ALTA2004
Introduction to VoiceXML

Rolf Schwitter

schwitt@ics.mq.edu.au

The Program

The Program

Recommended Literature

James A. Larson
VoiceXML: Introduction to Developing Speech Applications
Prentice Hall, 2003.

Related Web Sites

- Speech Technology Magazine
  http://www.speechtechmag.com/
- VoiceBrowser Activity – Voice enabling the Web!
  http://www.w3.org/Voice/
- VoiceXML Forum
  http://www.voicexml.org/
- VoiceXML Version 2.0
  http://www.w3.org/TR/2004/REC-voicexml20-20040316/

Related Web Sites

- SALTforum
  http://www.saltforum.org/
- Tellme Studio
  http://studio.tellme.com/
- BeVocal Café
  http://cafe.bevocal.com/
- OptimTalk
  http://www.optimsys.cz/news/

Introduction to VoiceXML
1. Spoken Language Dialog Systems

Rolf Schwitter
schwitt@ics.mq.edu.au

What is a Spoken Language Dialog System?

- An SLDS is a computer system that you can talk to in order to carry out some task.
- SLDSs are typically of two kinds:
  - Information-provision systems provide information in response to a query, such as a request for timetable information or weather information.
  - Transaction-based systems allow you to undertake some transaction, such as buying or selling stocks, or reserving a seat on a plane.

Two Uses of Speech Recognition Technology

- Desktop-based:
  - speaker-dependent
  - large vocabulary (tens of thousands of words)
  - dictation tasks
- Telephony-based:
  - speaker-independent
  - relatively small vocabulary (hundreds of words)
  - interactive tasks

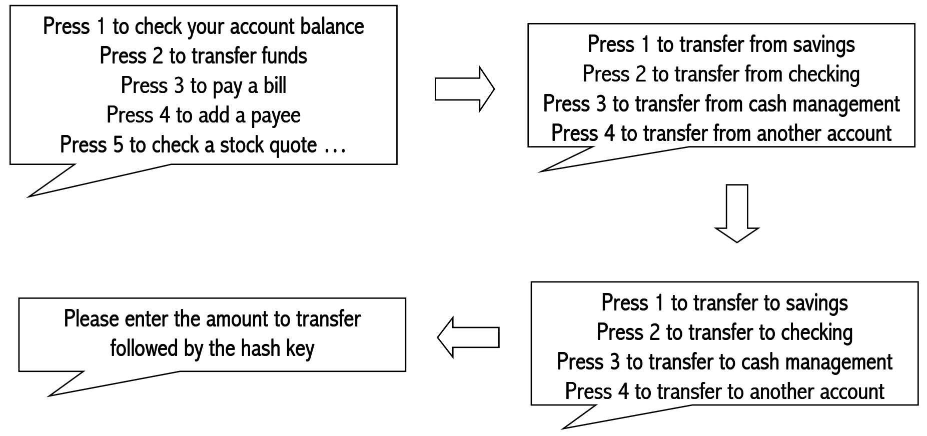
Uses of Desktop-based SLDS

- Dictation
- E-mail
- Voice control
- Navigation
- Instant translation

Uses of Telephony-Based SLDSs

- Remote banking
- Travel reservation
- Information enquiry
- Stock transaction
- Telebetting
- Directory assistance
- Taxi booking
- Pizza ordering

Traditional Interactive Voice Response Systems

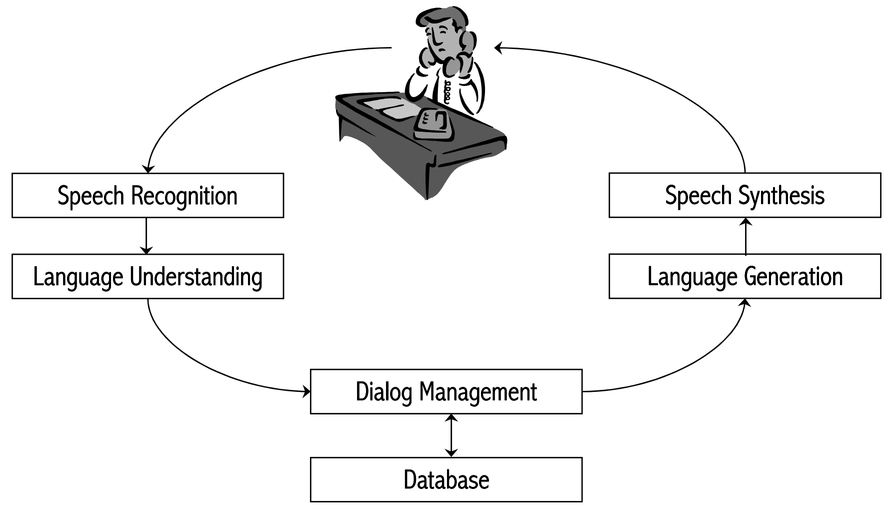

Press 1 to check your account balance
Press 2 to transfer funds
Press 3 to pay a bill
Press 4 to add a payee
Press 5 to check a stock quote …
Press 1 to transfer from savings
Press 2 to transfer from checking
Press 3 to transfer from cash management
Press 4 to transfer from another account
Press 1 to transfer to savings
Press 2 to transfer to checking
Press 3 to transfer to cash management
Press 4 to transfer to another account
Please enter the amount to transfer followed by the hash key

Speech-Enabled Interaction

Transfer 500 dollars from savings to checking next Wednesday after 3pm

~25 seconds via speech — as compared with two minutes via touch tone

The Architecture of a SLDS



What a SLDS Contains

- Speech Recognition - analyses the audio speech input signal to extract linguistic units such as words or phonemes.
- Language Understanding - determines the meaning of the input.
- Dialog Management - manages the flow of the conversation, maintaining history and context, directing its course, accessing the database, and formulating responses.
- Database - stores the information which provides the dialog content.
- Language Generation - puts the responses into words.
- Speech Synthesis - produces the audio speech output signal.

Examples: SLDSs

- Early interactions with machines
- Real systems today:
  - YellowCab – Taxi Dispatch Demo
  - Manhatten ATM Finder
- In the labs:
  - CMU communicator

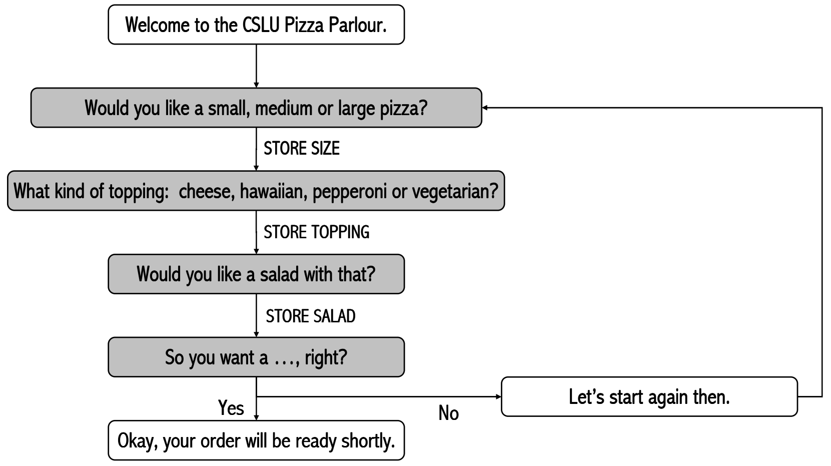
What's Involved in Building a SLDS

- Dialog Design
  - Working out how the interaction between human and machine will move from stage to stage.
- Prompt Design
  - Crafting speech messages played to a user to ask questions.
- Grammar Writing
  - Specifying what the user is permitted to say at any given state.
- Error Handling
  - Dealing with the inaccuracy of speech recognition technology.

Dialog Design

- To be habitable, SLDSs must behave in natural ways.
- The naturalness of the application depends in large part on the quality of the dialog flow.
- The dialog flow describes how the system advances from state to state in response to the user's inputs.

Example: Dialog Design


Welcome to the CSLU Pizza Parlour.
Would you like a small, medium or large pizza?
STORE SIZE
What kind of topping: cheese, hawaiian, pepperoni or vegetarian?
STORE TOPPING
Would you like a salad with that?
STORE SALAD
So you want a …, right?
Yes
No
Let's start again then.
Okay, your order will be ready shortly.

Prompt Design

- Prompts are the turn-taking cues within spoken dialogs.
- Prompts have two purposes:
  - cause the user to speak,
  - convey to the user what may be spoken.
- Design prompts to get the user say things you'd like them to say.
- Careful prompt design is one way of maintaining the system's control of the initiative in a dialog.

Example: Prompt Design

- Implicit versus explicit prompts:

Computer 1:	Welcome to ABC Bank. What would you like to do?
Computer 2:	Welcome to ABC Bank. You can check an account balance, transfer funds, or pay a bill. What would you like to do?
Computer 3:	Welcome to ABC Bank. You can check an account balance, transfer funds, or pay a bill. Say one of the following choices: check balance, transfer funds, or pay bills.

Grammar Writing

- Writing a good speech grammar is a trade-off:
  - broad coverage of the grammar is good because people express themselves in a variety of ways
  - but if the coverage of the grammar is too broad then recognition accuracy is increasingly challenged.
- Essential to coordinate prompt and grammar writing.

Example: Grammar Writing

```
<?xml version = "1.0"?>
<grammar xml:lang = "en" version = "1.0">
  <rule id = "city" scope = "public">
    <one-of>
      <item> berlin </item>
      <item> new york </item>
      <item> paris </item>
      <item> sydney </item>
    </one-of>
  </rule>
</grammar>
```

Error Handling

- All speech recognizers will make mistakes in recognition.
- You need to think about error handling from the beginning.
- Good design raises the accuracy of even poor recognizers.
- Bad design reduces the accuracy of the best recognizers.

Example: Error Handling

- What happens here – and how can we avoid this problem?

