihr eigenes Asterisk-System erfolgreich einrichten zu können, müssen Sie daher den Wählplan verstehen.

Machen Sie sich keine Sorgen, wenn sich das Schreiben eines Wählplans beängstigend anhört. Dieses Kapitel beschreibt Schritt für Schritt, wie ein Wählplan funktioniert, und bringt Ihnen alles bei, was Sie wissen müssen, um einen eigenen Wählplan einzurichten. Die Beispiele bauen aufeinander auf, d.h., Sie können zurückspringen und einen Abschnitt erneut lesen, wenn Ihnen etwas unklar ist. Beachten Sie aber auch, dass dieses Kapitel nicht umfassend alle Dinge erläutert, die man mit einem Wählplan machen kann. Wir wollen hier nur die Grundlagen vermitteln. Fortgeschrittene Aspekte des Wählplans werden in späteren Kapiteln behandelt.

Wählplan-Syntax

Der Asterisk-Wählplan wird in einer Datei namens *extensions.conf* definiert.



Die Datei *extensions.conf* liegt normalerweise im Verzeichnis */etc/asterisk/*, aber das hängt davon ab, wie Asterisk installiert wurde. Weitere gängige Speicherorte für diese Datei sind */usr/local/asterisk/etc/* und */opt/asterisk/etc/*.

Der Wählplan besteht aus vier Hauptteilen: Kontexten, Nebenstellen, Prioritäten und Anwendungen. In den nächsten Abschnitten behandeln wir jeden dieser Teile und erläutern, wie sie beim Aufbau eines Wählplans zusammenspielen. Nachdem wir erklärt haben, welche Rolle die jeweiligen Elemente im Wählplan spielen, gehen wir mit Ihnen den Aufbau eines grundlegenden, funktionierenden Wählplans durch.

Beispiel-Konfigurationsdateien

Wenn Sie bei der Installation von Asterisk auch die Beispiel-Konfigurationsdateien installiert haben, dann gibt es sehr wahrscheinlich bereits eine *extensions.conf*-Datei. Statt mit der Beispieldatei zu beginnen, empfehlen wir Ihnen, die *extensions.conf* von Grund auf neu aufzubauen. Das ist von Vorteil, weil es Ihnen ein besseres Verständnis der Wählplan-Konzepte und -Grundlagen vermittelt.

Dennoch wollen wir zugeben, dass die *extensions.conf*-Beispieldatei eine fantastische Ressource ist, voller Beispiele und Ideen, die Sie nutzen können, nachdem Sie die grundlegenden Konzepte verstanden haben. Wir empfehlen Ihnen, die Datei in so etwas wie *extensions.conf.sample* umzubenennen. Auf diese Weise können Sie sich die Datei bei Bedarf in Ruhe ansehen. Sie finden die Beispiel-Konfigurationsdateien auch im */configs/*-Verzeichnis der Asterisk-Quellen.

Kontexte

Wählpläne sind in einzelne Abschnitte unterteilt, die als *Kontexte* bezeichnet werden. Kontexte sind benannte Gruppen von Nebenstellen. Einfach ausgedrückt, regeln sie die Kommunikation verschiedener Teile des Wählplans untereinander. Eine in einem Kontext definierte Nebenstelle ist von allen Nebenstellen in anderen Kontexten isoliert, bis eine Interaktion explizit erlaubt wird. (Wie man eine Interaktion zwischen Kontexten erlaubt, behandeln wir gegen Ende dieses Kapitels.)

Nehmen wir als einfaches Beispiel zwei Unternehmen, die sich einen Asterisk-Server teilen. Wenn wir das Voice-Menü jedes Unternehmens in einem eigenen Kontext platzieren, sind diese effektiv voneinander getrennt. Dadurch können wir unabhängig definieren, was passiert, wenn zum Beispiel die Nebenstelle 0 gewählt wird: Die Leute, die die 0 im Voice-Menü von Unternehmen A drücken, erhalten die Rezeption von Unternehmen A, während Anrufer, die die 0 im Voice-Menü von Unternehmen B drücken, mit der Rezeption von Unternehmen B verbunden werden. (Dieses Beispiel setzt natürlich voraus, dass Sie Asterisk angewiesen haben, Anrufe an die Rezeption weiterzuleiten, wenn der Anrufer die 0 drückt.)

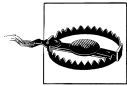
Kontexte werden definiert, indem man den Namen des Kontextes mit eckigen Klammern ([]) umgibt. Der Name kann aus den Buchstaben A bis Z (Groß- und Kleinbuchstaben),

den Ziffern 0 bis 9 sowie dem Bindestrich und dem Unterstrich bestehen.¹ Ein Kontext für eingehende Anrufe könnte zum Beispiel so aussehen:

[incoming]

Alle Anweisungen, die auf die Definition eines Kontexts folgen, sind Teil dieses Kontexts, bis der nächste Kontext definiert wird. Am Anfang des Wählplans stehen zwei spezielle Kontexte namens [general] und [globals]. Wir besprechen den [globals]-Kontext später in diesem Kapitel, momentan reicht es aus, wenn Sie wissen, dass diese beiden Kontexte sehr speziell sind.

Eine der wichtigsten Aufgaben von Kontexten ist die Sicherheit. Indem Sie Kontexte korrekt einsetzen, können Sie bestimmten Teilnehmern den Zugriff auf bestimmte Features gewähren (etwa Ferngespräche), die anderen Teilnehmern vorenthalten bleiben. Wenn Sie Ihren Wählplan nicht sorgfältig entwerfen, könnten Sie es den Teilnehmern versehentlich ermöglichen, Ihr System für betrügerische Zwecke zu missbrauchen. Bitte denken Sie daran, wenn Sie Ihr Asterisk-System einrichten.



Die Asterisk-Quellen enthalten eine sehr wichtige Datei namens *SECURITY*, die verschiedene Schritte beschreibt, die Sie ausführen sollten, um Ihr Asterisk-System abzusichern. Es ist äußerst wichtig, dass Sie diese Datei lesen und verstehen. Wenn Sie die darin beschriebenen Sicherheitsvorkehrungen ignorieren, könnten Sie es allen und jedem erlauben, auf Ihre Kosten Fern- und Auslandsgespräche zu führen!

Wenn Sie die Sicherheit Ihres Asterisk-Systems nicht ernst nehmen, könnten Sie das teuer bezahlen – im wahrsten Sinne des Wortes! *Bitte* nehmen Sie Zeit und Mühe in Kauf, um Ihr System vor betrügerischen Teilnehmern zu schützen.

Extensions

Innerhalb jedes Kontexts definieren wir eine oder mehrere Extensions. Eine *Extension* ist eine von Asterisk befolgte Anweisung, die durch einen eingehenden Anruf oder die auf einem Kanal gewählten Ziffern ausgelöst wird. Extensions legen fest, was mit Anrufen passiert, während sie sich ihren Weg durch den Wählplan bahnen. Zwar können Extensions genutzt werden, um Durchwahlen im traditionellen Sinn festzulegen (z. B. erreichen Sie John unter der Durchwahl 153), aber sie können bei Asterisk für wesentlich mehr eingesetzt werden.

Die Syntax für eine Extension verwendet das Wort *exten*, gefolgt von einem Pfeil, der aus dem Gleichheitszeichen und dem Größer-als-Zeichen zusammengesetzt wird:

exten =>

¹ Bitte beachten Sie, dass das Leerzeichen bewusst aus der Liste erlaubter Zeichen ausgenommen wurde. Verwenden Sie keine Leerzeichen in Ihren Kontextnamen – die Ergebnisse würden Ihnen nicht gefallen!

