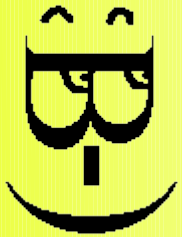
ChatScript



by

Bruce Wilcox

Agenda

Section 0 – Introduction

- **What and Who of ChatScript**

Section 1 – Fundamentals of Scripting

- Overview of Meaning / ChatScript
- Patterns in depth
- Topics, Output, Data, Functions

Section 2 – How does the engine work

What is ChatScript?

My personal NL research project (Open Source)

NL toolkit pipeline

Dialog manager

Expert system for understanding meaning

Expert system for planning

Who uses ChatScript

Hundreds of people in 25+ countries

Basic of NL tech for some companies:

Sensory, Inc (device control)

YourMd (medical diagnosis)

ToyTalk (own rewrite for children's software/toys)

AISoy & Engineered Arts (robot control & personality)

Kore, Inc (business bots)

What have we built with CS?

Conversational chatbots

- Ben + Meg (ESL conversation)
- K9 (for a robotic dog toy)
- Angela 18-year old female cat (60M+ downloads)
- Bodo (Star Wars alien demo)
- Big Bad Wolf & Pig (interactive stage play demo for kids with asthma)
- Kora (control over business bots)
- Morgan (Director Morgan Spurlock' clone)
- Rose + Suzette (Loebner winning chatbots)
- Dark Design ARG (multiple human bots + 1 parrot)
- Pearl (receptionist to experts)

What else have we/others built

Amazon improved product search demo

Wiki demo indexing sentences in Wikipedia

Omni command statistics analysis

Genie commanded basic phone abilities

NL controlled data retrieval from Dow Jones Database

Patient simulator for doctor training (Ohio State University)

Medical diagnosis app (YourMd)

Agenda

Section 0 – Introduction

Section 1 Fundamentals of Scripting

- **Overview of Meaning / ChatScript**
- Patterns in depth
- Topics, Output, Data, Functions

Section 2 How does the engine work

Fundmentals of Meaning

No intrinsic meaning. Language is agreement.

Consists of:

- Words, grammar, punctuation
- Context
  - topic of conversation – the bat flew out of his hands
  - prior sentence in volley: I like apples. Do you?
  - Prior sentence in prior volley – Do you like apples? Yes.
  - Words in nearby sentences – pronoun resolution
- Idioms/Phrases - :) what do you do?

Fundamentals of ChatScript Script

- Rules
- Topics
- Keywords
- Concepts
- Comments
  - # for user comments
  - #! for ChatScript directive comments
  - ##<< and ##>> for block comments

Rules

The basis for all meaning detection and action.

Fundamentally an IF/THEN statement.

4 parts:

kind – what types of sentences does it reply to

label – way for other rules to refer to it

pattern – IF test for allowing output to execute

output – THEN part for taking action

u: MYRULE (I love meat) So do I.

Kinds of rules

Responders – react to user input

?: - responds to questions from user

s: - responds to statements from user

u: - responds to union of questions and statements

Gambits – program says when it has sente

t: Have you ever had a pet?

Rejoinders – based on what we just said to user

a: (yes) What kind of pet?

b: (dog) I like dogs.

b: (cat) I prefer dogs.

a: (no) Pets are fun. You should try having one.

Questions about sentence kinds

What if user omits punctuation?

System infers it from sentence structure, words.

What is a command?

By default a statement. But script can remap them to be questions: *Describe an elephant* is like *What is an elephant?*

What about exclamations?

They are statements. You can match ! in patterns.

Patterns

Patterns detect meaning.

Constrains when the output of a rule can fire.

Patterns are like regular expressions, but for meaning instead of characters.

Patterns consist of tokens separated by spaces rather than letters all jammed together.

Output

Executed if a rule's pattern matches

Can be simple text

?: (<< you ~like pizza >>) I love pizza.

Can be random text

?: (<< you ~like pizza >>) [Of course][Absolutely] I [love it.][adore it.]

Text autoformats

Can be code and text intermix or code only

?: (<< what you age >>) I am ^compute(%year – 1989) years old.

