



Dialog Requirements for Voice Markup Languages

W3C Working Draft 23 December 1999

This version:

<http://www.w3.org/TR/1999/WD-voice-dialog-reqs-19991223>

Latest version:

<http://www.w3.org/TR/voice-dialog-reqs>

Previous version:

<http://www.w3.org/Voice/Group/1999/dialog-reqs-19991130.html>

Editor:

Scott McGlashan

Copyright © 1999 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C [liability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply.

Abstract

The W3C Voice Browser working group aims to develop specifications to enable access to the Web using spoken interaction. This document is part of a set of requirements studies for voice browsers, and provides details of the requirements for marking up spoken dialogs.

Status of this document

This document describes the requirements for marking up dialogs for spoken interaction, as a precursor to starting work on specifications. Related requirement drafts are linked from the [introduction](#). The requirements are being released as working drafts but are not intended to become proposed recommendations.

This specification is a Working Draft of the Voice Browser working group for review by W3C members and other interested parties. This is the first public version of this document. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress".

Publication as a Working Draft does not imply endorsement by the W3C membership, nor of members of the Voice Browser working groups. This is still a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite W3C Working Drafts as other than "work in progress."

This document has been produced as part of the [W3C Voice Browser Activity](#), following the procedures set out for the [W3C Process](#). The authors of this document are members of the [Voice Browser Working Group](#). This document is for public review. Comments should be sent to the public mailing list <www-voice@w3.org> ([archive](#)) by 14th January 2000.

A list of current W3C Recommendations and other technical documents can be found at <http://www.w3.org/TR>.

0. Introduction

The process will consist of the following steps:

1. Collect requirements on spoken dialog.
2. Prioritize these requirements.
3. Distribute requirements to, and take feedback from, relevant groups working on specific markup languages supporting speech dialog.
4. Propose further work on how the spoken dialogs can be integrated and synchronized with other input/output media to provide co-ordinated multi-modal interaction.

0.1 Scope

The core activity focuses on defining three types of requirements on the voice markup language: modality, functional, and format. Modality requirements concern the types of modalities (media in combination with an input/output mechanism) supported by the markup language for user input and system output. Functional requirements concern the behaviour (or operational semantics) which results from interpreting a voice markup language. Format requirements constrain the format (or syntax) of the voice markup language itself.

The environment and capabilities of the voice browser interpreting the markup language will affect these requirements. There may be differences in the modality and functional requirements for desktop versus telephony-based environments (and in the latter case, between fixed, mobile and internet telephony environments). The capability of the voice browser device will also have an important impact on requirements; for example, telephones without graphical displays versus those with graphical displays. Requirements affected by the environment or capabilities of the voice browser device will be explicitly marked as such.

The Subgroup will not directly address how these requirements are implemented in specific SGML or XML languages. It is agnostic between the (X)HTML approach (where, for example, [Conversational Computing](#), [PipeBeach](#), [Productivity Works](#), and [Vocalis](#) interpret HTML as voice markup) and XML languages specifically designed for spoken dialog ([VoxML](#), [SpeechML](#), [TalkML](#), [VoiceXML](#), etc). However, for illustrative purposes, examples and explanations of requirements may be given in specific markup languages.

This Subgroup does not arbitrate on the extent to which dialogs have graphical web browsers as their reference model: i.e. the 'dialog' could be provided by any standard spoken dialog system, or a (voice) browser with spoken dialog capabilities. However, in both cases the voice markup needs to support meta-commands (see Section 2.6).

Finally, features like call processing and billing are not regarded as dialog requirements but as application design issues. While they may have an impact on dialog requirements, they are not part of the dialog markup or behaviour itself.

0.2 Interaction with Other Groups

The activities of the Dialog Requirements Subgroup will be coordinated with the activities of the Grammar Representation Subgroup, the Synthesis Markup Subgroup, the Natural Language Subgroup, the Multimodal Interaction Subgroup and the Reusable Dialog Components Subgroup.