Diesem folgt der Name der Extension. Wenn man mit Telefonsystemen arbeitet, denkt man bei Nebenstellen an Nummern, die man wählt, um andere Teilnehmer zu erreichen. Bei Asterisk geht es um sehr viel mehr – so können die Namen für Extensions beispielsweise eine beliebige Kombination aus Ziffern und Buchstaben sein. Im Verlauf dieses und des nächsten Kapitels werden Sie sowohl numerische als auch alphanumerische Extensions verwenden.



Die Zuweisung von Namen an Nebenstellen mag wie ein revolutionäres Konzept erscheinen, aber wenn man bedenkt, dass viele Voice-over-IP-Transporte das Wählen mittels Name oder E-Mail-Adresse unterstützen (oder sogar aktiv fördern), dann macht die Sache Sinn. Dies ist eines der Features, die Asterisk so flexibel und leistungsfähig machen.

Eine vollständige Extension besteht aus drei Komponenten:

- Der Name (oder die Nummer) der Extension
- Die Priorität (jede Extension kann mehrere Schritte umfassen; die Nummer eines Schrittes wird als »Priorität« bezeichnet)
- Die Anwendung (der Befehl), die irgendetwas mit dem Anruf macht

Diese drei Komponenten werden durch Kommas voneinander getrennt:

```
exten => name,priorität,anwendung()
```

Hier ein einfaches Beispiel, wie eine echte Extension aussehen könnte:

```
exten => 123,1,Answer()
```

In diesem Beispiel hat die Extension den Namen 123 und die Priorität 1, und die Anwendung ist `Answer()`. Gehen wir nun einen Schritt weiter und sehen uns Prioritäten und Anwendungen an.

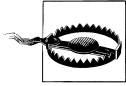
Prioritäten

Jede Extension kann aus mehreren Schritten zusammengesetzt sein, die als *Prioritäten* bezeichnet werden. Jede Priorität wird fortlaufend durchnummeriert, beginnend bei 1. (Tatsächlich gibt es bei dieser Regel eine Ausnahme, die im Kasten »Nichtnummerierte Prioritäten« behandelt wird.) Jede Priorität führt eine bestimmte Anwendung aus. Zum Beispiel würde die folgende Extension den Anruf beantworten (mit Priorität 1) und dann auflegen (Priorität 2):

```
exten => 123,1,Answer()
```

```
exten => 123,2,Hangup()
```

Machen Sie sich keine Sorgen, wenn Sie nicht verstehen, was `Answer()` und `Hangup()` machen – wir behandeln sie in Kürze. Der wesentliche Punkt ist hier, dass Asterisk für eine bestimmte Extension die Prioritäten in numerischer Reihenfolge abarbeitet.



Sie müssen sicherstellen, dass die Prioritäten bei 1 beginnen und dann fortlaufend durchnummeriert werden. Wenn Sie eine Priorität auslassen, macht Asterisk nicht weiter. Wenn Sie feststellen, dass Asterisk nicht alle Prioritäten einer Extension abarbeitet, dann müssen Sie sicherstellen, dass keine Prioritäten ausgelassen oder falsch nummeriert wurden.

Nichtnummerierte Prioritäten

Es ist nicht gerade schön, wenn wir Ihnen sagen, dass die Prioritäten fortlaufend nummeriert werden müssen, um uns dann im nächsten Satz zu widersprechen. Nun, es muss wohl so sein.

Die Version 1.2 von Asterisk hat eine neue Variante der Nummerierung von Prioritäten eingeführt, die *n*-Priorität, was für »nächste« Priorität steht. Sobald Asterisk eine Priorität namens *n* erkennt, nimmt es die Nummer der vorangegangenen Priorität und erhöht sie um 1. Das macht Änderungen an Ihrem Wahlplan einfacher, weil Sie nicht alle Schritte neu durchnummerieren müssen. Zum Beispiel könnte Ihr Wahlplan wie folgt aussehen:

```
exten => 123,1,Answer()  
exten => 123,n,mach was  
exten => 123,n,mach was anderes  
exten => 123,n,eine letzte Sache noch  
exten => 123,n,Hangup()
```

Die Version 1.2 gestattet es Ihnen auch, Prioritäten mit Textlabeln zu versehen. Um ein Textlabel an eine Priorität zuzuweisen, geben Sie das Label in runden Klammern hinter der Priorität ein:

```
exten => 123,n(label),mach was
```

Im nächsten Kapitel beschreiben wir, wie man basierend auf der Wahlplan-Logik zu verschiedenen Prioritäten springen kann.

Anwendungen

Anwendungen sind die Arbeitspferde eines Wahlplans. Jede Anwendung führt eine bestimmte Aktion auf dem aktuellen Kanal aus, etwa das Abspielen eines Sounds, die Verarbeitung von Eingaben oder das Beenden der Verbindung. Im vorigen Beispiel haben wir zwei einfache Anwendungen eingeführt: `Answer()` und `Hangup()`. Wie diese funktionieren, werden Sie gleich erfahren.

Einige Anwendungen, wie zum Beispiel `Answer()` und `Hangup()`, benötigen keine weiteren Anweisungen, um ihre Aufgabe erledigen zu können. Andere Anwendungen benötigen zusätzliche Informationen. Diese zusätzlichen Informationen, die man als *Argumente* bezeichnet, können an die Anwendungen übergeben werden, um die Ausführung der Aktion zu beeinflussen. Sie übergeben die Argumente, die durch Kommas voneinander getrennt werden, in den runden Klammern direkt hinter dem Namen der Anwendung.



Gelegentlich sieht man auch das Pipe-Zeichen (|) anstelle des Kommas als Trennsymbol zwischen Argumenten. Suchen Sie sich das von Ihnen bevorzugte Zeichen aus. Bei den Beispielen in diesem Buch verwenden wir aber Kommas, um die Argumente einer Anwendung zu trennen.

Während wir im nächsten Abschnitt unseren ersten Wählplan erstellen, werden Sie lernen, Anwendungen (und deren Argumente) zu Ihrem Vorteil zu nutzen.

Ein einfacher Wählplan

Nun sind wir so weit, unseren ersten Wählplan zu entwerfen. Wir beginnen mit einem sehr einfachen Beispiel. Wir entwerfen den Wählplan so, dass Asterisk eingehende Anrufe annimmt, eine Sounddatei abspielt und dann auflägt. Wir nutzen dieses einfache Beispiel, um Ihnen die wichtigsten Grundlagen eines Wählplans zu vermitteln.

Damit die Beispiele in diesem Kapitel korrekt laufen, setzen wir voraus, dass zumindest ein Zap-Kanal angelegt und konfiguriert wurde (wie im vorigen Kapitel besprochen) und dass alle eingehenden Anrufe an den [incoming]-Kontext weitergeleitet werden. Wenn Sie andere Arten von Kanälen nutzen, müssen Sie die Beispiele an Ihre jeweiligen Gegebenheiten anpassen.

Die s-Extension

Bevor wir mit unserem Wählplan anfangen, müssen wir eine spezielle Extension namens s erläutern. Wenn ein Anruf in einen Kontext ohne bestimmte Ziel-Extension eingeht (z. B. eine klingelnde FXO-Leitung), wird er automatisch von der s-Extension verarbeitet. (Das s steht für »Start«, da die meisten Anrufe in der s-Extension beginnen.) Da das genau das ist, was wir für unseren Wählplan brauchen, wollen wir damit beginnen, die einzelnen Teile zusammensetzen. Wir führen drei Aktionen für einen Anruf aus (annehmen, Sounddatei abspielen, auflägen), d. h., wir müssen eine Extension namens s mit drei Prioritäten anlegen. Wir platzieren alle drei Prioritäten innerhalb von [incoming], da alle eingehenden Anrufe in diesem Kontext beginnen müssen:

```
[incoming]
exten => s,1,anwendung()
exten => s,2,anwendung()
exten => s,3,anwendung()
```

Jetzt müssen wir nur noch die jeweiligen Anwendungen einfügen, und schon haben wir unseren ersten Wählplan angelegt.

