

Informationstechnik
**Programmierschnittstelle für Sprachsteuerung
 (VOCAPI)**

DIN
33880

ICS 35.080; 35.240.60

Information technology – Voice Control Application
 Programming Interface (VOCAPI)

API de commande vocale

Inhalt

	Seite		Seite
Vorwort	2	6 Die API-Funktionen	11
Einleitung	2	6.1 Allgemeines	11
1 Anwendungsbereich	2	6.2 Überblick	12
2 Allgemeines	2	6.3 Allgemeine Funktionen	13
3 Das Zustandsmodell der Schnittstelle	4	6.4 Vokabularfunktionen	17
3.1 Allgemeines	4	6.5 Erkennungsfunktionen	21
3.2 Zustandsmodell bei synchroner Wiedergabe	4	6.6 Verarbeitungsfunktionen	23
3.3 Zustandsmodell bei asynchroner Wiedergabe	6	6.7 Sprachmodellfunktionen	24
4 Begrenzungen, Konstanten und Datentypen	6	6.8 Funktionen für benutzerdefinierte Wörter	27
4.1 Begrenzungen	6	6.9 Ansagefunktionen	30
4.2 Konstanten	7	Anhang A (informativ) Implementierungs- vorschlag für die Programmiersprache C	33
4.3 Datentypen	7	A.1 Allgemeines	33
4.4 Konventionen	9	A.2 Abbildung auf die C-Programmiersprache	33
5 Parameter	9	A.2.1 Präfix als Namenszusatz	33
5.1 Wortabhängige Parameter	9	A.2.2 Konstantendefinitionen	33
5.2 Allgemeine Parameter	10	A.2.3 Typendefinitionen	33
5.3 Spracherkennerspezifische Parameter	11	A.2.4 VOCAPI-C-Funktionen	34
		A.2.5 Optionale Pakete	34
		A.2.6 Regeln für die Speicherverwaltung	34
		A.3 VOCAPI-C-Header	35

Fortsetzung Seite 2 bis 42

Normenausschuss Informationstechnik (NI) im DIN Deutsches Institut für Normung e.V.

Vorwort

Diese Norm wurde vom Unterausschuss 6 „Sprachdialog“ des Arbeitsausschusses Ergonomie im Normenausschuss Informationstechnik im DIN (NI-Erg/UA6) erarbeitet. Sie beruht auf dem Vorschlag einer Spracherkennerschnittstelle, die von einem deutschen Industriekonsortium im Rahmen des BMBF-geförderten Projektes MOTIV erarbeitet wurde. Der Unterausschuss 6 entwickelte den ursprünglichen Entwurf auf Basis der Erfahrung der Ausschussmitglieder weiter und bereitete die Schnittstellenbeschreibung zur vorliegenden Norm auf.

Einleitung

Sprachsteuerung wird zunehmend in Fahrzeugen eingesetzt werden und dort Aufgaben übernehmen, wo der Fahrer jetzt noch Tasten drücken muss. Bei der Bedienung des Autoradios, des Telefons, des Navigationssystems oder anderer Geräte können die Hände am Lenkrad bleiben, denn der Fahrer bedient einfach mittels Sprache. Ein Spracherkenner transformiert die akustische Eingabe in digitale Signale und leitet diese an eine Steuereinheit weiter, die dann Anweisungen an die entsprechenden Bordgeräte verteilt.

In der gegebenen Anordnung bildet der Spracherkenner eine Einheit für sich. Er kommuniziert mit dem zu steuernden Gerät über eine Schnittstelle. Ziel dieser Norm ist die genaue Definition dieser Schnittstelle, die „Voice Control Application Programming Interface“ genannt wird, kurz VOCAPI.

Die Definition von VOCAPI ging von einer vorhandenen API für Einzelwortsteuerung aus. Die zusätzlichen Anforderungen der kontinuierlichen Spracherkennung waren ausschlaggebend für die VOCAPI-Spezifikation. Weitere Anforderungen, z. B. für die Verwendung von statistischen Sprachmodellen, wurden ebenfalls berücksichtigt.

Der Abschnitt 2 gibt einen Überblick über das Grundkonzept der Spracherkennungsschnittstelle. Das Zustandsmodell für die Bedienung des Spracherkenners wird im Abschnitt 3 vorgestellt. Eine Zusammenfassung verschiedener Festlegungen über Begrenzungen, Konstanten, Datentypen und die verwendeten Konventionen kann dem Abschnitt 4 entnommen werden. Im Abschnitt 5 werden die Parameter erläutert, mit denen allgemeine Einstellungen des Erkenners von einer Applikation beeinflusst werden können. Die Auflistung aller VOCAPI-Funktionen sowie die Beschreibung des extern sichtbaren Verhaltens dieser Funktionen erfolgen schließlich im Abschnitt 6.