0.3 Terminology

Although defining a dialog is highly problematic, some basic definition must be provided to establish a common basis of understanding and avoid confusion. The following terminology is based upon an event-driven model of dialog interaction.

| | |
|------------------------------|--|
| Voice Markup Language | a language in which voice dialog behaviour is specified. The language may include reference to style and scripting elements which can also determine dialog behaviour. |
| Voice Browser | a software device which interprets a voice markup language and generates a dialog with voice output and/or input, and possibly other modalities. |
| Dialog | a model of interactive behaviour underlying the interpretation of the markup language. The model consists of states, variables, events, event handlers, inputs and outputs. |
| State | the basic interactional unit defined in the markup language; for example, an < input > element in HTML. A state can specify variables, event handlers, outputs and inputs. A state may describe output content to be presented to the user, input which the user can enter, event handlers describing, for example, which variables to bind and which state to transition to when an event occur. |
| Events | generated when a state is executed by the voice browser; for example, when outputs or inputs in a state are rendered or interpreted. Events are typed and may include information; for example, an input event generated when an utterance is recognized may include the string recognized, an interpretation, confidence score, and so on. |
| Event Handlers | are specified in the voice markup language and describe how events generated by the voice browser are to be handled. Interpretation of events may bind variables, or map the current state into another state (possibly itself). |
| Output | content specified in an element of the markup language for presentation to the user. The content is rendered by the voice browser; for example, audio files or text rendered by a TTS. Output can also contain parameters for the output device; for example, volume of audio file playback, language for TTS, etc. Events are generated when, for example, the audio file has been played. |
| Input | content (and its interpretation) specified in an element of the markup language which can be given as input by a user; for example, a grammar for DTMF and speech input. Events are generated by the voice browser when, for example, the user has spoken an utterance and variables may be bound to information contained in the event. Input can also specify parameters for the input device; for example, timeout parameters, etc. |

The dialog requirements for the voice markup language are annotated with the following priorities. If a feature is deferred from the initial specification to a future release, consideration may be given to leaving open a path for future incorporation of the feature.

| | |
|------------------------|---|
| must have | The first official specification must define the feature. |
| should have | The first official specification should define the feature if feasible but may defer it until a future release. |
| nice to have | The first official specification may define the feature if time permits, however, its priority is low. |
| future revision | It is not intended that the first official specification include the feature. |

1. Modality Requirements

These requirements will be co-ordinated with the Multimodal Interaction Subgroup.

1.1 Audio Modality Input and Output (must have)

The markup language can specify which spoken user input is interpreted by the voice browser, as well as the content rendered as spoken output by the voice browser.

1.2 Sequential multi-modal Input (must have)

The markup language specifies that user input from multiple modalities is to be interpreted by the voice browser. There is no requirement that the input modalities are simultaneously active. For example, a voice browser interpreting the markup language in a telephony environment could accept DTMF input in one dialog state, and spoken input in another.

1.3 Unco-ordinated, Simultaneous, Multi-modal Input (should have)

The markup language specifies that user input from different modalities is to be interpreted at the same time. There is no requirement that interpretation of the input modalities are co-ordinated. For example, a voice browser in a desktop environment could accept keyboard input or spoken input in same dialog state.

1.4 Co-ordinated, Simultaneous Multi-modal Input (nice to have)

The markup language specifies that user input from multiple modalities is interpreted at the same time and that interpretation of the inputs are co-ordinated by the voice browser. For example, in a telephony environment, the user can type *200* on the keypad and say *transfer to checking account* and the interpretations are co-ordinated so that they are understood as *transfer 200 to checking account*.

1.5 Sequential multi-modal Output (must have)

The markup language specifies that content is rendered in multiple modalities by the voice browser. There is no requirement the output modalities are rendered simultaneously. For example, a voice browser could output speech in one dialog state, and graphics in another.

1.6 Unco-ordinated, Simultaneous, Multi-modal Output (nice to have)

The markup language specifies that content is rendered in multiple modalities at the same time. There is no requirement the rendering of output modalities are co-ordinated. For example, a voice browser in a desktop environment could display graphics and provide audio output at the same time.

1.7 Co-ordinated, Simultaneous Multi-modal Output (nice to have)

The markup language specifies that content is to be simultaneously rendered in multiple modalities and

2. Functional Requirements

These requirements are intended to ensure that the markup language is capable of specifying co-operative dialog behaviour characteristic of state-of-the-art spoken dialog systems. In general, the voice browser should compensate for its own limitations in knowledge and performance compared with equivalent human agents; for example, compensate for limitations in speech recognition capability by confirming spoken user input when necessary.