Die Anwendungen `Answer()`, `Playback()` und `Hangup()`

Wenn wir einen Anruf annehmen, eine Sounddatei abspielen und dann wieder auflegen wollen, müssen wir lernen, wie man das genau macht. Die `Answer()`-Anwendung wird verwendet, um einen klingelnden Kanal anzunehmen. Sie übernimmt die Rufannahme für den Kanal, der den eingehenden Anruf empfängt. (Bei einigen wenigen Anwendungen ist eine Rufannahme nicht nötig, aber die ordentliche Annahme eines Anrufs vor irgendwelchen Aktionen ist eine sehr vernünftige Angewohnheit.) Wie bereits früher erwähnt, verlangt `Answer()` keine Argumente.

Die `Playback()`-Anwendung wird verwendet, um eine bereits aufgenommene Sounddatei über einen Kanal abzuspielen. Wenn Sie die `Playback()`-Anwendung nutzen, werden Eingaben vom Benutzer einfach ignoriert.



Asterisk wird mit vielen professionell aufgenommenen Sounddateien ausgeliefert, die im Standard-sounds-Verzeichnis (üblicherweise `/var/lib/asterisk/sounds/`) zu finden sind. Sie wurden im GSM-Format aufgezeichnet und tragen daher die Datei-Endung `.gsm`. Wir nutzen diese Dateien in vielen unserer Beispiele. Einige der in unseren Beispielen verwendeten Dateien stammen aus dem `asterisk-sounds`-Modul. Nehmen Sie sich also die Zeit, dieses Modul zu installieren (siehe Kapitel 3).

Um `Playback()` zu verwenden, übergeben Sie einen Dateinamen (ohne Endung) als Argument. Zum Beispiel würde `Playback(dateiname)` die Sounddatei namens `dateiname.gsm` aus dem Standard-sounds-Verzeichnis abspielen. Sie können bei Bedarf auch den vollständigen Pfad auf die Datei angeben:

```
Playback(/home/john/sounds/dateiname)
```

Dieses Beispiel würde `dateiname.gsm` aus dem Verzeichnis `/home/john/sounds/` abspielen. Sie können auch relativ zum Asterisk-sounds-Verzeichnis liegende Pfade verwenden:

```
Playback(custom/filename)
```

Dieses Beispiel würde `dateiname.gsm` aus dem Unterverzeichnis `custom/` des Standard-sounds-Verzeichnisses abspielen. Enthält das angegebene Verzeichnis mehr als eine Datei mit diesem Dateinamen, aber mit unterschiedlichen Endungen, dann spielt Asterisk automatisch die »beste« Datei ab.²

Die `Hangup()`-Anwendung macht genau das, was ihr Name vermuten lässt: Sie legt den aktiven Kanal auf. Der Anrufer erhält einen Hinweis (die so genannte Indication), dass die Gegenseite aufgelegt hat. Sie verwenden diese Anwendung am Ende eines Kontexts,

² Asterisk wählt die beste Datei basierend auf den Übersetzungskosten aus, d.h., es wählt die Datei, die bei der Umwandlung in ihr natives Audioformat am wenigsten CPU-intensiv ist. Beim Start von Asterisk berechnet es die Übersetzungskosten für die verschiedenen Audioformate (die von System zu System variieren können). Sie können sich die Übersetzungskosten ansehen, indem Sie `show translation` auf der Asterisk-Kommandozeile eingeben. Wir behandeln die verschiedenen Audioformate (*Codecs*) in Kapitel 8.

wenn Sie den aktuellen Anruf beenden wollen, um sicherzustellen, dass die Anrufer dem Wählplan nicht weiter folgen. Diese Anwendung benötigt keine Argumente.

Unser erster Wählplan

Nachdem wir nun unsere Extension angelegt, drei verschiedene Prioritäten vergeben und die benötigten Anwendungen kennen gelernt haben, wollen wir diese Teile zu unserem ersten Wählplan zusammenfügen. Wie für technische Bücher üblich (insbesondere für Computer-Bücher), heißt unser erstes Beispiel »Hello World!«.

Die erste Priorität unserer Extension nimmt den Anruf entgegen. Die zweite spielt eine Sounddatei namens *hello-world.gsm* und die dritte beendet den Anruf. Unser Wählplan sieht wie folgt aus:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Playback(hello-world)
exten => s,3,Hangup()
```

Wenn Sie ein oder zwei Kanäle konfiguriert haben, können Sie das Ganze gleich ausprobieren! Legen Sie einfach eine neue *extensions.conf* mit diesem kurzen Wählplan an. (Wenn es nicht funktioniert, überprüfen Sie die Asterisk-Konsole nach Fehlermeldungen, und stellen Sie sicher, dass Ihre Kanäle so konfiguriert sind, dass eingehende Anrufe an den [incoming]-Kontext übergeben werden.)

Obwohl dieses Beispiel sehr kurz und einfach ist, macht es die Kernkonzepte der Kontexte, Extensions, Prioritäten und Anwendungen deutlich. Nachdem wir diese grundlegenden Konzepte erläutert haben, wollen wir auf unserem Beispiel weiter aufbauen. Ein Telefonsystem, das einfach eine Sounddatei abspielt und dann wieder aufhängt, ist schließlich nicht besonders nützlich!

Logik für den Wählplan

Der gerade erstellte Wählplan ist statisch – er führt bei jedem Anruf die gleiche Aktion aus. Wir wollen unseren Wählplan nun um etwas zusätzliche Logik erweitern, so dass er basierend auf den Benutzereingaben unterschiedliche Aktionen durchführt.

Die Anwendungen `Background()` und `Goto()`

Eine wichtige Schlüsselstellung beim Aufbau interaktiver Asterisk-Systeme nimmt die `Background()`-Anwendung ein. Wie `Playback()` spielt sie eine aufgezeichnete Sounddatei ab. Im Gegensatz zu `Playback()` wird die Wiedergabe aber unterbrochen, sobald der Anrufer eine Taste (oder eine Reihe von Tasten) drückt, und zu der Extension gesprungen, die der/den eingegebenen Ziffer(n) entspricht. Drückt ein Anrufer zum Beispiel die 5, hält Asterisk die Wiedergabe der Sounddatei an und übergibt die Kontrolle des Anrufs an die erste Priorität der Extension 5.

Die `Background()`-Anwendung wird vorzugsweise für den Aufbau von Voice-Menüs eingesetzt (die auch als *Auto Attendants* bezeichnet werden). Viele Unternehmen nutzen Voice-Menüs, um Anrufer an die richtigen Nebenstellen weiterzuleiten, ohne dass der Empfang jeden einzelnen Anruf entgegennehmen muss.

`Background()` verwendet die gleiche Syntax wie `Playback()`:

```
exten => 123,1,Background(hello-world)
```

Eine andere nützliche Anwendung ist `Goto()`. Wie es der Name schon andeutet, wird sie verwendet, um einen Anruf an einen anderen Kontext, eine Extension und Priorität zu übergeben. Die `Goto()`-Anwendung macht es einfach, einen Anruf programmatisch zwischen verschiedenen Teilen des Wählplans zu verschieben. Die Syntax der `Goto()`-Anwendung verlangt von uns die Übergabe des Zielkontexts, der Extension und der Priorität:

```
exten => 123,1,Goto(kontext,extension,priorität)
```

In unserem nächsten Beispiel verwenden wir `Background()` und `Goto()`, um einen etwas komplexeren Wählplan aufzubauen. Er erlaubt dem Anrufer die Interaktion mit dem System durch Drücken der Ziffern auf dem Tastenfeld. Beginnen wir damit, `Background()` die Eingaben des Anrufers akzeptieren zu lassen:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
```

In diesem Beispiel geben wir einfach die Sounddatei namens *enter-ext-of-person.gsm* wieder. Das ist zwar nicht die perfekte Begrüßung für einen Auto Attendant, aber für das Beispiel reicht es sicherlich aus. Lassen Sie uns nun zwei Extensions hinzufügen, die angestoßen werden, sobald der Anrufer während der Ansage eine 1 oder 2 eingibt:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
exten => 1,1,Playback(digits/1)
exten => 2,1,Playback(digits/2)
```

Bevor wir weitermachen, wollen wir zusammenfassen, was wir bisher getan haben. Wenn der Benutzer unseren Wählplan anruft, hört er als Begrüßung »Please enter the number you wish to call«. Wenn er die 1 drückt, hört er die Nummer eins, und wenn er die 2 drückt, dann hört er die Nummer zwei. Das ist ein guter Anfang, den wir nun weiter ausschmücken wollen. Wir wollen die `Goto()`-Anwendung nutzen, damit der Wählplan die Begrüßung nach der Wiedergabe der Nummer wiederholt:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
exten => 1,1,Playback(digits/1)
exten => 1,2,Goto(incoming,s,1)
exten => 2,1,Playback(digits/2)
exten => 2,2,Goto(incoming,s,1)
```

Die beiden neuen (fett gedruckten) Zeilen übergeben die Kontrolle wieder an die Extension s, nachdem die gewählte Nummer wiedergegeben wurde.



Wenn Sie sich die Details der Goto()-Anwendung ansehen, dann erkennen Sie, dass Sie tatsächlich ein, zwei oder drei Argumente an die Anwendung übergeben können. Wenn Sie ein einzelnes Argument übergeben, wird davon ausgegangen, dass die Zielpriorität der aktuellen Extension gemeint ist. Wenn Sie zwei Argumente übergeben, werden sie als Extension und Priorität des aktuellen Kontexts interpretiert.

In unserem Beispiel haben wir der Klarheit halber alle drei Argumente übergeben, aber die Übergabe der Extension und der Priorität hätten ausgereicht.

Ungültige Einträge und Timeouts behandeln

Da unser erstes Voice-Menü fast fertig ist, wollen wir einige zusätzliche spezielle Extensions hinzufügen. Zuerst benötigen wir eine Extension für ungültige Einträge. Wenn ein Anrufer eine ungültige Taste drückt (im obigen Beispiel etwa die 3), dann wird der Anruf an die Extension i übergeben. Zweitens benötigen wir eine Extension, die Situationen handhabt, in denen der Benutzer die Angaben nicht innerhalb einer gewissen Zeitspanne eingibt (der Standard-Timeout liegt bei 10 Sekunden). Anrufe werden an die t-Extension übergeben, wenn der Anrufer zu lange braucht, um eine Ziffer einzugeben, nachdem Background() die Sounddatei abgespielt hat. So sieht unser Wahlplan aus, nachdem wir die beiden Extensions aufgenommen haben:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
exten => 1,1,Playback(digits/1)
exten => 1,2,Goto(incoming,s,1)
exten => 2,1,Playback(digits/2)
exten => 2,2,Goto(incoming,s,1)
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup()
```

Die Verwendung der i- und t-Extensions macht den Wahlplan etwas robuster und benutzerfreundlicher. Dennoch ist er noch recht beschränkt, weil die Anrufer keine Möglichkeit haben, die Verbindung mit einer lebenden Person herzustellen. Zu diesem Zweck müssen wir eine weitere Anwendung namens Dial() kennen lernen.

Die Dial()-Anwendung

Eines der nützlichsten Features von Asterisk ist die Fähigkeit, einzelne Anrufer miteinander zu verbinden. Das ist insbesondere dann nützlich, wenn Anrufer unterschiedliche Methoden der Kommunikation nutzen. Zum Beispiel könnte Anrufer A ein einfaches

analoges Telefonnetz verwenden, während Anrufer B in einem Café am anderen Ende der Welt sitzt und in ein IP-Telefon spricht. Glücklicherweise übernimmt Asterisk einen Großteil der Schwerarbeit, die bei der Verbindung und Übersetzung unterschiedlicher Netzwerke notwendig ist. Sie müssen nur lernen, wie man die `Dial()`-Anwendung einsetzt.

Die Syntax der `Dial()`-Anwendung ist etwas komplexer als die der bisher von uns verwendeten Anwendungen, aber lassen Sie sich davon nicht abschrecken. `Dial()` verlangt bis zu vier Argumente. Das erste ist das Ziel, das Sie anrufen wollen. Dieses Ziel besteht aus einer Technologie (einem Transport), über die der Anruf erfolgt, einem Slash und einer entfernten Ressource (üblicherweise ein Kanalname oder eine -nummer). Nehmen wir zum Beispiel an, Sie wollen einen Zap-Kanal namens `Zap/1` anwählen, bei dem es sich um einen FXS-Kanal mit angeschlossenem Analogtelefon handelt. Die Technologie heißt »Zap«, und die Ressource ist »1«. Ähnlich verhält es sich bei einem Anruf zu einem SIP-Gerät mit dem Ziel `SIP/1234` und dem Anruf eines IAX-Geräts mit dem Ziel `IAX/fred`. Soll Asterisk den `Zap/1`-Kanal anwählen, wenn die Extension 123 im Wählplan erreicht wird, dann müssen wir die folgende Extension hinzufügen:

```
exten => 123,1,Dial(Zap/1)
```

Wenn diese Extension ausgeführt wird, ruft Asterisk das Telefon an, das mit dem Kanal `Zap/1` verbunden ist. Wird der Anruf angenommen, verbindet Asterisk den eingehenden Anruf mit dem `Zap/1`-Kanal. Wir können auch mehrere Kanäle gleichzeitig anwählen, indem wir die Ziele mit einem Kaufmanns-Und (Ampersand, `&`) verknüpfen:

```
exten => 123,1,Dial(Zap/1&Zap/2&Zap/3)
```

Die `Dial()`-Anwendung verbindet den eingehenden Anruf mit dem zuerst annehmenden Kanal.

Das zweite Argument der `Dial()`-Anwendung ist ein Timeout in Sekunden. Wird ein Timeout angegeben, versucht `Dial()` das/die Ziel(e) für die angegebene Zeitspanne anzurufen, bevor es mit der nächsten Priorität in der Extension weitermacht. Wird kein Timeout angegeben, wählt `Dial()` den/die Kanal/Kanäle so lange an, bis jemand annimmt oder der Anrufer auflegt. Lassen Sie uns einen Timeout von 10 Sekunden für unsere Extension hinzufügen:

```
exten => 123,1,Dial(Zap/1,10)
```

Wird der Anruf vor dem Timeout angenommen, werden die Kanäle verbunden, und der Wählplan ist abgeschlossen. Wenn das Ziel nicht antwortet, macht `Dial()` mit der nächsten Priorität der Extension weiter. Ist der Zielkanal hingegen besetzt, macht `Dial()` mit der Priorität `n+101` weiter, falls diese existiert (`n` ist dabei die Priorität, an der die `Dial()`-Anwendung aufgerufen wurde). Dadurch können unbeantwortete Anrufe anders behandelt werden als Anrufe, deren Leitungen besetzt sind.

Lassen Sie uns das bisher Erlernte in einem weiteren Beispiel zusammenfassen:

```

exten => 123,1,Dial(Zap/1,10)
exten => 123,2,Playback(vm-nobodyavail)
exten => 123,3,Hangup( )
exten => 123,102,Playback(tt-allbusy)
exten => 123,103,Hangup( )

```

Wie Sie sehen können, spielen wir in diesem Beispiel die Sounddatei *vm-nobodyavail.gsm*, wenn der Anruf unbeantwortet bleibt, und die Sounddatei *tt-allbusy.gsm*, wenn der Zap/1-Kanal besetzt ist.

Das dritte Argument für `Dial()` ist ein Optionsstring. Dieser kann ein oder mehrere Zeichen enthalten, die das Verhalten der `Dial()`-Anwendung bestimmen. Die Liste der möglichen Optionen ist zwar zu lang, um sie hier zu behandeln, aber die populärste Option ist `r`. Wenn Sie den Buchstaben `r` als drittes Argument angeben, hört die anrufende Seite einen Klingelton, während der Zielkanal über den eingehenden Anruf informiert wird.

Beachten Sie, dass die Option `r` nicht immer benötigt wird, um ein Klingeln anzudeuten, da Asterisk automatisch einen Klingelton erzeugt, wenn es versucht, einen Kanal aufzubauen. Sie können die Option `r` aber nutzen, um ein Klingeln durch Asterisk zu erzwingen, auch wenn keine Verbindung aufgebaut wird. Um die Option `r` in unser Beispiel einzufügen, ändern wir einfach die erste Zeile:

```

exten => 123,1,Dial(Zap/1,10,r)
exten => 123,2,Playback(vm-nobodyavail)
exten => 123,3,Hangup( )
exten => 123,102,Playback(tt-allbusy)
exten => 123,103,Hangup( )

```

Weil die Extensions 1 und 2 in unserem Wählplan eigentlich nutzlos sind, können wir sie jetzt (da wir wissen, wie die `Dial()`-Anwendung funktioniert) durch die Extensions 101 und 102 ersetzen, die es von außen kommenden Anrufern erlauben, John und Jane zu erreichen:

```

[incoming]
exten => s,1,Answer( )
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(Zap/1,10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup( )
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup( )
exten => 102,1,Dial(SIP/Jane,10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup( )
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup( )
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup( )

```

Das vierte und letzte Argument der `Dial()`-Anwendung ist eine URL. Wenn der Zielkanal den Empfang einer URL während des Anrufs unterstützt, dann wird die angegebene URL gesendet. (Wenn Sie zum Beispiel ein IP-Telefon verwenden, das den Empfang einer URL unterstützt, dann erscheint die URL im Display des Telefons; bei einem Softphone könnte die URL hingegen auf dem Bildschirm erscheinen.).Dieses Argument wird nur sehr selten benutzt.

Bei ausgehenden Anrufen über einen FXO Zap-Kanal können Sie die folgende Syntax verwenden, um eine Nummer auf diesem Kanal zu wählen:

```
exten => 123,1,Dial(Zap/4/5551212)
```

Dieses Beispiel würde die Nummer 5551212 auf dem Zap/4-Kanal wählen. Bei anderen Kanaltypen wie SIP und IAX geben Sie einfach das Ziel als Ressource an, wie die folgenden beiden Zeilen zeigen:

```
exten => 123,1,Dial(SIP/1234)
exten => 124,1,Dial(IAX2/john@asteriskdocs.org)
```

Beachten Sie, dass Sie jedes dieser Argumente weglassen können. Soll beispielsweise eine Option, aber kein Timeout angegeben werden, dann lassen Sie das Timeout-Argument einfach weg:

```
exten => 123,1,Dial(Zap/1,,r)
```

Einen Kontext für interne Anrufe hinzufügen

In unseren Beispielen haben wir uns bislang auf einen einzelnen Kontext beschränkt, aber es ist wohl nur logisch anzunehmen, dass nahezu jede Asterisk-Installation mehr als einen Kontext in ihrem Wählplan besitzen wird. Wie zu Beginn des Kapitels erwähnt, besteht eine wichtige Funktion von Kontexten darin, die Privilegien für verschiedene Arten von Anrufern zu trennen (etwa das Recht, Ferngespräche zu führen oder bestimmte Nebenstellen anrufen zu können). In unserem nächsten Beispiel legen wir zwei interne Nebenstellen an und geben ihnen die Möglichkeit, sich gegenseitig anzurufen. Zu diesem Zweck legen wir einen neuen Kontext namens `[internal]` an.



Wie bei den vorangegangenen Beispielen setzen wir voraus, dass bereits ein FXS Zap-Kanal (in diesem Fall Zap/1) konfiguriert wurde und dass Ihre `zapata.conf` so konfiguriert ist, dass alle von Zap/1 stammenden Anrufe im `[internal]`-Kontext beginnen. Bei einigen Beispielen gegen Ende dieses Kapitels gehen wir auch davon aus, dass ein FXO Zap-Kanal als Zap/4 konfiguriert wurde und auf diesem Kanal eingehende Anrufe an den `[incoming]`-Kontext übergeben werden. Dieser Kanal wird für ausgehende Anrufe verwendet.

Wir sind auch davon ausgegangen, dass Sie über mindestens einen SIP-Kanal (namens SIP/jane) verfügen, der ursprünglich aus dem `[internal]`-Kontext stammt. Wir haben das gemacht, um Sie an die Verwendung anderer Kanaltypen heranzuführen.

Wenn Sie keine Hardware für die oben aufgeführten Kanäle (wie Zap/4) besitzen oder wenn Sie Hardware mit anderen Kanalnamen verwenden (z.B. nicht SIP/jane), dann passen Sie die Beispiele einfach an Ihre jeweilige Systemkonfiguration an.

Unser Wählplan sieht jetzt wie folgt aus:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(Zap/1,10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup()
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup()
exten => 102,1,Dial(SIP/Jane,10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup()
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup()
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup()

[internal]
exten => 101,1,Dial(Zap/1,,r)
exten => 102,1,Dial(SIP/jane,,r)
```

In diesem Beispiel haben wir zwei neue Extensions für den [internal]-Kontext hinzugefügt. Auf diese Weise kann die Person, die den Zap/1-Kanal nutzt, den Hörer abnehmen und die Person an Kanal SIP/jane anrufen, indem sie die 102 wählt. In gleicher Weise kann das als SIP/jane registrierte Telefon Zap/1 erreichen, indem die 101 gewählt wird.

Wir haben uns für unsere Beispiele willkürlich für die Nebenstellen 101 und 102 entschieden, aber Sie können für Ihre Durchwahlen natürlich jedes beliebige Nummernschema wählen. Sie sollten sich auch dessen bewusst sein, dass Sie nicht auf dreistellige Durchwahlen beschränkt sind – Sie können so viele Ziffern verwenden, wie Sie wollen. (Na ja, fast. Extensions dürfen nicht länger als 80 Zeichen sein, und Sie dürfen keine Extensions mit nur einem Zeichen verwenden, da diese reserviert sind.) Denken Sie daran, dass Sie auch Namen verwenden können:

```
[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(Zap/1,10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup()
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup()
exten => 102,1,Dial(SIP/Jane,10)
```

```
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup( )
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup( )
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup( )
```

```
[internal]
exten => 101,1,Dial(Zap/1,,r)
exten => john,1,Dial(Zap/1,,r)
exten => 102,1,Dial(SIP/jane,,r)
exten => jane,1,Dial(SIP/jane,,r)
```

Es schadet natürlich nicht, benannte Extensions aufzunehmen, wenn Ihre Benutzer über einen Namen unterstützenden VoIP-Transport angerufen werden könnten.

Nachdem sich die internen Teilnehmer nun untereinander anrufen können, sind wir auf dem besten Weg zu einem vollständigen Wählplan. Als Nächstes sehen wir uns an, wie wir unseren Wählplan skalierbarer gestalten und die zukünftige Modifikation vereinfachen können.