Computer:	Stock name?
Caller:	"<u>Texaco</u>."
Computer:	Shares of <u>PepsiCo</u> to sell?
Caller:	"… umh … No, that’s wrong …"

Introduction to VoiceXML
2. VoiceXML and W3C Speech Interface Framework

Rolf Schwitter

schwitt@ics.mq.edu.au

Developing Speech Interfaces

- Speech interfaces can be developed using
  - general-purpose languages (e.g. C++, Java, Python)
  - special-purpose languages (e.g. VoiceXML, SALT)
- A special-purpose language can
  - simplify application development,
  - separate interaction code from application logic code,
  - reduce network traffic,
  - provide portability and simplicity,
  - support prototyping and refinement.

What is VoiceXML?

- VoiceXML (Voice eXtensible Markup Language)
  - is an XML based markup language for specifying dialogs,
  - brings the Web to telephones,
  - is based upon extensive industry experience,
  - was contributed to W3C by members of the VoiceXML Forum.
- Check
  - W3C: http://www.w3.org/TR/voicexml20/
  - VoiceXML Forum: http://www.voicexml.org/index.html

History of VoiceXML

1995: Project at AT&T led to the PhoneMarkup Language (PML).

1998: AT&T and Lucent had variants of PML.
Motorola had VoxML.
IBM had Speech ML.
HP had TalkML, and
PipeBeach had VoiceHTML.

1999: VoiceXML Forum (AT&T, IBM, Lucent, Motorola) produced VoiceXML 0.9.

History of VoiceXML

2000: VoiceXML Forum released VoiceXML 1.0.
2000: VoiceXML Forum submitted the spec to W3C.
2002: VoiceXML 2.0 was released by the W3C.
2004: VoiceXML 2.0 is a W3C recommendation.
2004: VoiceXML 2.1 is a W3C working draft.

VoiceXML: Example 1

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">

  <form>
    <block> Hello World! </block>
  </form>

</vxml>
```

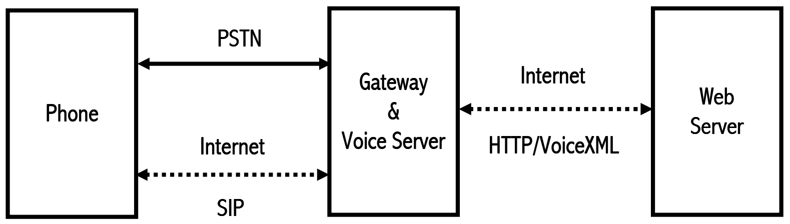
VoiceXML: Example 2

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">

<var name = "hi" expr = "'Hello World!'"/>

  <form id = "say_hi">
    <block>
      <prompt> <value expr = "hi"/> </prompt>
      <goto next = "#say_goodbye"/>
    </block>
  </form>

  <form id = "say_goodbye">
    <block> Goodbye! </block>
  </form>

</vxml>
```

VoiceXML: Example 3

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">

  <form>
    <field name = "drink">
      <prompt>
        Would you like to fly to New York or Boston?
      </prompt>
      <grammar src  = "destination.grxml"
               type = "application/srgs+xml"/>
    </field>
    <block>
      <submit namelist = "drink"
              next     = "http://www.../destination.py"/>
    </block>
  </form>

</vxml>
```

VoiceXML Architecture



- regular phone
- wireless phone
- soft phone

- telephony interface
- voice browser
- automated speech recognition
- text-to-speech synthesis
- touchtone
- audio play/record

- VoiceXML documents
- audio files
- service logic (CGI)
- transaction processing
- database interface

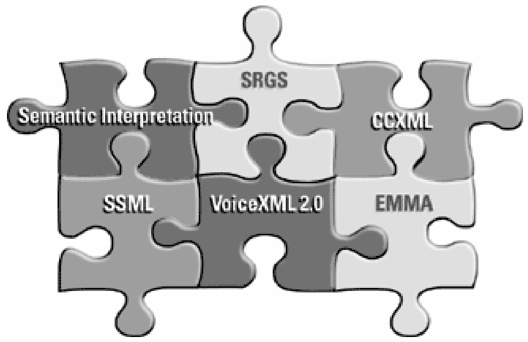
A VoiceXML Scenario

- A customer dials the phone number of a travel agent.
- The VoiceXML gateway receives the call along with information about the dialed number.
- The VoiceXML gateway searches a database.
- If successful, it maps the dialed number to an URL.
- This URL is the location of the agent's main page (`kuoni.vxml`).
- The gateway retrieves the `kuoni.vxml` page together with associated files such as grammars and recorded audio from the HTTP server.
- These associated files may be cached on the VoiceXML gateway.

A VoiceXML Scenario

- The VoiceXML interpreter parses and executes the VoiceXML document.
- The interpreter steps through `kuoni.vxml` playing prompts, hearing responses and passing them on to a speech recognition engine.
- If necessary, additional VoiceXML documents and associated files are retrieved from the HTTP server.
- Recorded audio is served by specifying the URL of the WAV file.
- Communications between the voice gateway and the HTTP server follow standard HTTP protocols.

The W3C Speech Interface Framework



A Voice XML Fragment

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">
…
  <form id = "travel">
    <field name = "destination">                                <!-- VoiceXML  -->
      <prompt>
        Do you want to fly to
        <emphasis level = "strong"> New York </emphasis> or to  <!-- SSML -->
        <emphasis level = "strong"> Washington </emphasis>
      </prompt>
      <grammar mode = "voice" root = "destination-city">        <!-- SRGS -->
        <rule id = "destination-city">
          <one-of>
            <item tag = "NEW-YORK"> New York </item>            <!-- SI -->
            <item tag = "NEW-YORK"> Big Apple </item>
            <item tag = "WASHINGTON"> Washington </item>
            <item tag = "WASHINGTON"> The Capital </item>
          </one-of>
        </rule>
      </grammar>
    </field>
  </form>
</vxml>
```

VoiceXML

- VoiceXML is designed for creating audio dialogs that feature
  - recognition of spoken and DTMF key input,
  - recording of spoken input,
  - mixed initiative conversation,
  - synthesized speech,
  - digitized audio,
  - telephony.
- Its major goal is to bring the advantages of Web-based development and content delivery to interactive voice response applications.

SRGS

- The Speech Recognition Grammar Specification defines the syntax for representing grammars for use in speech recognition.
- The syntax of the grammar format is presented in two forms, an XML Form and an Augmented BNF Form.
- The specification makes the two representations mappable to allow automatic transformations between the two forms.

Example: SRGS

ABNF

```
$destination-city =
  to London {"LONDON"} |
  to Paris {"Paris"}
```

↔

XML format

```
<rule id = "destination-city">
  <one-of>
    <item tag = "LONDON">
      to London
    </item>
    <item tag = "PARIS">
      to Paris
    </item>
  </one-of>
</rule>
```

SSML

- The Speech Synthesis Markup Language specification provides a markup language for assisting the generation of synthetic speech in Web and other applications.
- SSML allows to control aspects of speech such as
  - pronunciation,
  - volume,
  - pitch,
  - rate

  across different synthesis-capable platforms.

Example: SSML

```
<prompt>
   <emphasis>Welcome</emphasis>
   to the Bird Seed Emporium.
   <audio src="rtsp://www.birdsounds.example.com/thrush.wav"/>
   We have
   <say-as interpret-as="number">250</say-as>
   kilogram drums of thistle seed for
   <say-as interpret-as="currency">$299.95</say-as>
   plus shipping and handling this month.
   <audio src="http://www.birdsounds.example.com/mourningdove.wav"/>
</prompt>
```

Semantic Interpretation

- The Semantic Interpretation language allows for attaching instructions to grammar rules that describe how to extract semantic information from recognised utterances.
- A speech recognition grammar processor searches for a best match.
- Recognising the uttered words is not enough.
- What is needed is the semantic result of the recognised input.
- SI tags provide a means to attach instructions to grammar rules.
- SI will be generating results that can be integrated into EMMA.

Example: Semantic Interpretation

```
<grammar mode = "voice" root = "destination-city">
  <rule id = "destination-city">
    <one-of>
      <item> New York </item>
      <item tag = "New York"> Big Apple </item>
      <item> Washington </item>
      <item tag = "Washington"> The Capital </item>
    </one-of>
  </rule>
 </grammar>
```

CCXML

- The Call Control eXtensible Markup Language provides telephony call control support for VoiceXML.
- CCXML allows VoiceXML to move calls around and connect them to dialog resources.
- The two languages are separate.
- They are not required in an implementation of either language.
- CCXML can be used as call control manager in any telephony system.

Example: CCXML

```
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0">
  <eventhandler>

    <!-- Lets handle the incoming call -->
    <transition event="connection.CONNECTION_ALERTING" name="evt">
      <log expr="'The caller ID is ' + evt.callerid + '.'"/>
      <if cond="evt.callerid == '8315551234'">
        <accept/>
      <else/>
        <reject/>
      </if>
    </transition>

    <!-- Lets handle the call being answered -->
    <transition event="connection.CONNECTION_CONNECTED">
    </transition>
  </eventhandler>
</ccxml>
```

Example: EMMA

- The Extensible MultiModal Annotation markup language is used for providing semantic interpretations for a variety of input modes:
  - speech,
  - natural language text,
  - graphical user interface,
  - and electronic ink input.
- The markup will be used as a standard data interchange format.
- The markup will be automatically generated by interpretation components to represent the semantics of users' inputs.

EMMA: Example

Utterance: “Zoom in here”

```
  <emma:interpretation>
     <command>
        <zoom><location/></zoom>
     </command>
</emma:interpretation>
```

Area circled by pen

```
<emma:interpretation >
    <area>(200,200), (200,400), (400,400),
    (400,200)</area>
</emma:interpretation>
```

Integration of information

```
<emma:interpretation>
   <command>
      <zoom>
         <area> (200,200), (200,400),
         (400,400), (400,200)</area>
      </zoom>
   </command>
</emma:interpretation>
```

Introduction to VoiceXML
3. VoiceXML: Dialogs, Forms and Fields

Rolf Schwitter

schwitt@ics.mq.edu.au

VoiceXML Documents

- A VoiceXML document forms a conversational finite state machine.
- The caller is always in one conversational state, or dialog, at a time.
- Each dialog determines the next dialog to transition to.
- Transitions are specified using URIs, which define the next document and dialog to use.
- Execution is terminated
  - when a dialog does not specify a successor, or
  - if it has an element that explicitly exits the conversation.