2.1 Mixed Initiative: Form Level (must have)

Mixed initiative refers to dialog where one participant take the initiative by, for example, asking a question and expects the other participant to respond to this initiative by, for example, answering the question. The other participant, however, responds instead with an initiative by asking another question. Typically, the first participant then responds to this initiative, before the second participant responds to the original initiative. This behaviour is illustrated below:

S-A1: When do you want to fly to Paris?
U-B1: What did you say?
S-B2: I said when do you want to fly to Paris?
U-A2: Tuesday.

where A1 is responded to in A2 after a nested interaction, or sub-dialog in B1 and B2. Note that the B2 response itself could have been another initiative leading to further nesting of the interaction.

The form-level mixed initiative requirement is that the markup language can specify to the voice browser that it can take the initiative when user expects a response, and also allow the user to take the initiative when it expects a response where the content of these initiatives is relevant to the task at hand, contains navigation instructions or concerns general meta-communication issues. This mixed initiative requirement is particularly important when processing form input (hence the name) and is further elaborated in requirements 2.1.1, 2.1.2, 2.1.3 and 2.1.4 below.

2.1.1 Clarification Subdialog (must have)

The markup language can specify that a clarification sub-dialog should be performed when the user provides incomplete, form-related information. For example, in a flight enquiry service, the departure city and date may be required but the user does not always provide all the information at once:

S1: How can I help you?
U1: I want to fly to Paris.
S2: When?
U1: Monday

U1 is incomplete (or 'underinformative') with respect to the service (or form) and the system then initiates a sub-dialog in S2 to collect the required information. If additional parameters are required, further sub-dialogs may be initiated.

2.1.2 Confirmation Subdialog (must have)

The markup language can specify that a confirmation sub-dialog is to be performed when the confidence associated with the interpretation of the user input is too low.

U1: I want to fly to Paris.
S1: Did you say 'I want a fly to Paris'?
U2: Yes.
S2: When?

Note confirmation sub-dialogs take precedence over clarification sub-dialogs.

2.1.3 Over-informative Input: corrective (must have)

The markup language can specify that unsolicited user input in a sub-dialog which corrects earlier input is to be interpreted appropriately. For example, in a confirmation sub-dialog users may provide corrective information relevant to the form:

S1: Did you say you wanted to travel from Paris?

U1: No, from Perros. (modification)

U1': Yes, from Paris (repetition)

2.1.4 Over-informative Input: additional (nice to have)

The markup language can specify that unsolicited user input in a sub-dialog which is not corrective but additional, relevant information for the current form is to be interpreted appropriately. For example, in a confirmation sub-dialog users may provide additional information relevant to the form:

S1: Did you say you wanted to travel from Paris?

U1: Yes, I want to fly to Paris on Monday around 11.30

2.2 Mixed Initiative: Task Level (must have)

The markup language needs to address mixed initiative in dialogs which involve more than one task (or topic). For example, a portal service may allow the user to interact with a number of specific services such as car hire, hotel reservation, flight enquiries, etc, which may be located on the different web sites or servers. This requirement is further elaborated in requirements 2.2.1, 2.2.2, 2.2.3, 2.2.4 and 2.2.5 below.

2.2.1 Explicit Task Switching (must have)

The markup language can specify how users can explicitly switch from one task to another. For example, by means of a set of global commands which are active in all tasks and which take the user to a specific task; e.g. *Take me to car hire, Go to hotel reservations.*

2.2.2 Implicit Task Switching (should have)

The markup language can specify how users can implicitly switch from one task to another. For example, by means of simply uttering a phrases relevant to another task; *I want to reserve a McLaren F1 in Monaco next wednesday.*

2.2.3 Manual Return from Task Switch (must have)

The markup language can specify how users can explicitly return to a previous task at any time. For example, by means of global task navigation commands such as *previous task.*

2.2.4 Automatic Return from Task Switch (should have)

The markup language can specify that users can automatically return to the previous task upon completion or explicit cancellation of the current task.

2.2.5 Suspended Tasks (should have)

The markup language can specify that when task switching occurs the previous task is suspended rather than cancelled. Thus when the user returns to the previous task, the interaction is resumed at the point it was suspended.

2.3 Help Behaviour (should have)

The markup language can specify help information when requested by the user. Help information should be available in all dialog states.

S1: Howcan I help you?

U1: What can you do?

S2: I can give you flight information about flights between major cities world-wide just like a travel agent. Howcan I help you?

U1: I want a flight to Paris ...