Variablen

Variablen können im Asterisk-Wählplan verwendet werden, um die Schreibaarbeit zu reduzieren, die Klarheit zu erhöhen und um zusätzliche Logik in den Wählplan zu integrieren. Wenn Sie etwas von der Programmierung von Computern verstehen, wissen Sie bereits, was eine Variable ist. Wenn nicht, keine Sorge, wir erklären, was Variablen sind und wie man sie verwendet.

Sie können sich eine Variable als einen Container vorstellen, der zu einer bestimmten Zeit einen bestimmten Wert enthalten kann. So können wir zum Beispiel eine Variable namens JOHN anlegen und ihr den Wert Zap/1 zuweisen. Beim Schreiben unseres Wählplans können wir Johns Kanal immer über den Namen angeben, statt uns merken zu müssen, dass John Zap/1 verwendet. Um einer Variablen einen Wert zuzuweisen, geben Sie einfach den Namen der Variablen ein, gefolgt von einem Gleichheitszeichen und dem gewünschten Wert:

```
JOHN=Zap/1
```

Es gibt zwei Möglichkeiten, eine Variable zu referenzieren. Um den Namen einer Variablen zu referenzieren, geben Sie einfach deren Namen ein, etwa JOHN. Wenn Sie andererseits den Wert der Variablen referenzieren wollen, müssen Sie ein Dollarzeichen, eine öffnende geschweifte Klammer, den Namen der Variablen, und eine schließende geschweifte Klammer eingeben. Einen Variablenwert referenzieren Sie innerhalb der Dial()-Anwendung also wie folgt:

```
exten => 555,1,Dial(${JOHN},,r)
```

Wenn Sie im Wählplan \${JOHN} schreiben, ersetzt Asterisk diesen Wert automatisch durch den Wert, der der Variablen JOHN zugewiesen wurde.



Beachten Sie, dass Variablenamen nicht großgeschrieben werden müssen. Wir machen das in diesem Buch nur, um die Lesbarkeit zu erhöhen.

Es gibt drei Arten von Variablen, die wir in unserem Wählplan verwenden können: globale Variablen, Kanalvariablen und Umgebungsvariablen. Sehen wir uns jede Art von Variable einmal genauer an.

Globale Variablen

Wie es der Name impliziert, gelten *globale* Variablen für alle Extensions in allen Kontexten. Globale Variablen sind nützlich, da sie überall innerhalb eines Wählplans genutzt werden können, um die Lesbarkeit und Wartbarkeit zu erhöhen. Stellen Sie sich einen großen Wählplan vor, der hunderte Referenzen auf den Zap/1-Kanal enthält. Stellen Sie sich nun vor, wie Sie den gesamten Wählplan durchgehen und alle Referenzen in Zap/2 ändern müssen. Das wäre zumindest ein langer und fehleranfälliger Prozess.

Wenn Sie andererseits zu Beginn Ihres Wählplans eine globale Variable mit dem Wert Zap/1 definiert und diesen dann referenziert hätten, müssten Sie nur eine einzige Zeile ändern.

Globale Variablen müssen im [globals]-Kontext am Anfang der *extensions.conf*-Datei definiert werden. Sie können mit Hilfe der `SetGlobalVar()`-Anwendung auch programmatisch definiert werden. Innerhalb eines Wählplans sehen beide Methoden wie folgt aus:

```
[globals]
JOHN=Zap/1

[internal]
exten => 123,1,SetGlobalVar(JOHN=Zap/1)
```

Kanalvariablen

Eine *Kanal*-Variable ist eine Variable, die nur mit einem bestimmten Anruf verknüpft ist (etwa die Anruferkennung). Im Gegensatz zu globalen Variablen sind Kanalvariablen nur für die Dauer des aktuellen Anrufs definiert und nur innerhalb des Kanals verfügbar, über den der Anruf läuft.

Es gibt viele vordefinierte Kanalvariablen, die innerhalb des Wählplans verwendet werden können. Sie sind in der Datei *README.variables* im *doc*-Unterverzeichnis der Asterisk-Quellen beschrieben. Kanalvariablen werden über die `Set()`-Anwendung gesetzt:

```
exten => 123,1,Set(MAGICNUMBER=42)
```

Wir verwenden im nächsten Kapitel verschiedene Kanalvariablen.

Umgebungsvariablen

Umgebungsvariablen bieten die Möglichkeit, aus Asterisk heraus auf die Unix-Umgebungsvariablen zuzugreifen. Die Referenzierung erfolgt in der Form `${ENV(var)}`, wobei *var* die Unix-Umgebungsvariable ist, die Sie referenzieren wollen.

Variablen in den Wählplan einfügen

Nachdem wir die Variablen kennen gelernt haben, wollen wir sie in unseren Wählplan integrieren. Lassen Sie uns Variablen für zwei Personen, John und Jane, hinzufügen:

```
[globals]
JOHN=Zap/1
JANE=SIP/jane

[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(${JOHN},10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup()
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup()
exten => 102,1,Dial(${JANE},10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup()
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup()
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup()

[internal]
exten => 101,1,Dial(${JOHN},,r)
exten => 102,1,Dial(${JANE},,r)
```

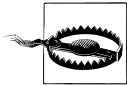
Pattern Matching

Häufig wäre es eine langwierige Aufgabe, alle möglichen Extensions in einen Wählplan einzutragen. Das gilt insbesondere für ausgehende Anrufe. Können Sie sich einen Wählplan vorstellen, der eine Extension für jede mögliche Nummer enthält, die Sie wählen könnten? Glücklicherweise hat Asterisk genau für solche Fälle das richtige Mittel: Mit *Pattern Matching* (Mustererkennung) können Sie einen Codeabschnitt für viele verschiedene Extensions verwenden.

Pattern-Matching-Syntax

Beim Pattern Matching verwenden wir verschiedene Buchstaben und Symbole, um die möglichen Ziffern darzustellen, die wir erkennen möchten. Muster beginnen immer mit

einem Unterstrich (). Damit teilen wir Asterisk mit, dass wir ein Muster verarbeiten und nicht den Namen einer Extension. (Das bedeutet natürlich, dass die Namen Ihrer Extensions niemals mit einem Unterstrich beginnen dürfen.)



Wenn Sie den Unterstrich zu Beginn des Musters vergessen, geht Asterisk einfach von einer benannten Extension aus und nimmt kein Pattern Matching vor.

Auf den Unterstrich können ein oder mehrere der folgenden Zeichen folgen:

X

Erkennt alle Ziffern zwischen 0 und 9.

Z

Erkennt alle Ziffern zwischen 1 und 9.

N

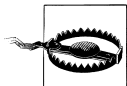
Erkennt alle Ziffern zwischen 2 und 9.

[15-7]

Erkennt jede Ziffer, bzw. jeden angegebenen Bereich von Ziffern. In diesem Fall werden 1, 5, 6 oder 7 erkannt

. (Punkt)

Wildcard-Symbol. Erkennt ein oder mehrere Zeichen.



Wenn Sie nicht vorsichtig sind, können Wildcards Ihre Wählpläne zu Dingen veranlassen, die Sie so vielleicht nicht erwarten. Sie dürfen Wildcards in einem Muster erst einsetzen, wenn schon so viele Ziffern wie möglich angegeben wurden. Zum Beispiel sollte das folgende Muster niemals verwendet werden:

-*

Tatsächlich gibt Asterisk eine Warnung aus, wenn Sie das versuchen. Verwenden Sie stattdessen, wenn möglich, dieses Muster:

 X.