Dialogs

- Dialog elements present information and collect data.
- There are two kinds of dialog elements:
  - forms
  - menus.

Forms

- Forms collect values for a set of field item variables.
- A field specifies an input item to be gathered from the user.
- Grammars define the allowable inputs for fields.
- Platform throws events if the input is out-of-grammar.
- Actions are performed when field items are filled.

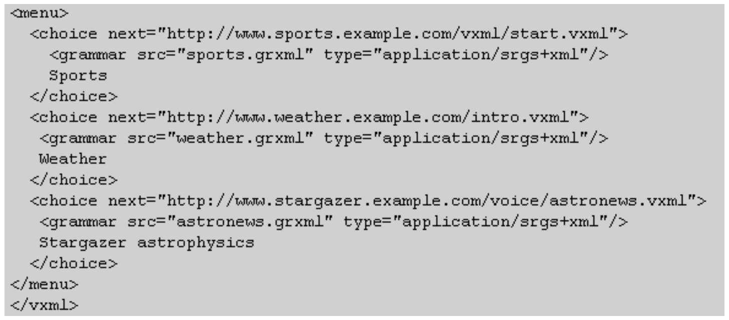
Example: Form

```
<form id = "pizza-ordering">

  <field name = "size">
    <prompt>  What pizza size would you like?  </prompt>
    <grammar src = "size.grxml"  type = "application/srgs+xml"/>
    <catch event = "help">
      You can choose a small, medium, large or extra-large pizza.
    </catch>
  </field>
  …
   <block>
   <submit next = "http://www..../pizza.py" namelist = "size ..."/>
   </block>
</form>
```

Menus

- Menus present the caller with a set of options.
- Transitions to another dialog are based on a choice.
- The <menu> element is a shortcut for a form with only one field.
- It is a convenient way to ask the user to pick one option from a list.

Example: Menu



```
<menu>
  <choice next="http://www.sports.example.com/vxml/start.vxml">
    <grammar src="sports.grxml" type="application/srgs+xml"/>
    Sports
  </choice>
  <choice next="http://www.weather.example.com/intro.vxml">
   <grammar src="weather.grxml" type="application/srgs+xml"/>
   Weather
  </choice>
  <choice next="http://www.stargazer.example.com/voice/astronews.vxml">
   <grammar src="astronews.grxml" type="application/srgs+xml"/>
   Stargazer astrophysics
  </choice>
</menu>
</vxml>
```

VoiceXML Elements

- So far, we used the following VoiceXML elements:
  - <vxml> top-level element in each VoiceXML document
  - <form> a dialog for presenting information and collecting data
  - <prompt> queues speech synthesis and audio output to user
  - <field> declares an input field in a form
  - <catch> catches an event
  - <block> a container of (non-interactive) executable code

VoiceXML Elements

– <menu>	a dialog for choosing amongst alternative destinations
– <choice>	defines a menu item
– <grammar>	specifies a speech recognition or DTMF grammar
– <goto>	goes to another dialog in the same or different document
– <submit>	submits values to a document server

Example: Are You Sleepy?

Computer:	Are you sleepy?
User:	<says nothing>
Computer:	Hey, don't sleep!
User:	Ooops.
Computer:	Say 'yes' or 'no'.
User:	Yes.
Computer:	So, you are sleepy, me too.

Example: Are You Sleepy?

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">

  <form id = "start">

    <field name = "answer">

      <prompt> Are you sleepy? </prompt>
      <grammar src = "yesno.grxml" type = "application/srgs+xml"/>

      <noinput> Hey, don't sleep!  </noinput>
      <nomatch> Say 'yes' or 'no'. </nomatch>
```

Example: Are You Sleepy?

```
      <filled>
        <if cond = "answer == 'yes'">
            So you are sleepy. Me too.
        <else/>
            So you are not sleepy. But I am.
        </if>
      </filled>

    </field>

  </form>

</vxml>

<!-- Note: "answer == 'yes' " is a JavaScript expression! -->
```

More VoiceXML Elements

- The last example introduced the following new elements:

– <noinput>	catches a noinput event
– <nomatch>	catches a nomatch event
– <filled>	actions executed when fields are filled
– <if>	simple conditional logic
– <else>	used in <if> element

SRGS: yesno.grxml

```
<?xml version = "1.0"?>

<grammar root = "main" version = "1.0">
  <rule id = "main" scope = "public">
    <one-of>
      <item tag = "yes"/> <ruleref uri = "#yes"/> </item>
      <item tag = "no"/>  <ruleref uri = "#no"/> </item>
    </one-of>
  </rule>
```

SRGS: yesno.grxml

```
 <rule id = "yes">
    <one-of>
      <item> yes </item>
      <item> yeah </item>
      <item> yep </item>
      <item> sure </item>
    </one-of>
  </rule>

  <rule id = "no">
    <one-of>
      <item> no </item>
      <item> not </item>
      <item> nope </item>
    </one-of>
  </rule>
</grammar>
```

SRGS: Comments

- The grammar is in XML form.
- The attribute "root" defines the root rule of the grammar.
- A rule definition is represented by the <rule> element.
- A public-scoped rule may be referenced in the rule definitions of other grammars.
- The <one-of> element identifies a set of alternative elements.
- Each alternative expansion is contained in a <item> element.
- Tags contain content for semantic interpretation.

Example: Weather Information

Computer:	Welcome to the weather information service.
Computer:	What state?
User:	Help!
Computer:	Please speak the state for which you want the weather.
User:	New South Wales.
Computer:	What city?
User:	Gosford.

Attributes and Values

- VoiceXML elements have <u>attributes</u> with specific values:

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">

  <form id = "weather_info">

    <block> Welcome to the weather information service. </block>

    <field name = "state">
      <prompt> What state? </prompt>
      <grammar src  = "state.grxml"
               type = "application/srgs+xml"/>
      <catch event = "help">
        Please speak the state for which you want the weather.
      </catch>
    </field>
```

Attributes and Values

```
    <field name = "city">
      <prompt> What city? </prompt>
      <grammar src = "city.grxml"
               type = "application/srgs+xml"/>
      <catch event = "help">
        Please speak the city for which you want the weather.
      </catch>
    </field>

    <block>
      <submit next = "cgi-bin/weather.py"
              namelist = "city state"/>
    </block>

  </form>

</vxml>
```

CENTRE FOR LANGUAGE TECHNOLOGY
MACQUARIE UNIVERSITY - SYDNEY

Introduction to VoiceXML
4. VoiceXML: Development Tools

Rolf Schwitter

schwitt@ics.mq.edu.au

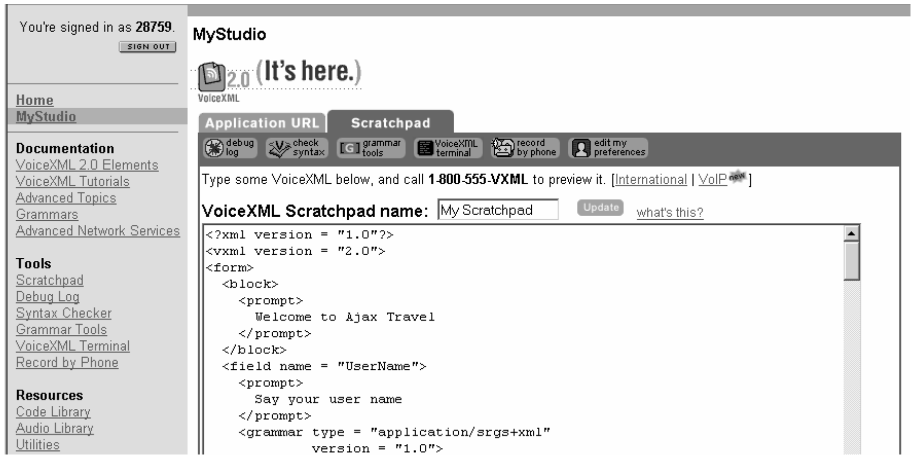
VoiceXML Implementations

- Web-based VoiceXML development tools:
  - Tellme at http://studio.tellme.com
  - BeVocal at http://café.bevocal.com
  - HeyAnita at http://www.heyanita.com
- VoiceXML platforms and graphical development tools:
  - Nuance at http://www.nuance.com
  - OptimTalk at http://www.optimsys.cz/news/
  - Open VXI VoiceXML at http://sourceforge.net/projects/openvxi/

Tellme Studio

- Tellme studio is a suite of Web-based VoiceXML development tools.
- Tellme studio enables you
  - to build, test, and publish VoiceXML applications
  - without buying or installing any hardware or software.
- By registering, you can develop your application for free.
- But check out first the VoiceXML elements supported by the Tellme voice interpreter.

MyStudio

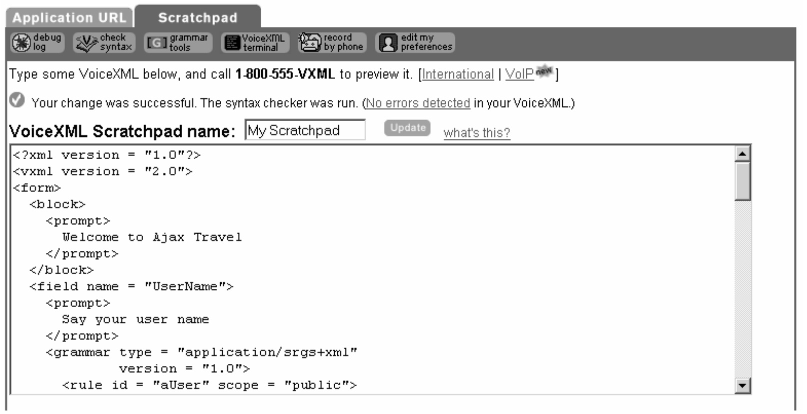


Type some VoiceXML below, and call **1-800-555-VXML** to preview it. [International | VoIP new]

VoiceXML Scratchpad name: My Scratchpad

```
<?xml version = "1.0"?>
<vxml version = "2.0">
<form>
  <block>
    <prompt>
      Welcome to Ajax Travel
    </prompt>
  </block>
  <field name = "UserName">
    <prompt>
      Say your user name
    </prompt>
    <grammar type = "application/srgs+xml"
            version = "1.0">
```

VoiceXML Scratchpad



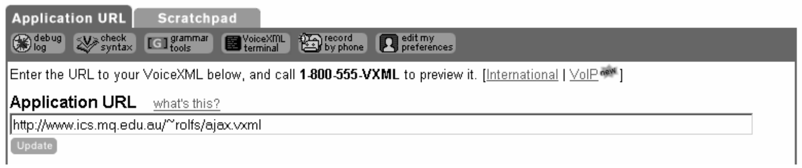
Type some VoiceXML below, and call **1-800-555-VXML** to preview it. [International | VoIP]

Your change was successful. The syntax checker was run. (No errors detected in your VoiceXML.)