Rule output is discarded if code fails in a rule

Agenda

Section 0 – Introduction

Section 1 Fundamentals of Scripting

- Overview of Meaning / ChatScript
- **Patterns in depth**
- Topics, Output, Data, Functions

Section 2 How does the engine work

Patterns in Depth

Generally done on the current sentence

Tokens are instructions to process, left to right.

```
#! what is my name
?: ( what is my name)  - simple pattern

# grab indicated letter if we have a phone number
u: LETTERS($$phone @_10+ _*1 )  - esoteric pattern
```

Simple Pattern Match

Match words in sentence

u: (I love pizza)

?? *I love pizza*

?? I will love pizza

Pattern words cased as they should be

u: (I love pizza from Pizza Hut) matches these:``

"i love pizza from pizza hut"

"I LOVE PIZZA FROM PIZZA HUT"

"i LOVE pizza From pizza Hut"

Pattern Problem

Linear sequences not scalable.

AIML improved this by:

- eliminate case – all upper case (losing useful data)
- eliminate punctuation (losing useful data)
- adding wildcard * to match 1 or more words

I * LOVE * YOU

AIML Problem

Requires match of entire input sentence.

Needs 4 patterns to match.

WORD . WORD

* WORD … WORD

WORD … WORD *

* WORD … WORD *

2015 AIML's 1st revision in 20 years. Provides wildcard matching 0 or more words.

* WORD … WORD *

Patterns- CS canonical forms

Input creates 2 sentences- original + canonical

Matching looks at both

nouns: bicycles -> bicycle

verbs: were -> be

numbers: two -> 2 2.0 -> 2 two thousand and fifty -> 2050

determiners: the -> a

personal pronouns: my -> I him -> he whom -> who

u: (I love pizza) matches

I loved pizza

me loves pizzas

Patterns- Canonical forms

Can require original form only using '

u: ('I 'love 'pizza)

Can require original form if not canonical

u: (I loved pizzas)

?? me love pizza

?? me loved pizzas

Patterns – bags of words

[ ] - find one

u: (I love [pizza bacon sausage hamburger])

{ } - maybe find one

u: (I love {the my} moon)

<< >> - find all in any order anywhere

u: (<< pizza I love >>)

Particularly important. Breaks the straight-jacket of sequence.

Patterns – composed bags

u: (<< I [like love adore] [pizza bacon] >>) So do I.

“I love bacon”

“Pizza is what I adore”

“I hate pizza but love rocks”

u: (I love [meat << flesh animal >>]) So do I.

“I love meat”

“I love animal flesh”

“I love the flesh of an animal”

Pattern - wildcards

Keeping sequence but not requiring contiguity

***** - any number of words (including 0)

u: (I love * pizza) - matches "I love spicy pepperoni pizza"

but matches “I love you and hate pizza” – still a flaw with AIML

***1** - 1 word exactly *2, *3 ..

u: (I love *1 pizza) - matches "I love pepperoni pizza"

***~1** - 0 or 1 word *~2, *~3 ..

u: (I *~2 love *~2 pizza) ?? examples?

But “I do not love pizza” is a problem.

Patterns - exclusions

! - make sure this not seen later in sentence

u: (!not I *~2 love *~2 pizza)

u: (!not !pepperoni I *~2 love *~2 pizza

u: (![not pepperoni] I *~2 love *~2 pizza

Patterns- Phrases

Phrase = sequence of 1-5 words in “ ”

u: (I * work * “International Business Machines”)

How is this different from

u: (I * work * International Business Machines)

The phrase is a single idea, an entity. Your pattern should reflect that. Humans will understand your pattern better. And CS will handle it correctly against a range of tokenization options. Particularly if it involves periods and commas

u: (I * live * “Raleigh, North Carolina”)

Patterns - Concepts

A concept set is a bag of words.

concept: ~like (like adore love lust “be big on”)

Can be used in patterns in place of words.

Same canonical rules as patterns, ' or not.

u: (!~negativeWords << I ~like ~nutrient >>)

This rule captures a probable meaning

I like some kind of food or drink

Can still go wrong. Rules are not perfect.

Patterns- memorization

_ - memorize word of input

Auto assigned match variable numbering.

u: (I * love * _~nutrient) I love _0 too.

Stores original, canonical, and position.

u: (I * _~like * _~nutrient) I love '_1 too.

u: (I like _{~pizzaToppings} pizza)

"I like pizza" - _0 is null

"I like pepperoni pizza" - _0 is pepperoni

Patterns - comparisons

Can test relationships > < == !=

u: (_~number _0==3) -- relatively pointless since u: (3) is equivalent

u: (_~number _0>0 _0<10)

Can test concept membership ? !?

u: (_~animals _0?~pets)

Can test bit patterns & ^

u: (_~number _0&1)

Can test existence (not null)

u: (I love _{pepperoni} pizza _0) – but a pointless pattern here

Patterns- tag-along patterns

Output requests more matching via ^refine().