Help information can be tapered so that it can be elaborated upon on subsequent user requests.

2.4 Error Correction Behaviour (must have)

The markup language can specify how error events generated by the voice browser are to be handled. For example, by initiating a sub-dialog to describe and correct the error:

S1: Howcan I help you?

U1: <audio but no interpretation>

S2: Sorry, I didn't understand that. Where do you want to travel to?

U2: Paris

The markup language can specify how specific types of errors encountered in spoken dialog, e.g. no audio, too loud/soft, no interpretation, no audio, internal error, etc, are to be handled as well as providing a general 'catch all' method.

2.5 Timeout Behaviour (must have)

The markup language can specify what to do when the voice browser times out waiting for input; for example, a timeout event can be handled by repeating the current dialog state:

S1: Did you say monday?

U1: <timeout>

S2: Did you say Monday?

Note that the strategy may be dependent upon the environment; in a desktop environment, repetition for example may be irritating.

2.6 Meta-Commands (should have)

The markup language specifies a set of meta-command functions which are available in all dialog states; for example, repeat, cancel, quit, operator, etc.

The precise set of meta-commands will be co-ordinated with the Telephony Speech Standards Committee.

The markup language should specify how the scope of meta-commands like 'cancel' is resolved.

2.7 Barge-in Behaviour (should have)

The markup language specifies when the user is able to bargein on the system output, and when it is not allowed.

Note: The output device may generate timestamped events when barge-in occurs (see 3.9).

2.8 Call Transfer (should have)

The markup language specifies a mechanism to allow transfer of the caller to another line in a telephony environment. For example, in cases of dialog breakdown, the user can be transferred to an operator (cf. 'callto' in HTML). The markup language also provides a mechanism to deal with transfer failures such as when the called line is busy or engaged.

2.9 Quit Behaviour (must have)

The markup language provides a mechanism to terminate the session (cf. user-terminated sessions via a 'quit' meta-command in 2.6).

2.10 Interaction with External Components (must have)

The markup language must support a generic component interface to allow for the use of external components on the client and/or server side. The interface provides a mechanism for transferring data between the markup language's variables and the component. Examples of such data are: configuration parameters (such as timeouts), and events for data input and error codes. Except for event handling, a call to an external component does not directly change the dialog state, i.e. the dialog continues in the state from which the external component was called.

Examples of external components are pre-built dialog components and server scripts. Pre-built dialogs are further described in Section 3.3. Server scripts can be used to interact with remote services, devices or databases.

3. Format Requirements

3.1 Ease of Use (must have)

The markup language should be easy for designers to understand and author without special tools or knowledge of vendor technology or protocols (dialog design knowledge is still essential).

3.2 Simplicity and Power (must have)

The markup language allows designers to rapidly develop simple dialogs without the need to worry about interactional details but also allow designers to take more control over interaction to develop complex dialogs.

3.3 Support for Modularity and Re-use (should have)

The markup language complies with the requirements of the Reusable Dialog Components Subgroup.

The markup language can specify a number of pre-built dialog components. This enables one to build a library of reusable 'dialogs'. This is useful for handling both application specific input types, such as telephone numbers, credit card number, etc as well as those that are more generic, such as times, dates, numbers, etc.

3.4 Naming (must have)

Dialogs, states, inputs and outputs can be referenced by a URI in the markup language.

3.5 Variables (must have)

Variables can be defined and assigned values.

Variables can be scoped within namespaces: for example, state-level, dialog-level, document-level, application-level or session-level. The markup language defines the precise scope of all variables.

The markup language must specify if variables are atomic or structured.

Variables can be assigned default values. Assignment may be optional; for example, in a flight reservation form, a 'special meal' variable need not be assigned a value by the user.

Variables may be referred to in the output content of the markup language.

The precise requirements on variables may be affected by W3C work on modularity and XML schema

3.6 Variable Binding (must have)

User input can bind one or more state variables. A single input may bind a single variable or it may bind multiple variables in any order; for example, the following utterances result in the same variable bindings

- Transfer \$200 from savings to checking
- Transfer \$200 to checking from savings
- Transfer from savings \$200 to checking

3.7 Event Handler (must have)

The markup language provides an explicit event handling mechanism for specifying actions to be carried out when events are generated in a dialog state.