VoiceXML Scratchpad name: My Scratchpad Update what's this?

```
<?xml version = "1.0"?>
<vxml version = "2.0">
<form>
  <block>
    <prompt>
      Welcome to Ajax Travel
    </prompt>
  </block>
  <field name = "UserName">
    <prompt>
      Say your user name
    </prompt>
    <grammar type = "application/srgs+xml"
             version = "1.0">
      <rule id = "aUser" scope = "public">
```

Application URL



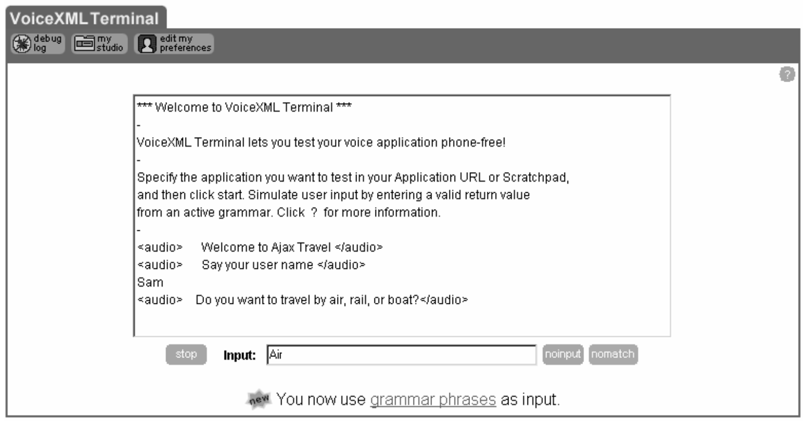
Enter the URL to your VoiceXML below, and call **1-800-555-VXML** to preview it. [International | VoIP]

Application URL what's this?

http://www.ics.mq.edu.au/~rolfs/ajax.vxml

Update

VoiceXML Terminal

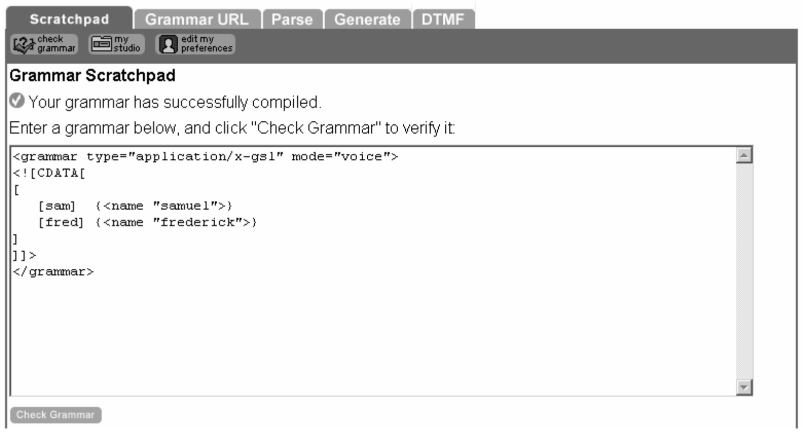


```
*** Welcome to VoiceXML Terminal ***
-
VoiceXML Terminal lets you test your voice application phone-free!
-
Specify the application you want to test in your Application URL or Scratchpad,
and then click start. Simulate user input by entering a valid return value
from an active grammar. Click ? for more information.
-
<audio>    Welcome to Ajax Travel </audio>
<audio>    Say your user name </audio>
Sam
<audio>   Do you want to travel by air, rail, or boat?</audio>
```

stop **Input:** Air noinput nomatch

You now use grammar phrases as input.

Grammar Scratchpad: GSL Grammar



Grammar Scratchpad

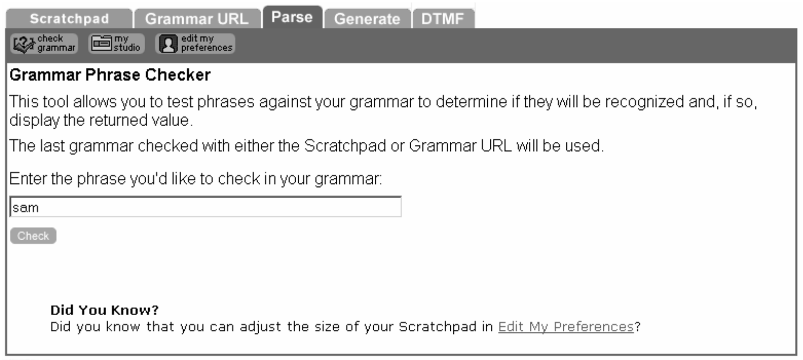
Your grammar has successfully compiled.

Enter a grammar below, and click "Check Grammar" to verify it:

```
<grammar type="application/x-gsl" mode="voice">
<![CDATA[
[
   [sam]  {<name "samuel">}
   [fred] {<name "frederick">}
]
]]>
</grammar>
```

Check Grammar

Grammar Phrase Checker



Grammar Phrase Checker

This tool allows you to test phrases against your grammar to determine if they will be recognized and, if so, display the returned value.

The last grammar checked with either the Scratchpad or Grammar URL will be used.

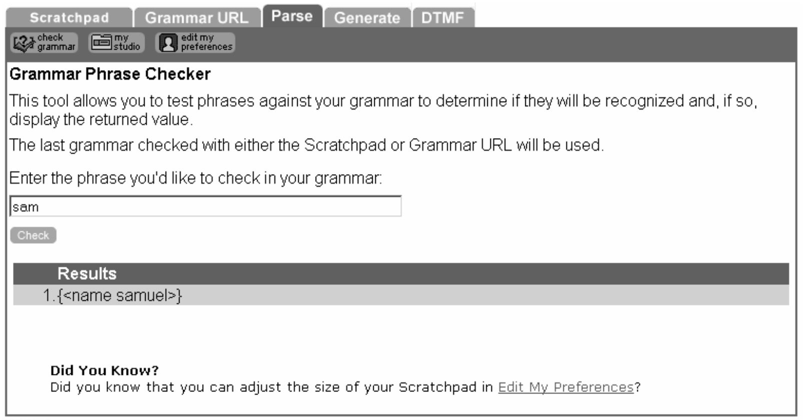
Enter the phrase you'd like to check in your grammar:

sam

Check

Did You Know?
Did you know that you can adjust the size of your Scratchpad in Edit My Preferences?

Grammar Phrase Checker: Results



Grammar Phrase Checker

This tool allows you to test phrases against your grammar to determine if they will be recognized and, if so, display the returned value.

The last grammar checked with either the Scratchpad or Grammar URL will be used.

Enter the phrase you'd like to check in your grammar:

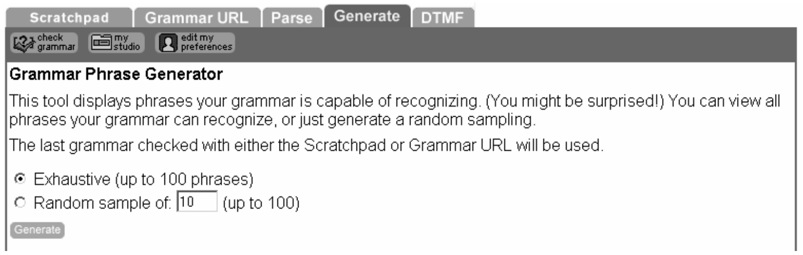
sam

Check

Results
1.{<name samuel>}

Did You Know?
Did you know that you can adjust the size of your Scratchpad in Edit My Preferences?

Grammar Phrase Generator



Grammar Phrase Generator

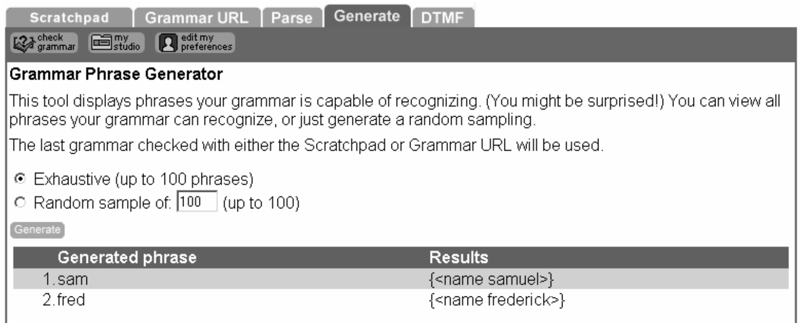
This tool displays phrases your grammar is capable of recognizing. (You might be surprised!) You can view all phrases your grammar can recognize, or just generate a random sampling.

The last grammar checked with either the Scratchpad or Grammar URL will be used.

● Exhaustive (up to 100 phrases)
○ Random sample of: 10 (up to 100)

Generate

Grammar Phrase Generator: Generated Phrase



Grammar Phrase Generator

This tool displays phrases your grammar is capable of recognizing. (You might be surprised!) You can view all phrases your grammar can recognize, or just generate a random sampling.

The last grammar checked with either the Scratchpad or Grammar URL will be used.