Acts like a switch statement, only 1 can match.

u: (I ~like _~nutrient) ^refine()

a: (_0?~dessert) I love brownies.

a: (_0?~meat) I like steak.

a: (_0?~beverage) I drink tea.

a: () All food is good.

Patterns - esoteric

Be aware:

- You can match forwards and/or backwards
- You can jump around in the sentence.
- You can match partial spellings of words.
  - u: (my sign is Sagitt*)
- You can hide or add concept meanings
  - u: (_coffee ~nutrient) ^unmark(~color _0)
    - " I like coffee ice cream" – coffee is not a color
- Force alignment to start or end of sentence
  - u: (< ice is nice >) ice is 1st word of sentence, nice is last word

Agenda

Section 0 - Introduction

Section 1 Fundamentals of Scripting

- Overview of Meaning / ChatScript
- Patterns in depth
- **Topics, Output, Data, Functions**

Section 2 How does the engine work

Topics

A named collection of rules tried in order

Can have keywords to find or invoked by name

Has properties like don't erase, allow repeat

```
topic: ~funerals repeat (die death burial funeral)
?: ( will you ever die ) Not soon I hope
t: have you ever been to a funeral?
    a: (yes) Did you enjoy it?
t: I have. It was boring.
```

- Use relevant keyword (! prepositions/pronouns)

Data - variables

1. Match variables from memorization like _0

2. User variables like $var

   $var = _0 - single $ are permanent

   $$tmp = 20 - double $$ last for the volley only

3. Read-only system variables like %time

Data

All data is text.

- Digit text autoconverts to numbers as needed.
- Active Strings ^”The answer is $answer.”
  - Convert on use into simple text
  - Is not autoformatted

Data - facts

A triple of relationship information

(you age 23)

(“The Wind and the Willows” author “Kenneth Grahame”)

Represents internals using MEMBER and IS

(black member ~colors) - CS representation of a concept

(Laborador~1 is dog~2) - Wordnet relationship of word meanings

Can query for facts given one or more fields

Facts- Tables

Can define tables of data to process into fact.

Processing happens at build time

Efficient scripting- all elaborate patterns in common topic

- table: ^favetable(~foods)
- chocolate candy "I love Cadbury's Flake chocolate bar."
- Easter candy "I love marshmallow bunnies."
- [salad _] dressing "I prefer sweet dressings to sour ones."
- pizza topping "My favorite pizza topping is pepperoni because I like meat."
- comfort food "My favorite comfort food is is a Gruyere cheese toastie."
- _ pizza "My favorite pizza is pepperoni because I like pizza with meat toppings."

Data - JSON

Classic JSON data used in web calls

Represented internally as facts

Functions to create, query, etc.

Functions

Classic functions with arguments start with ^

```
$$tmp = ^pick(~animals)
```

Call by reference

Variety of predefined functions

Can define user functions for patterns or output

Most useful are:

^gambit(~topic), ^respond(~topic)

^refine(), ^reuse(label)

Functions vs Rules vs Topics

Executes script — Matches patterns

Functions Rules — Rules Script (via If)

Can be called

Functions (with args) Rules (0 args)

So how are they different? They barely are.

Rules are separately labelled

Topics can be invoked by association (keyword)

Topics stop when output is generated

Functions & Rules execute to completion

Std Engine Control Flow

A control topic manages engine execution.

Below default engine behavior can be altered.

Input volley is processed sentence by sentence

Rule execution is top to bottom in a topic

Rules can invoke topics or other rules

New rules tried until user output is generated.

Rules erase after generating user output

Rules decline to repeat output close together

Agenda

Section 0 – Introduction

Section 1 Fundamentals of Scripting

Section 2 How does the engine work

- **What is the NL pipeline**
- How does the engine work

NL Pipeline

1. Process characters into words/sentences
2. Apply standard transformations

adjust case (i->I)

expand contractions

spell correction

remove end punctuation

British → American

texting → American

dialog act conversion

interjection splitting

merge dates, numbers, proper nouns

substitutions (FB-> Facebook, 1am → 1 am)

separation (5'11” → 5 foot 11 inch, Bob's → Bob 's)

remove noise (anonymous stranger -> stranger)

NL Pipeline

3. Part-of-Speech tagging and parsing
4. Dual stream of original and canonical words
5. Mark words with what concepts and where
   All stored in dictionary for hash lookup.