Event handlers can be ordered so that if multiple event handlers match the current event, only the handler with the highest ranking is executed. By default, event handler ranking is based on proximity and specificity: i.e. the handler closest in the event hierarchy with the most specific matching conditions.

Actions can be conditional upon variable assignments, as well as the type and content of events (e.g. input events specifying media, content, confidence, and so on).

Actions include: the binding of variables with information, for example, information contained in events; transition to another dialog state (including the current state).

3.8 Builtin Event Handlers (should have)

The markup language can provide implicit event handlers which provide default handling of, for example, timeout and error events as well as handlers for situations, such as confirmation and clarification, where there is a transition to a implicit dialog state. For example, there can be a default handler for user input events such that if the recognition confidence score is below a given threshold, then the input is confirmed in a sub-dialog.

Properties of implicit event handlers (thresholds, counters, locale, etc) can be explicitly customized in the markup language.

Implicit event handlers are always overridden by explicit handlers.

3.9 Output Content and Events (must have)

The markup language complies with the requirements developed by the Speech Synthesis Markup Subgroup for output text content and parameter settings for the output device. Requirements on multimodal output will be co-ordinated by the Multimodal Interaction Subgroup (cf. Section 1).

In addition, the markup supports the following output features (if not already defined in the Synthesis Markup):

1. Pre-recorded audio file output
2. Streamed audio
3. Playing/synthesizing sounds such as tones and beeps
4. variable level of detail control over structured text

The output device generates timestamped events including error events and progress events (output started/stopped, current position).

3.10 Richer Output (nice to have)

The markup language allows for richer output than variable substitution in the output content. For

3.11 Input Content and Events (must have)

The markup language complies with the requirements developed by the Grammar Representation Subgroup for the representation of speech grammar content. Requirements on multimodal input will be co-ordinated by the Multimodal Interaction Subgroup (cf. Section 1).

The markup language can specify the activation and deactivation of multiple speech grammars. These can be user-defined, or builtin grammars (digits, date, time, money, etc).

The markup language can specify parameters for speech grammar content including timeout parameters --- maximum initial silence, maximum utterance duration, maximum within-utterance pause --- energy thresholds necessary for bargein, etc.

The input device generates timestamped events including input timeout and error events, progress events (utterance started, interference, etc), and recognition result events (including content, interpretation/variable bindings, confidence).

In addition to speech grammars, the markup language allows input content and events to be specified for DTMF and keyboard devices.

4. Other Requirements

4.1 Event Handling (must have)

One key difference between contemporary event models (e.g. DOM Level 2, 'try-catch' in object-oriented programming) is whether the same event can be handled by more than one event handler within the hierarchy. The markup language must motivate whether it supports this feature or not.

4.2 Logging (nice to have)

For development and testing it is important that data and events are to be logged by the voice browser. At the most detailed level, this will include logging of input and output audio data. A mechanism which allows logged data to be retrieved from a voice browser, preferably via standard internet protocol (http, ftp, etc), is also required.

One approach is to require that the markup language can control logging via, for example, an optional meta tag. Another approach is for logging to be controlled by means other than the markup language, such as via proprietary meta tags.

4.3 Speaker Verification (should have)

The markup language could provide the ability to verify a speaker's identity through a dialog containing both acoustic verification and knowledge verification. The acoustic verification may compare speech samples to an existing model (kept in some, possibly external, repository) of that speaker's voice. A verification result returns a value indicating whether the acoustic and knowledge tests were accepted or rejected. Results for verification and results for recognition may be returned simultaneously.

5. Acknowledgments

Subgroup Members

Laurence Ferrieux (France Telecom)

Linda Dorrian (Productivity Works)

Andreas Kellner (Philips)

Kenneth Rehor (Bell Labs)

David Attwater (BT)

Danniel Burnett (Nuance)

- Andrew Hunt (Sun Labs)
- Robert Keiller (Canon)
- James Larson (Intel)
- William Ledingham (SpeechWorks)
- Bruce Lucas (IBM)
- Jen Marschner (Philips)
- Scott McGlashan (PipeBeach)
- Michael Phillips (SpeechWorks)
- Stephen Potter (Entropic)
- David Raggett (W3C/HP)
- Volker Steinbiss (Philips)
- Ramesh Sarukkai (L & H)
- Dwight Smith (Motorola)
- Michael Brown (Bell Labs)
- Marianne Hickey (HP)
- George White (General Magic)