● Exhaustive (up to 100 phrases)
○ Random sample of: 100 (up to 100)

Generate

Generated phrase	Results
1.sam	{<name samuel>}
2.fred	{<name frederick>}

Connecting to Tellme Studio

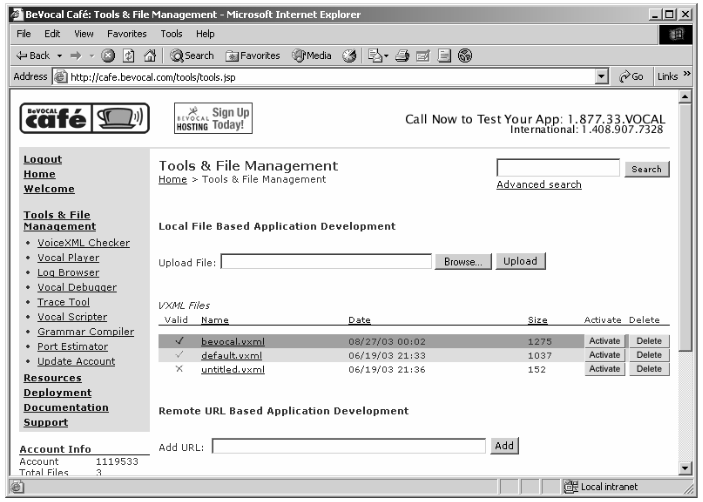
- To preview your application, you can use a phone and call
  - (408)-678-4465

  or you can use a soft phone such as X-Lite and call
  - sip:8005558965@sip.studio.tellme.com

BeVocal Café

- BeVocal Café offers similar functionality as Tellme Studio.
- BeVocal Café is available at
  - http://cafe.bevocal.com
- BeVocal Café comes with a
  - VoiceXML Checker
  - VocalScripter
  - Grammar Compiler.

Tools & File Management



BeVocal Café: Tools & File Management - Microsoft Internet Explorer

Address http://cafe.bevocal.com/tools/tools.jsp

Call Now to Test Your App: 1.877.33.VOCAL
International: 1.408.907.7328

Logout
Home
Welcome

Tools & File Management
- VoiceXML Checker
- Vocal Player
- Log Browser
- Vocal Debugger
- Trace Tool
- Vocal Scripter
- Grammar Compiler
- Port Estimator
- Update Account

Resources
Deployment
Documentation
Support

Account Info
Account 1119533
Total Files 3

Tools & File Management
Home > Tools & File Management

Local File Based Application Development

Upload File:

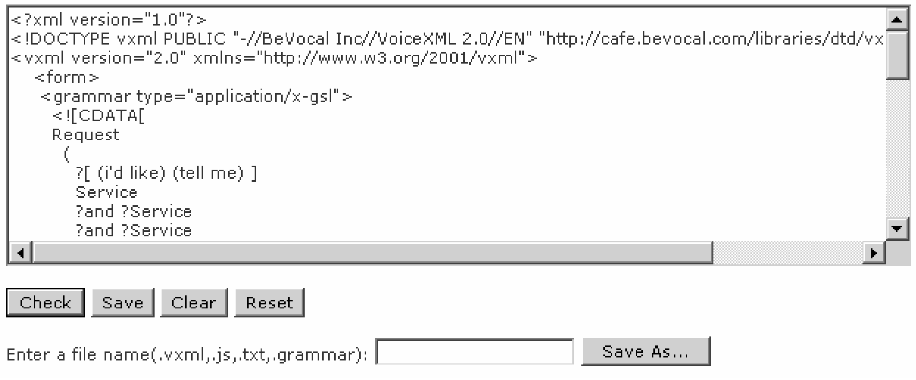
VXML Files

Valid	Name	Date	Size	Activate	Delete
✓	bevocal.vxml	08/27/03 00:02	1275	Activate	Delete
✓	default.vxml	06/19/03 21:33	1037	Activate	Delete
✕	untitled.vxml	06/19/03 21:36	152	Activate	Delete

Remote URL Based Application Development

Add URL:

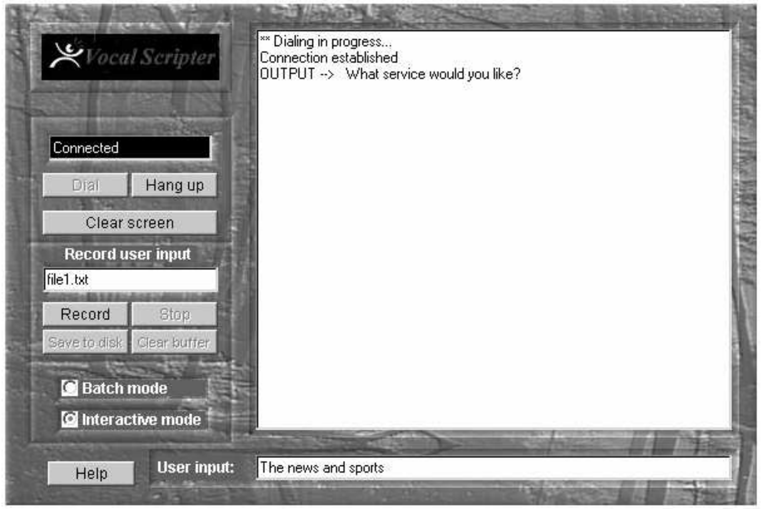
VoiceXML Checker



```
<?xml version="1.0"?>
<!DOCTYPE vxml PUBLIC "-//BeVocal Inc//VoiceXML 2.0//EN" "http://cafe.bevocal.com/libraries/dtd/vx
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form>
   <grammar type="application/x-gsl">
    <![CDATA[
    Request
     (
       ?[ (i'd like) (tell me) ]
       Service
       ?and ?Service
       ?and ?Service
```

Enter a file name(.vxml,.js,.txt,.grammar):

Vocal Scripter



```
** Dialing in progress...
Connection established
OUTPUT -->  What service would you like?
```

User input: The news and sports

OptimTalk

- OptimTalk
  - is a free VoiceXML platform for desktop computers,
  - consists of a set of libraries,
  - provides only a command line interface.
- These libraries support:
  - VoiceXML 2.0,
  - SRGS,
  - SSML,
  - Semantic Interpretation.

OptimTalk: Command Line Interface

- OptimTalk provides a command line interface
  ```
  C:\OptimTalk\bin>optimtalk_test.exe <vxmldoc>
  ```
  whereas
  ```
  <vxmldoc>
  ```
  is either a file name or a URI of a VoiceXML document.
- If you are fetching a document via a URI, the document needs to be on a Web server (html directory).
- Make sure that the VoiceXML document is readable.

OptimTalk: Input and Output Components

- Input and output components can be configured via a config. file:
  optimtalk_test.cfg
- For example:

```
# output component using Microsoft Speech API 5.1 to produce
# speech output. The output is also sent to the console
output=com.optimtalk.cons_and_sapi_output

# input component which uses keyboard for input
input=com.optimtalk.keyboard_input
```

CENTRE FOR LANGUAGE TECHNOLOGY
MACQUARIE UNIVERSITY - SYDNEY

Introduction to VoiceXML
5. VoiceXML: Control Flow

Rolf Schwitter

schwitt@ics.mq.edu.au

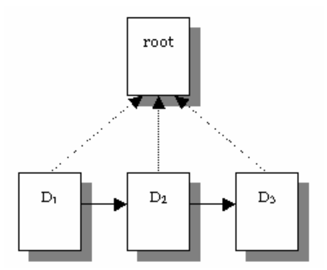
Application Root Document

- A VoiceXML application consists of one or more documents.
- These documents share an <u>application root document</u>.
- The application root document is (and remains) loaded
  - when the caller interacts with a document in the application
  - when the caller transitions between documents in the application.
- The application root document is unloaded
  - when the caller transitions to a document that is not in the application.

Application Root Document

- While the application root document is loaded
  - its variables are available to the other (leaf) documents
  - its grammars remain active for the duration of the application.

Transition between Documents



Transitioning between documents in an application

Executing a One-Document Application

- Normally, each document runs as an isolated application.
- Documents are composed of dialogs (= forms and menus).
- Document execution begins at the first dialog by default.
- As each dialog executes, it determines the next dialog.
- When a dialog does not specify a successor dialog, document execution stops.

Executing a One-Document Application

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">

  <var name = "hello" expr = "'Hello World!'"/>
  <form>
    <block>
      <value expr = "hello"/>
      <goto next = "#say_goodbye"/>
    </block>
  </form>
  <form id = "say_goodbye">
    <block>
      Goodbye!
    </block>
  </form>

</vxml>
```

Executing a Multi-Document Application

- If you want a multi-document application, you select
  - one document to be the application root document, and
  - the rest to be application leaf documents.
- Each leaf document names the root document in its <vxml> element using the "application" attribute.

Executing a Multi-Document Application

- During interpretation one of the following conditions always hold:
- Condition 1:

  The application root document is loaded and the caller is executing in it: there is no leaf document.
- Condition 2:

  The application root document and a single leaf document are both loaded and the caller is executing in the leaf document.

Executing a Multi-Document Application

- Application root document (app-root.vxml)

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">
```

- Leaf document (leaf.vxml)

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml"
 application = "app-root.vxml">
```

Example Dialog

Computer:	Would you like to say Ciao?
Caller:	Si. <"yes" or "no" is expected here>
Computer:	I did not understand what you said.
	<a platform-specific default message>
Computer:	Shall we say Ciao?
Caller:	Ciao. <"yes" or "no" is expected here>
Computer:	I did not understand what you said.
Caller:	Operator.
Computer:	<goes to operator.vxml, which transfers the call>

Leaf Document (leaf.vxml)

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml"
 application = "app-root.vxml">

<form id = "say_goodbye">
  <field name = "answer">
    <grammar type = "application/srgs+xml"
             src = "/grammars/boolean.grxml"/>
    <prompt count = "1">
      Would you like to say <value expr = "application.bye"/>?
    </prompt>
    <prompt count = "2">
      Shall we say <value expr = "application.bye"/>?
    </prompt>
```

Leaf Document (leaf.vxml)

```
      <filled>
        <if cond = "answer">
          <exit/>
        </if>
        <clear namelist = "answer"/>
      </filled>
     </field>
    </form>
  </vxml>
```

Application Root Document (app-root.vxml)

```
<?xml version = "1.0"?>
<vxml version = "2.0" xmlns = "http://www.w3.org/2001/vxml">

  <var name = "bye" expr = "'Ciao'"/>

  <link next = "operator.vxml">
     <grammar type = "application/srgs+xml"
              root = "root"
              version = "1.0">
        <rule id = "root" scope = "public">
          operator
        </rule>
     </grammar>
   </link>

</vxml>
```

Comments

- In our example:
  1. The "leaf.vxml" document is loaded first.
  2. The "app-root.vxml" document is loaded.
  3. The variable "bye" is created in "app-root.vxml".
  4. The link "operator.vxml" is defined in "app-root.vxml".
  5. The dialog starts in the "say_goodbye" form of "leaf.vxml".