Can use :prepare to see results or :trace

Now ready to execute script.

Agenda

Section 0 – Introduction

Section 1 Fundamentals of Scripting

Section 2 How does the engine work

- What is the NL pipeline
- **How does the engine work**

CS Design Goals

Low memory – runnable locally in cellphone

High speed – runnable on weak processors

Concise – rules take few characters

Precise – can represent specific meanings

General – beyond chat, general NL tool

Commercial grade

- scaleable
- script testing and analytic support
- Well documented

Engine sequencing

Startup load dictionary, layer 0, layer 1

Read user input (username + botname + input)

Read in user topic file

Run NL pipeline

Run Script

Write changed user topic file back

Send output to user

Dictionary

- Every word, phrase, fact value, concept name, function name, etc stored in hashtable
- Lower & uppercase words have same hash
  - Uppercase words use next bucket up
  - Enables case-based lookups of same word
- New words from user go into reserved bucket
  - Are ripped back at end of volley
- Every word has lists of facts using it as S V O
  - Enables query by field value
  - Facts also have lists of facts using it as S V O

Marking

- Dictionary entry for every word, phrase, concept
- Words have timestamp + loc. in sentence.
- Words have lists of facts they participate in.
- Marking takes original/canonical word
  - Fills in location in sentence
  - Propagates from facts in which word is subject
    - using MEMBER & IS to object of fact and marks that
  - Propagate restricted by POS or Wordnet meaning
- Pattern test can then look up word for status

Memory Management

- No dynamic malloc/free
- Mark/release per volley, no interim release
- Memory pre-allocated to pools of:
  - Dictionary entries
  - Facts
  - String space
  - Buffers (buffer pool array, not mark-release)
- Facts & Words inter-locked, unpeel to release

Script Control

Bot definition assigns topics to variables.

Only $cs_controlmain is required.

$cs_controlpre – once per volley at start

$cs_prepass - once per sentence

$cs_controlmain – once per sentence

$cs_controlpost – once per volley at end

Simple Control Topic

The Harry control flow processes sentences until a single output is generated. Does this:

- See if rejoinder matches
- Respond in current topic
- Respond in other topics based on keyword.
- Respond from keywordless topic.
- Quibble sometimes.
- Gambit from current topic.
- Gambit from other topics based on keyword.
- Gambit from any topic.

Error Handling

All functions return 2 things:

- Output - is either assigned or dumped to user
  - ^myfunction(_0) - result seen in user output
  - $var = ^myfunction(_0) - result stored on variable
- Error code - tells how to proceed
  - NoProblem
  - FailRule FailTopic FailInput FailSentence
  - EndRule EndTopic EndSentence EndInput
  - RetryRule RetryTopic RetrySentence RetryInput

Out-of-band data

ChatScript manages 2 channels of data

- Input and output with the user
- Input and output with the application

Application data (OOB) in [ ] first.

- [category: vet specialty: cat] Hi, I'm Rosa. →
- → [callback: 600] My name is Pearl.

Application responsible for handling [ ] in output

Script responsible for handling [ ] in input

Engine aware of [ ], does not apply NL pipeline

^Function Support

- API routines for:
  - Manipulating topics
  - Marking and parsing
  - Input control
  - Manipulating numbers
  - Output generation and access
  - Word manipulation
  - Manipulating facts
  - Manipulating JSON
  - Debugging
  - Postgres DB use
- Controlling flow

ChatScript Server

One thread devoted to CS main engine

Other threads accept and read input

Then queue for engine service

Then handle output and file writing

Can run server on each core, tied to same port.

Server is CPU bound.

:Command Support

- Testing and display- :trace, :testpattern
- Proving your code works- :verify, :regress
- Analytics on logs- :trim :abstract
- Document reading- :document
- Word information- :up, :down, :word, :prepare
- Control- :build, :restart, :reset
- Topic data- :topicstats, :topicinfo

  and lots more

Extensive Documentation

- Manuals: Guide to Documentation
  - Basic + advanced User manuals
  - System functions (API guide)
  - JSON, facts
  - System variables and Engine-defined concepts
  - Finalizing a bot
  - Debugging
  - …..
  - Papers on history and theory of ChatScript/chat