Um das Pattern Matching in Ihrem Wählplan zu verwenden, tragen Sie einfach anstelle des Extension-Namens (oder der Nummer) das Muster ein:

```
exten =>   NXX,1,Playback(auth-thankyou)
```

In diesem Beispiel würde das Muster jede dreistellige Extension zwischen 200 und 999 erkennen (N erkennt jede Ziffer zwischen 2 und 9, und X erkennt jede Ziffer zwischen 0 und 9). Wenn der Anrufer in diesem Kontext eine dreistellige Zahl zwischen 200 und 999 eingeben würde, würde er die Sounddatei *auth-thankyou.gsm* hören.

Eine weitere wichtige Sache, die Sie über das Pattern Matching wissen sollten, ist, dass Asterisk bei zwei passenden Mustern für eine gewählte Extension *das genauere* verwendet. Nehmen wir an, Sie haben die beiden folgenden Muster definiert und ein Anrufer wählt 888-555-1212:

D

Tja, das mit der 888 ist so eine Sache. Erst einmal ist es falsch, die Extension wird nicht erkannt, da sie ja mit 888 anfängt. Wenn aber die 888 eine Vorwahl wäre und der Provider nur die Stammnummer übergibt, also die 5551212, dann funktioniert das Beispiel.

```
exten => _555XXXX,1,Playback(digits/1)
```

```
exten => _55512XX,1,Playback(digits/2)
```

In diesem Fall würde die zweite Extension verwendet werden, weil sie genauer ist.

Pattern-Matching-Beispiele

Bevor wir weitermachen, wollen wir uns einige weitere Pattern-Matching-Beispiele ansehen. Versuchen Sie bei jedem Muster zu erkennen, was es macht, bevor Sie die Erläuterung lesen. Wir beginnen mit einem einfachen Muster:

```
_NXXXXXX
```

Haben Sie es? Dieses Muster würde jede siebenstellige Zahl erkennen, deren erste Ziffer zwei oder höher ist. Entsprechend dem Nordamerikanischen Nummerierungsplan (NANP) erkennt dieses Muster jede lokale Nummer.

Sehen wir uns ein anderes Beispiel an:

```
_1NXXNXXXXXX
```

Das ist etwas schwieriger. Das Muster erkennt die Zahl 1, gefolgt von einer Vorwahl zwischen 200 und 999, gefolgt von einer beliebigen siebenstelligen Nummer. Dem NANP entsprechend würde dieses Muster jedes Ferngespräch erkennen.

Jetzt ein noch etwas schwierigeres Beispiel:

```
_011.
```

Wenn Sie sich jetzt am Kopf kratzen, sehen Sie noch einmal genau hin. Haben Sie den Punkt am Ende bemerkt? Dieses Muster erkennt jede Nummer, die mit 011 beginnt und noch mindestens eine weitere Ziffer verwendet. Nach dem NANP weist dies auf eine internationale Telefonnummer hin.³

³ Wenn Ihnen aufgefallen ist, dass wir Muster gewählt haben, die zur Wahl externer Nummern im NANP verwendet werden, dann haben Sie völlig recht. Wir verwenden diese Muster im nächsten Abschnitt, um unseren Wählplan um die Möglichkeit ausgehender Anrufe zu erweitern.

NANP und Telefon-Abzocke

Der Nordamerikanische Nummernplan (NANP) ist ein Telefon-Nummerierungsschema, das von 19 Staaten in Nordamerika und der Karibik verwendet wird.

In den USA und Kanada sind die Telekommunikationsregelungen ähnlich (und vernünftig) genug, dass Sie davon ausgehen können, dass die meisten Ferngespräche für den Ländercode 1 eine angemessene Gebühr kosten. Was viele Leute aber übersehen, ist die Tatsache, dass 19 Länder mit zum Teil sehr unterschiedlichen Telecom-Regulierungen den NANP nutzen. (Weitere Informationen finden Sie auf <http://www.nanpa.com>.)

Der Trick vieler Telefon-Abzocker besteht nun darin, in den USA ansässige Leute dazu zu bringen, unglaublich teure kostenpflichtige Nummern in einem Karibikstaat anzurufen. Die Anrufer glauben, nur die Kosten für ein normales Ferngespräch zu zahlen, weil sie eine 1-NPA-NXX-XXXX-Nummer anrufen. Da der fragliche Staat aber über Regulierungen verfügen kann, die diese Form der Abzocke erlauben, wird der Anrufer letztendlich für die Gesprächskosten zur Verantwortung gezogen.

Die einzige Möglichkeit, so etwas zu verhindern, besteht darin, bestimmte Vorwahlen zu sperren (z.B. 809), und diese Beschränkung nur bei Bedarf aufzuheben. Treffen Sie also die entsprechenden Maßnahmen, damit die Mitarbeiter das Telefon nicht missbrauchen können.

Die Kanalvariable \${EXTEN}

Wir wissen, was Sie denken. Sie sitzen da und fragen sich: »Ich möchte das Pattern Matching nutzen, aber woher weiß ich, welche Ziffern tatsächlich gewählt wurden?« Glücklicherweise hat Asterisk auch hier die passende Antwort. Sobald eine Extension gewählt wird, weist Asterisk der Kanalvariablen \${EXTEN} die Ziffern zu, die gewählt wurden. Sie können eine Anwendung namens SayDigits() verwenden, um das durchzutesten:

```
exten => _XXX,1,SayDigits(${EXTEN})
```

Bei diesem Beispiel ruft SayDigits() die von Ihnen gewählte dreistellige Extension ab.

Häufig ist es nützlich, \${EXTEN} zu korrigieren, indem man eine bestimmte Anzahl von Ziffern zu Beginn der Extension entfernt. Sie erreichen dies über die Syntax \${EXTEN:x}, wobei x die Anzahl der Ziffern ist, die Sie entfernen wollen. Wenn EXTEN also beispielsweise den Wert 95551212 enthält, entspricht \${EXTEN:1} dem Wert 5551212. Sehen wir uns ein weiteres Beispiel an:

```
exten => _XXX,1,SayDigits(${EXTEN:1})
```

In diesem Beispiel würde SayDigits() nur die letzten beiden Ziffern der gewählten Extension abrufen.

Ist x negativ, gibt SayDigits() nur die letzten x Ziffern der gewählten Extension zurück. Im nächsten Beispiel gibt SayDigits() nur die letzte Ziffer der gewählten Extension zurück:

```
exten => _XXX,1,SayDigits(${EXTEN:-1})
```

Ausgehende Anrufe aktivieren

Nachdem wir das Pattern Matching eingeführt haben, wollen wir den Benutzern nun erlauben, ausgehende Anrufe zu tätigen. Als Erstes müssen wir eine Variable in den [globals]-Kontext aufnehmen, um den Kanal zu definieren, der für ausgehende Anrufe verwendet werden soll:

```
[globals]
JOHN=Zap/1
JANE=SIP/jane
OUTBOUNDTRUNK=Zap/4
```

Als Nächstes fügen wir Kontexte für ausgehende Anrufe in unseren Wahlplan ein.

An diesem Punkt fragen Sie sich vielleicht: »Warum brauche ich separate Kontexte für ausgehende Anrufe?« Nun, auf diese Weise können wir regulieren und kontrollieren, wer ausgehende Anrufe tätigen darf und welche Arten von ausgehenden Anrufen erlaubt sind.