Benefits to Multi-Document Applications

- Root variables <var> are available for use by the leaf documents.
- Root document <property> elements can be used to specify default values for properties used in the leaf documents.
- Property values affect platform behavior, for example:
  - recognition properties (confidencelevel, sensitivity, etc)
  - prompt and collect properties (bargein, timeout, etc)
  - fetching properties (fetchaudio, fetchtimeout).

Benefits to Multi-Document Applications

- ECMAScript code can be defined in root document element <script> and used in the leaf documents.
- Root document <catch> elements define default event handling for the leaf documents.
- If a root document has a document-level link <link>, its grammars are active when the user is in a leaf document.

Form Items

- Form items are visited by a form interpretation algorithm (FIA).
- There are two types of form items:
  - input items (e.g. <field>)
  - control items (e.g. <block>)
- Input items direct the FIA to gather a result for a specific element.
- Control items tell the FIA to execute code or to initialize a specific behaviour.

Input Items

- An <u>input item</u> specifies a form item variable (e.g. name = "answer ").
- Input items consists of:

– <field>	declares an input field in a form
– <record>	records an audio sample
– <transfer>	transfers a caller to another destination
– <object>	interacts with a custom extension
– <subdialog>	invokes another dialog as a subdialog

Control Items

- There are two types of <u>control items</u>:
  - <block> executes a sequence of procedural statements
  - <initial> controls the initial interaction in mixed initiative.
- The <block> control item has an implicit form item variable:
  - it is set to true just before the block is interpreted.
- The <initial> control item has an explicit form item variable:
  - it is set to true, if at least one input item variable is filled.

Form Interpretation Algorithm

- Forms are interpreted by an implicit form interpretation algorithm.
- The form interpretation algorithm has a main loop that
  - selects a form item
  - and then visits it.
- The selected form item is
  - the first in document order
  - whose guard condition is not satisfied.

Form Interpretation Algorithm

- Interpreting a form item generally involves:
  - selecting and playing one or more prompts
  - collecting a user input or throwing of some event
  - interpreting any <filled> actions.

Form Interpretation Algorithm

- The form interpretation algorithm ends
  - when it interprets a transfer of control statement (e.g. <goto> or <submit>)
  - when no form item remains to select (implied <exit>).

Customizing the Form Interpretation Algorithm

- The FIA can be customized in several ways:
  - assigning a value to a form item variable (<assign>)
  - setting a form item variable to undefined (<clear>)
  - specifying the next form item to visit (<goto>).

CENTRE FOR LANGUAGE TECHNOLOGY
MACQUARIE UNIVERSITY - SYDNEY

Introduction to VoiceXML
6. VoiceXML: Grammars

Rolf Schwitter

schwitt@ics.mq.edu.au

Grammar Standard

- No standard SR grammar format was available for VoiceXML 1.0.
- Voice browser developers had to define the grammar and format.
- This problem was rectified with the

    Speech Recognition Grammar Specification

  introduced with VoiceXML 2.0.
- Check: http://www.w3.org/TR/2004/REC-speech-grammar-20040316/

Grammar Formats

- The Speech Recognition Grammar Specification provides two formats:

    XML and ABNF (a plain text representation).
- VoiceXML 2.0 platforms <u>must</u> support the XML format.
- VoiceXML 2.0 platforms <u>should</u> also support the ABNF format.
- Both grammars have the power of a Context Free Grammar[1].
- The two formats are automatically mappable.

[1] A grammar processor that does not support recursive grammars has the expressive power of a Finite State Machine or regular language.

XML Format

```
<?xml version = "1.0"?>
<grammar mode = "voice" xml:lang = "en-US" root = "name"
         type = "application/srgs+xml" version = "1.0">

  <rule id = "name">
    <ruleref uri = #firstName"/>
    <ruleref uri = #lastName"/>
  </rule>

  <rule id = "firstName">
    <one-of>
      <item> Marc </item>
      <item> John </item>
    </one-of>
  </rule>
```

XML Format

```
    <rule id = "lastName">
      <one-of>
        <item> Miller </item>
        <item> Yates </item>
        <item> King </item>
      </one-of>
    </rule>

  </grammar>
```

ABNF Format

```
<grammar type = "application/srgs">

  #ABNF 1.0;
  language en-US;
  mode = voice;

  root $name;
    $name      = $firstName $lastName;
    $firstName = Marc | John;
    $lastName  = Miller | Yates | King;

</grammar>
```

Grammar Formats

- VoiceXML 2.0 platforms <u>may</u> support vendor-dependent formats:
  – Nuance Grammar Specification Language (GSL),
  – Java Speech Grammar Format (JSGF).
- OptimTalk supports the XML format.
- Tellme Studio and BeVocal also support GSL.

Using Grammars

- The <grammar> element is used to provide a speech grammar.
- The <grammar> element can be used to specify an inline or an external grammar.
- However, we can distinguish three uses of grammars:
  – built-in grammar
  – inline grammar or
  – external grammar (**`<grammar src = "URI">`**).

Built-in Grammar

```
…
<field name = "ticket_num" type = "digits">
  <prompt> Read the 12 digit number from your ticket. </prompt>
  <help> The 12 digit number is to the lower left. </help>
  <filled>
    <if cond = "ticket_num.length != 12">
      <prompt> Sorry, I didn't hear exactly 12 digits. </prompt>
      <assign name = "ticket_num" expr = "undefined"/>
    </if>
  </filled>
</field>
```

Inline Grammar

```
<grammar mode="voice" xml:lang="en-US" version="1.0" root="command">
  <!-- Command is an action on an object -->
  <!-- e.g. "open a window" -->
  <rule id="command" scope="public">
    <ruleref uri="#action"/> <ruleref uri="#object"/>
  </rule>

  <rule id="action">
    <one-of>
      <item> open </item>
      <item> close </item>
      <item> delete </item>
      <item> move </item>
    </one-of>
  </rule>

  <rule id="object">
   <item repeat="0-1">
      <one-of> <item> the </item> <item> a </item> </one-of>
    </item>
    <one-of>
      <item> window </item>
      <item> file </item>
      <item> menu </item>
    </one-of>
  </rule>
</grammar>
```

External Grammar

```
<?xml version = "1.0"?>
<grammar version = "1.0">
  <rule id = "drink" scope = "public">
    <one-of>
      <item tag = "coke"> coca cola </item>
      <item> coke </item>
      <item> sprite </item>
      <item> fanta </item>
    </one-of>
  </rule>
</grammar>
```

Operators

- The SRGS defines a set of operators.
- Operators allow us to recognise complex word patterns.
- For example, the caller might say one of the following things:
  - Um, my name is Marc Miller.
  - My name is Miller.
  - Um, yeah, well, I'am Marc Miller.

  but we are only interested in the last name.

Repeat

```
<rule id = "name">
  <item repeat = "0-1"> um
    <item repeat = "0-1"> yeah well </item>
  </item>
  <one-of>
    <item repeat = "0-1"> my name is </item>
    <item repeat = "0-1"> I'm </item>
  </one-of>
  <item repeat = "0-1">
    <ruleref uri = "#firstName"/>
  </item>
  <ruleref uri = "#lastName"/>
</rule>
```

Zero or More

- Matching zero or more instances of a token:

```
<rule id = "mood">
  I am <item repeat = "0-"> very </item> lucky
</rule>
```

- For example:
  – I am lucky.
  – I am very lucky.
  – I am very very very lucky.

One or More

- Matching one or more instances of a token:

```
<rule id = "mood">
  I am <item repeat = "1-"> very </item> lucky
</rule>
```

- Examples:
  – I am very lucky.
  – I am very very very lucky.
  – But not: I am lucky.

Token Ranges and Exact Matches

```
<rule id = "increase">
  <item repeat = "1-5"> Please </item>
  increase my salary
</rule>

<rule id = "increase">
  <item repeat = "5"> Please </item>
  increase my salary
</rule>

<rule id = "increase">
  <item repeat = "5-"> Please </item>
  increase my salary
</rule>
```

Nuance Grammar Specification Language (GSL)

- GSL is still widely used in industry.
- Nuance provides development tools (Nuance Grammar Builder).
- GSL grammars can be compiled.
- Probabilities can be assigned to phrases.
- But GSL syntax uses characters that are reserved by XML.
- Therefore, in-line grammars must be protected (CDATA section).

GSL Grammar

- Here is an in-line grammar in GSL format:

```
<grammar type = "application/x-gsl" mode = "voice">
  <![CDATA[
  [
    [(new york) (big apple)]   {<destination "NEW YORK">}
    [washington (the capital)] {<destination "WASHINGTON">}
  ]
  ]]>
</grammar>
```

GSL Grammar

- The value of the "name" attribute (destination) is set to the value returned by the grammar.

```
<form>
  <field name = "destination">
  <prompt> Do you want to fly to New York or Washington? </prompt>
  <grammar type = "application/x-gsl" mode = "voice">
    <![CDATA[
     [[(new york) (big apple)]   {<destination "new york">}
      [washington (the capital)] {<destination "washington">}]
    ]]>
  </grammar>
…
```

GSL Grammar at Work

```
<?xml version = "1.0"?>
<vxml version = "2.0">

  <form>
    <field name = "destination">
    <prompt> Do you want to fly to New York or Washington? </prompt>
     <grammar type = "application/x-gsl" mode = "voice">
       <![CDATA[
         [[(new york) (big apple)] {<destination "new york">}
          [washington (the capital)] {<destination "washignton">}]
         ]]>
     </grammar>
```

GSL Grammar at Work

```
      <catch event = "nomatch noinput">
        <reprompt/>
      </catch>
      <filled>
        <prompt> You said <value expr = "destination"/> </prompt>
      </filled>
      </field>
    </form>

  </vxml>
```

GSL – Yes/No Grammar

```
YES_NO [
  [ yes
    yeah
    yup
    sure
    okay
    correct
    right
    ( ?( ?yes that's ) [ right correct ] )
    ( ?yes it is )
    ( you got it )
    ( yes i do )
    ( yes i would )
    ( yes it is correct )
  ] {return("yes")}
```

GSL – Yes/No Grammar

```
  [  no
     nope
     incorrect
     ( no way )
     ( no it isn't )
     ( ?no [ it's that's ] not [ correct right ] )
     ( ?no it isn't )
     ( ?no it is not )
     ( ?no it's not )
     ( no i don't )
     ( no i do not )
     ( no i wouldn't )
   ] {return("no")}
 ]
```

Introduction to VoiceXML
7. VoiceXML: Mixed Initiative

Rolf Schwitter

schwitt@ics.mq.edu.au

Dialog Styles

Mixed Initiative	
Application Directed Forms Menus	User Directed Dictation Query Command and Control

Form Filling Dialog Model

- In a form filling dialog model
  the application typically prompts the caller
  for discrete pieces of information
  in a pre-determined order.
- In this model, the VoiceXML application mainly
  consists of a number of call states
  that collect input from the caller.

Form Filling Dialog

Computer:	Welcome to ACME Travel.
Computer:	Where are you flying from?
Caller:	I wanna fly <u>from San Francisco, California</u>.
Computer:	Where do you want to go to?
Caller:	<u>To Boston, Massachusetts</u>.
Computer:	Okay, to summarize, you'd like to fly from San Francisco, California to Boston, Massachusetts. Is that correct?
Caller:	Yes.

Problems

- For callers, "form filling" can become quite cumbersome.
- Especially when callers are accustomed to provide multiple pieces of information
  - in succession without interruption of intermediary prompts
  - in a different order than specified by the application.
- For example:
  - (A + B + C)
  - (B + A) + C

Mixed Initiative Dialog Model

- In a mixed initiative dialog model
  - the call flow

    can be directed by the caller or by the application
  - the application

    collects pieces of information in a single call state.
- Caution: VoiceXML does not allow for true mixed-initiative utterances.
- However, it is possible to approximate this dialog style.

Mixed Initiative Dialog

Computer:	Welcome to ACME travel.
Computer:	Please tell me your starting and destination cities.
Caller:	I wanna fly <u>from San Francisco, California</u> <u>to Boston, Massachusetts</u>.
Computer:	Okay, to summarize, you'd like to fly from San Francisco, California to Boston, Massachusetts. Is that correct?
Caller:	Yes.

Mixed Initiative Dialogs in VoiceXML

- A mixed initiative dialog in VoiceXML is essentially a way to

  prompt the caller for multiple pieces of information at once

  have a grammar construct that allows this to happen

  and then fall back on machine-directed dialog (if needed)

  that sequentially walks the caller through any questions they neglected to answer in their original response.

Implementing Mixed Initiative Dialogs

- The following things need to be done:
  - define subgrammars to collect each piece of information
  - define a form level grammar that uses the subgrammars to collect the information
  - define a mixed initiative dialog that collects input from the caller.
- The mixed initiative dialog can be built on top of a form-filling dialog.

Defining the Subgrammar

- In the ACME example, the origin and destination are similar pieces of information:
  - from San Francisco, California
  - to Boston, Massachusetts
- Therefore, a single subgrammar can be defined for this data.

Defining the Subgrammar (airports.gsl)

```
Airports [ [ (albuquerque new_mexico) (a b q) ]
              { return(albuquerque_nm) }
           [ (boston massachusetts) (b o s) ]
              { return(boston_ma) }
           [ (charlotte north_carolina) (c l t) ]
              { return(charlotte_nc) }
           [ (los angeles california) (l a x) ]
              { return(los_angeles_ca)}
           [ (portland oregon) (p d x) ]
              { return(portland_or) }
           [ (san francisco california) (s f o) ]
              {return(san_francisco_ca)}
           [ (seattle washington) (s e a) ]
              {return(seattle_wa) }
         ]
```

Defining the Form Level Grammar

- In the next step, we need to define the form level grammar that utilizes the Airport subgrammar.
- The caller should be able to utter sentences such as:
  - I wanna fly from S F O to Boston, Massachusetts.
  - I want to go to Albuquerque, New Mexico from San Antonio, Texas.
  - From Cleveland, Ohio. To Portland, Oregon.

Defining the Form Level Grammar (travel.gsl)

```
( ?( i [(want to) wanna] [ go fly ] )
 [ ( from Airports:x ) { <from $x> }
   ( to Airports:y ) { <to $y> }
   ( from Airports:x to Airports:y ) { <from $x> <to $y> }
   ( to Airports:y from Airports:x ) { <from $x> <to $y> }
 ] )
```

Defining the Mixed Initiative Dialog

- A mixed initiative dialog in VoiceXML consists of the following parts:
  - grammars defined at form level (just discussed)
  - an <initial> element that prompts for form-wide information
  - a field with subgrammar for each piece of information to collect
  - a confirmation field.

Example: Mixed Initiative

```
<vxml version="2.0">

<!-- helper script that maps city/state names to audio/tts -->
<script src="citystate.js"/>
<script>
  var csobj = new CityStateReader();
</script>

<!-- application entrypoint -->
<form id="start">
<block>
    <audio src="wav/01.wav">Welcome to ACME travel.</audio>
        <break time="500ms"/>
    <goto next="#get_origin_dest"/>
</block>
</form>

<!-- retrieves the origin and destination using mixed initiative -->
<form id="get_origin_dest">
   <property name="confidencelevel" value="0.4"/>
   <grammar type="application/x-gsl" mode="voice" src="travel.gsl"/>

   <catch event="nomatch noinput">
      <audio src="wav/04.wav">sorry, i didn't catch that.</audio>
      <reprompt/>
   </catch>
```

Example: <initial> Element

```
<!-- designates the initial state in a mixed initiative dialog -->
<initial name="init">
   <prompt>
     <audio src="wav/03.wav">
        please tell me your starting and destination cities.
     </audio>
   </prompt>
   <catch event="nomatch noinput">
     <audio src="wav/04.wav">sorry, i didn't catch that.</audio>
     <audio src="wav/11.wav">
        please say where you'd like to go to and from.
     </audio>
   </catch>
   <catch event="nomatch noinput" count="2">
     <audio src="wav/04.wav">sorry, i didn't catch that.</audio>
     <assign name="init" expr="true"/>
     <reprompt/>
   </catch>
   <help>
      <audio>
        to book a flight you need to specify your origin and destination cities.
      </audio>
      <audio>
        for example, you can say, from san francisco,
        california to boston massachusetts.
      </audio>
   </help>
 </initial>
```

Example: Origin

```
<!-- retrieve origin in case it didn't happen in initial state -->
<field name="origin" slot="from">
  <grammar type="application/x-gsl" mode="voice" src="airports.gsl"/>
  <prompt>
    <audio src="wav/06.wav">where are you flying from?</audio>
  </prompt>
  <filled>
    <prompt>
       <value expr="csobj.GetCSTTS(origin)"/>
    </prompt>
  </filled>
</field>
```

Example: Destination

```
<!-- retrieve destination in case it didn't happen in initial state -->
<field name="to">
  <grammar type="application/x-gsl" mode="voice" src="airports.gsl"/>
  <prompt>
    <audio src="wav/08.wav"> where do you want to go? </audio>
  </prompt>
  <filled>
    <prompt>
       <value expr="csobj.GetCSTTS(to)"/>
    </prompt>
  </filled>
</field>
```

Example: Confirmation

```
<!-- confirm origin and destination -->
    <field name="confirm" type="boolean">
<prompt>
  <audio src="wav/10.wav">
     Okay! To summarize, you'd like to fly from
  </audio>
  <prompt>
     <value expr="csobj.GetCSTTS(origin)"/>
  </prompt>
  <audio src="wav/05.wav"> to </audio>
  <prompt>
     <value expr="csobj.GetCSTTS(to)"/>
  </prompt>
      <audio>is that correct?</audio>
</prompt>
    <catch event="nomatch noinput">
      Sorry I didn't get that.
      <reprompt/>
    </catch>
<filled>
     <if cond="confirm">
                    <goto next="#bookit"/>
     <else/>
                    <clear/>
            </if>
       </filled>
 </field>
```

Example: Move along …

```
   </form>

   <!-- move along now that origin and dest have been collected... -->
   <form id="bookit">
   <block>
      <audio>booking your flight</audio>
      <goto next="#start"/>
   </block>
   </form>
</vxml>
```

CENTRE FOR LANGUAGE TECHNOLOGY
MACQUARIE UNIVERSITY - SYDNEY

Introduction to VoiceXML
8. VoiceXML: Scripting

Rolf Schwitter

schwitt@ics.mq.edu.au

What is JavaScript?

- JavaScript is an object-oriented scripting language.
- Client-side JavaScript
  - is an implementation of ECMAScript
  - is usually embedded directly in HTML pages
  - is interpreted.
- Sever-side JavaScript
  - is used with Web servers such as Apache (mod_javascript)
  - can access the file system and connect to relational databases
  - is compiled.

JavaScript and VoiceXML

- VoiceXML has very few programming features of its own.
- JavaScript is the required scripting language for VoiceXML.
- VoiceXML documents can contain JavaScripts in two contexts:
  - as values of many attributes
  - as arbitrary code in <script> elements.
- VoiceXML documents can refer to external JavaScripts via the "src" attribute of the <script> element.

Special Characters

- JavaScript has 3 characters which have also meaning in VoiceXML.

Character	Escape sequence
<	&lt;
>	&gt;
&	&amp;

- You must escape these characters or wrap them in a CDATA section.