Zuerst legen wir einen Kontext für lokale Anrufe an. Um die Konsistenz zu traditionellen Telefonsystemen zu wahren, stellen wir unseren Mustern eine 9 voran, d.h., die Benutzer müssen eine 9 vor der ausgehenden Nummer wählen (in Deutschland normalerweise die 0 statt der 9):

```
[outbound-local]
exten => _9NXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _9NXXXXXX,2,Congestion()
exten => _9NXXXXXX,102,Congestion()
```

D

In Deutschland funktioniert das Erkennen von Telefonnummern anders, da es unterschiedlich lange Vorwahlnummern gibt. Es gibt nur die 0 für ein Ferngespräch und die 00 für ein internationales Gespräch. Somit sieht das obige Beispiel abgeändert so aus:

```
[outbound-local]
exten => _0X.,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _0X.,2,Congestion()
exten => _0.,102,Congestion()
```



Beachten Sie, dass das Wählen der 9 Ihnen im Gegensatz zu vielen traditionellen Telefonanlagen keine Amtsleitung gibt. Sobald Sie die 9 auf einer FXS-Leitung wählen, hört der Wählton auf. Soll der Wählton auch zu hören sein, nachdem Sie die 9 gewählt haben, nehmen Sie die folgende Zeile auf (direkt hinter der Kontext-Definition):

```
ignorepat => 9
```

Diese Direktive weist Asterisk an, den Wählton auch dann noch auszugeben, wenn der Benutzer das angegebene Muster gewählt hat.

Fassen wir zusammen, was wir gerade getan haben. Wir haben eine globale Variable namens OUTBOUNDTRUNK hinzugefügt, mit der wir festlegen, welcher Kanal für ausgehende Anrufe verwendet werden soll. In Priorität 1 nehmen wir die gewählte Nummer, entfer-

nen die 9 mittels `${EXTEN:1}` und versuchen dann, diese Nummer auf dem Kanal zu wählen, der in `OUTBOUNDTRUNK` festgelegt wurde. Ist der Anruf erfolgreich, wird der Anrufer mit dem ausgehenden Kanal verbunden. Ist der Anruf nicht erfolgreich (weil der Kanal belegt ist, oder die Nummer aus irgendeinem Grund nicht angewählt werden kann), wird die `Congestion()`-Anwendung aufgerufen, die ein »schnelles Besetztzeichen« ausgibt, um den Anrufer wissen zu lassen, dass der Anruf nicht erfolgreich war.

Bevor wir weitermachen, wollen wir sicherstellen, dass unser Wählplan ausgehende Notrufnummern erlaubt:

```
[outbound-local]
exten => _9NXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _9NXXXXXX,2,Congestion()
exten => _9NXXXXXX,102,Congestion()

exten => 110,1,Dial(${OUTBOUNDTRUNK}/110)
exten => 9110,1,Dial(${OUTBOUNDTRUNK}/110)
```

Wir gehen hier davon aus, dass wir uns in Deutschland befinden. Wenn Sie sich nicht in Deutschland befinden, müssen Sie die 110 durch die entsprechende Notrufnummer Ihres Landes ersetzen. Das ist etwas, das Sie in Ihrem Wählplan niemals vergessen dürfen!

Als Nächstes fügen wir einen Kontext für Ferngespräche hinzu:

```
[outbound-long-distance]
exten => _91NXXNXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _91NXXNXXXXXX,2,Congestion()
exten => _91NXXNXXXXXX,102,Congestion()
```

D Auch hier gilt für Deutschland wieder etwas anderes:

```
[outbound-long-distance]
exten => _00.,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _00.,2,Congestion()
exten => _00.,102,Congestion()
```

Nachdem wir nun zwei neue Kontexte besitzen, stellt sich die Frage, wie diese von den internen Benutzern verwendet werden können, diese zu nutzen. Wir benötigen eine Möglichkeit, um Kontexte innerhalb anderer Kontexte zu verwenden.

Includes

Asterisk ermöglicht es uns mit Hilfe der `include`-Direktive, einen Kontext innerhalb eines anderen Kontexts zu nutzen. Das dient dazu, um Zugriff auf unterschiedliche Abschnitte des Wählplans zu gewähren. Mit Hilfe der `include`-Funktionalität können wir den Benutzern unseres `[internal]`-Kontexts erlauben, ausgehende Anrufe zu tätigen. Aber zuerst wollen wir die Syntax behandeln.

Die `include`-Anweisung hat die folgende Form, wobei *kontext* der Name des Kontexts ist, den wir in den aktuellen Kontext integrieren wollen:

```
include => kontext
```

Wenn wir andere Kontexte in unseren aktuellen Kontext einbinden, müssen wir uns um die Reihenfolge Gedanken machen, in der wir sie einbinden. Asterisk versucht zuerst, die Extension im aktuellen Kontext zu erkennen. Wenn das fehlschlägt, versucht es sein Glück im ersten eingefügten Kontext und macht dann mit den anderen eingebundenen Kontexten weiter, und zwar in der Reihenfolge, in der sie eingefügt wurden.

So wie es aussieht, hat unser aktueller Wahlplan zwei Kontexte für ausgehende Anrufe, aber es gibt für die Teilnehmer im [internal]-Kontext keine Möglichkeit, diese zu verwenden. Wir wollen das ändern, indem wir die beiden ausgehenden Kontexte in den [internal]-Kontext einbinden:

```
[globals]
JOHN=Zap/1
JANE=SIP/jane
OUTBOUNDTRUNK=Zap/4

[incoming]
exten => s,1,Answer()
exten => s,2,Background(enter-ext-of-person)
exten => 101,1,Dial(${JOHN},10)
exten => 101,2,Playback(vm-nobodyavail)
exten => 101,3,Hangup()
exten => 101,102,Playback(tt-allbusy)
exten => 101,103,Hangup()
exten => 102,1,Dial(${JANE},10)
exten => 102,2,Playback(vm-nobodyavail)
exten => 102,3,Hangup()
exten => 102,102,Playback(tt-allbusy)
exten => 102,103,Hangup()
exten => i,1,Playback(pbx-invalid)
exten => i,2,Goto(incoming,s,1)
exten => t,1,Playback(vm-goodbye)
exten => t,2,Hangup()

[internal]
include => outbound-local
include => outbound-long-distance

exten => 101,1,Dial(${JOHN},,r)
exten => 102,1,Dial(${JANE},,r)

[outbound-local]
exten => _9NXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _9NXXXXXX,2,Congestion()
exten => _9NXXXXXX,102,Congestion()

exten => 911,1,Dial(${OUTBOUNDTRUNK}/911)
exten => 9911,1,Dial(${OUTBOUNDTRUNK}/911)

[outbound-long-distance]
exten => _91NXXXXXXXX,1,Dial(${OUTBOUNDTRUNK}/${EXTEN:1})
exten => _91NXXXXXXXX,2,Congestion()
exten => _91NXXXXXXXX,102,Congestion()
```

Diese beiden `include`-Anweisungen machen es den Teilnehmern des `[internal]`-Kontexts möglich, ausgehende Anrufe zu tätigen. Wir wollen auch darauf hinweisen, dass Sie aus Sicherheitsgründen darauf achten müssen, dass Ihr `[inbound]`-Kontext niemals ausgehende Anrufe erlaubt. (Wenn doch, könnten sich Leute in Ihr System einwählen und kostenpflichtige ausgehende Anrufe tätigen, die Ihnen in Rechnung gestellt werden!)

Schlussbemerkung

Und da ist er, unser einfacher, aber funktionierender Wählplan. Er ist noch nicht allumfassend, deckt aber alle Grundlagen ab. In den folgenden Kapiteln werden wir diese Grundlage immer mehr erweitern.

Wenn Ihnen Teile dieses Wählplans unverständlich sind, sollten Sie zurückgehen und ein oder zwei Abschnitte noch einmal lesen, bevor Sie mit dem nächsten Kapitel fortfahren. Es ist unbedingt erforderlich, dass Sie die hier vermittelten Prinzipien verstehen und anwenden können, sonst werden Sie die folgenden Kapitel nur noch mehr verwirren. Und wir wollen nicht, dass Sie verwirrt sind!