JavaScript Expressions in Attributes

- Many VoiceXML attributes use JavaScript expressions directly.
- For instance, the "expr" attribute of elements such as
  - <var>
  - <field>
  - <assign>
- For example:

```
<var name = "one" expr = "1"/>
<field name = "two" expr = "one + 1">
<assign name = "result" expr = "Math.sqrt(a)"/>
```

Example: Path Construction

- The following example assigns a value to the variable "ui_path":

```
<var name = "ui_path">
<assign name = "ui_path" expr = "'ui/'" />
```

- The next assignment references "ui_path" and uses the JavaScript concatenation operator "+":

```
<assign name = "intro_path" expr = "ui_path + 'welcome.wav'" />
```

JavaScript within Script Elements

- A <script> element may occur
  - in the <vxml> and <form> elements, or
  - in executable content (in <filled>, <if>, <block>, <catch>).
- The VoiceXML <script> element does not have a language type attribute.

Evaluating Scripts

- Scripts in the <vxml> element are evaluated just after the document is loaded, along with the <var> elements, in document order.
- Scripts in the <form> element are evaluated in document order, along with <var> elements and form item variables, each time execution moves into the <form> element.
- A <script> element in executable content is executed, like other executable elements, as it is encountered.

Accessing Variables

- A variable declared in VoiceXML is accessible from JavaScript:

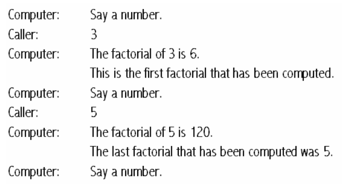
```
<block> <var name = "iCurrentMonth" />
  <script>
    var d = new Date();
    iCurrentMonth = d.getUTCMonth()); // Universal Time Conversion
  </script>

  <prompt>
    <audio expr = "'ui/months/' + iCurrentMonth + '.wav'"/>
  </prompt>
</block>
```

Example: Factorial Dialog

Computer:	Say a number.
Caller:	3
Computer:	The factorial of 3 is 6.
	This is the first factorial that has been computed.
Computer:	Say a number.
Caller:	5
Computer:	The factorial of 5 is 120.
	The last factorial that has been computed was 5.
Computer:	Say a number.

Example: Factorial in VoiceXML

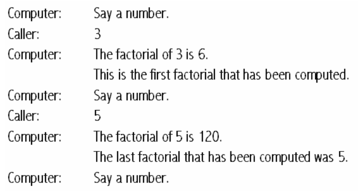
```
…
<form id = "form">
  <var name = "lastresult"
       expr = "'This is the first factorial
                that has been computed.'"/>

  <field name = "n" type = "number">
    <prompt> Say a number. </prompt>
    <grammar type = "application/srgs+xml"
             src = "/grammars/number.grxml"/>
```



Computer:	Say a number.
Caller:	3
Computer:	The factorial of 3 is 6.
	This is the first factorial that has been computed.
Computer:	Say a number.
Caller:	5
Computer:	The factorial of 5 is 120.
	The last factorial that has been computed was 5.
Computer:	Say a number.

Example: Factorial in VoiceXML

```
    <filled>
      <prompt> The factorial of <value expr = "n"/> is
        <value expr = "factorial(n)"/>
        <value expr = "lastresult"/>.
      </prompt>

      <assign name = "lastresult"
        expr = "'The last factorial
                 that has been computed
                 was ' + n + '.'" />

      <clear namelist = "n"/>
    </filled>
  </field>
</form>
```



Computer:	Say a number.
Caller:	3
Computer:	The factorial of 3 is 6.
	This is the first factorial that has been computed.
Computer:	Say a number.
Caller:	5
Computer:	The factorial of 5 is 120.
	The last factorial that has been computed was 5.
Computer:	Say a number.

Example: Speaking Clock

```
<?xml version = "1.0"?>
<vxml version = "2.0">
  <form>
    <var name = "hours"/>
    <var name = "minutes"/>
    <var name = "seconds"/>
    <block>
      <script>
        var d   = new Date();
        hours   = d.getHours();
        minutes = d.getMinutes();
        seconds = d.getSeconds();
      </script>
    </block>
```

Example: Speaking Clock

```
    <field name = "hear_another" type = "boolean">
      <prompt>
        The time is <value expr = "hours"/> hours,
        <value expr = "minutes"/> minutes, and
        <value expr = "seconds"/> seconds.
      </prompt>
      <prompt>
        Do you want to hear another time?
      </prompt>
```

Example: Speaking Clock

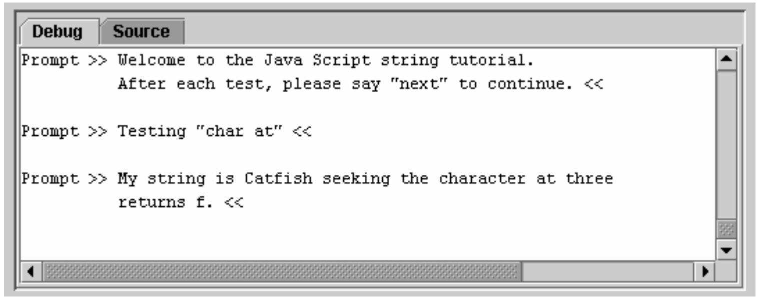
```
      <filled>
        <if cond = "hear_another">
          <clear/>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

Example: JavaScript and Grammars

```
<rule id = "toppings" scope = "public">
  <tag> $ = new Array(); </tag>
  <item repeat = "1-">
    <ruleref uri = "#topping"/>
    <tag> $.push($topping); </tag>
  </item>
</rule>

<rule id = "topping" scope = "public">
  <one-of>
    <item> cheese </item>
    <item> ham </item>
    <item> pepperoni </item>
    <item> mushrooms </item>
  </one-of>
</rule>
…
```

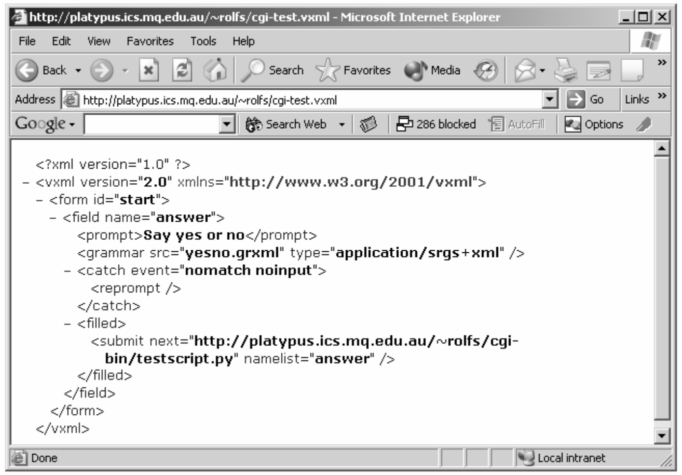
Example: JavaScript String Tutorial

- The tutorial demonstrates some JavaScript string object methods.



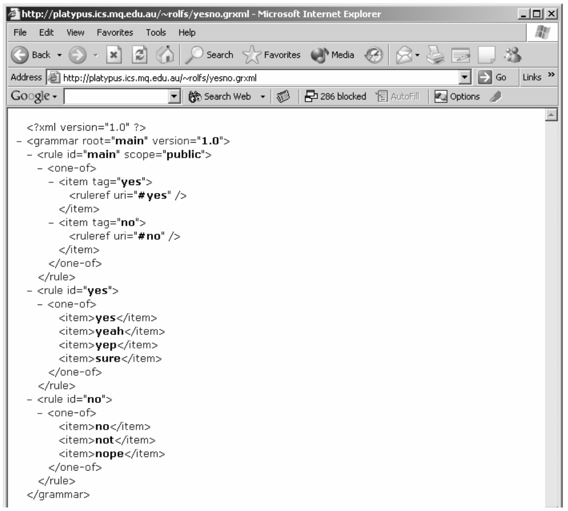
```
Prompt >> Welcome to the Java Script string tutorial.
          After each test, please say "next" to continue. <<

Prompt >> Testing "char at" <<

Prompt >> My string is Catfish seeking the character at three
          returns f. <<
```

VoiceXML and CGI Scripting


http://platypus.ics.mq.edu.au/~rolfs/cgi-test.vxml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://platypus.ics.mq.edu.au/~rolfs/cgi-test.vxml Go Links
Google Search Web 286 blocked AutoFill Options
Done
Local intranet

```
  <?xml version="1.0" ?>
- <vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  - <form id="start">
    - <field name="answer">
        <prompt>Say yes or no</prompt>
        <grammar src="yesno.grxml" type="application/srgs+xml" />
      - <catch event="nomatch noinput">
          <reprompt />
        </catch>
      - <filled>
          <submit next="http://platypus.ics.mq.edu.au/~rolfs/cgi-
            bin/testscript.py" namelist="answer" />
        </filled>
      </field>
    </form>
  </vxml>
```

Yes/No Grammar


http://platypus.ics.mq.edu.au/~rolfs/yesno.grxml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://platypus.ics.mq.edu.au/~rolfs/yesno.grxml Go Links
Google Search Web 286 blocked AutoFill Options

```
  <?xml version="1.0" ?>
- <grammar root="main" version="1.0">
  - <rule id="main" scope="public">
    - <one-of>
      - <item tag="yes">
          <ruleref uri="#yes" />
        </item>
      - <item tag="no">
          <ruleref uri="#no" />
        </item>
      </one-of>
    </rule>
  - <rule id="yes">
    - <one-of>
        <item>yes</item>
        <item>yeah</item>
        <item>yep</item>
        <item>sure</item>
      </one-of>
    </rule>
  - <rule id="no">
    - <one-of>
        <item>no</item>
        <item>not</item>
        <item>nope</item>
      </one-of>
    </rule>
  </grammar>
```

Python Script (testscript.py)

```
#!/usr/local/bin/python

import cgi

form = cgi.FieldStorage()

print "Content-type: text/xml\n\n"

if (form["answer"].value == 'yes') :
    print "<vxml version=\"2.0\" \
    xmlns = \"http://www.w3.org/2001/vxml\"> \
    <form><block>You just said yes</block></form></vxml>"
else:
    print "<vxml version=\"2.0\" \
    xmlns = \"http://www.w3.org/2001/vxml\"> \
    <form><block>You just said no</block></form></vxml>"
```