Die VOCAPI-Definitionen wurden allgemein gültig formuliert, d. h., sie wurden nicht für eine spezielle Programmiersprache geschrieben. An den Stellen, wo eine Codebeschreibung nützlich ist, wird „Pseudo-Code“ verwendet. Dadurch kann stets auf die gleiche Schnittstellenspezifikation zurückgegriffen werden, egal welche Programmiersprache oder -umgebung gewählt wird oder ob eine Funktions- oder Nachrichtenschnittstelle zu implementieren ist.

Wie die allgemein gültigen Regeln, die in dieser Norm gegeben werden, für eine gewünschte Zielsprache zu interpretieren sind, wird in informativen Anhängen festgelegt. Der vorliegende Norm-Entwurf enthält im Anhang A die Übersetzung der VOCAPI-Spezifikation in die Programmiersprache C. Zu einem späteren Zeitpunkt können Übersetzungen in andere Programmiersprachen als weitere Anhänge hinzugefügt werden.

1 Anwendungsbereich

Diese Norm legt eine Programmierschnittstelle für Sprachsteuerungseinheiten (Spracherkenner) für Fahrzeuge und andere Einsatzbereiche fest.

2 Allgemeines

Der Spracherkenner wird als eine Einheit gesehen, die akustische Eingangssignale in Wörter umwandelt. Bild 1 zeigt den prinzipiellen Datenfluss: Audioströme gelangen in den Eingangs-Ringpuffer des Spracherkenners. Sobald die Eingabe verarbeitet ist, werden verschiedene Ereignisse (Events) generiert, z. B. „begin of speech detected“, „word was recognized“, „end of speech detected“ usw. (siehe Abschnitt 3). Für jedes Ereignis wird eine Nachricht in einen Ereignis-Ringpuffer geschrieben. Ein Anwendungsprogramm kann diesen Ringpuffer lesen, um den Ablauf der Erkennungsprozedur zu verfolgen.

Jedesmal, wenn der Erkenner ein Wort aus der Audio-Eingabe erkannt hat, wird es in einen Wort-Ringpuffer geschrieben. Falls verschiedene Alternativen der Erkennung vorhanden sind, wird nur die akustisch beste Wahl übernommen. Ein Anwendungsprogramm, das die Ergebnisse der Erkennung verarbeiten soll, kann diesen Ringpuffer Wort für Wort auslesen.

Alternativ zu dem Wort-Ringpuffer kann ein Programm die vollständige Information über ein Ergebnis der Erkennung aus einem Wortgraphen entnehmen. Der Wortgraph ist erst verfügbar, nachdem die komplette Äußerung (auch Phrase oder Satz genannt) verarbeitet wurde. Er enthält verschiedene Erkennungsmöglichkeiten wie Wortalternativen, Wortreihenfolgen oder ein ganzes Netzwerk aus Wörtern und Wortgruppen, bei denen zu jedem Wort die akustische Wahrscheinlichkeit indiziert sein kann. Ein Anwendungsprogramm kann neben diesen akustischen Wahrscheinlichkeiten weitere Wissensquellen wie Sprachmodelle und Kontextinformationen heranziehen, um den Wortgraph zu interpretieren.

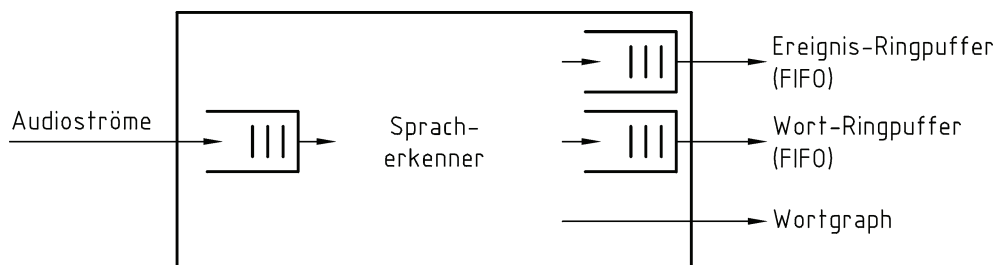


Bild 1 – Datenfluss durch den Spracherkenner

Anwendungsprogramme greifen auf den Spracherkenner über die in der vorliegenden Norm festgelegten VOCAPI-Schnittstelle zu. Wie Bild 2 entnommen werden kann, sind die VOCAPI-Funktionen in verschiedene Pakete aufgeteilt (siehe Abschnitt 6). Einige dieser Pakete sind für den Betrieb des Spracherkenners zwingend notwendig, während andere optional hinzugefügt werden können, je nach den Fähigkeiten des Erkenners und den Anforderungen an seine Implementation.

Die VOCAPI-Pakete können wiederum in zwei Kategorien unterteilt werden: in die Bedienungspakete und in ein Verarbeitungspaket. Die Bedienungspakete erlauben einer Anwendung, den Spracherkenner zu steuern und auf seine Ergebnisse über Ereignis-Ringpuffer, Wort-Ringpuffer und Wortgraphen zuzugreifen. Die Anwendung muss die Funktionen des Bedienungspaketes sequentiell aufrufen, weil nicht davon ausgegangen werden kann, dass die Funktionen dieser Pakete gleichzeitig mehrfach ausführbar (reentrant) sind.

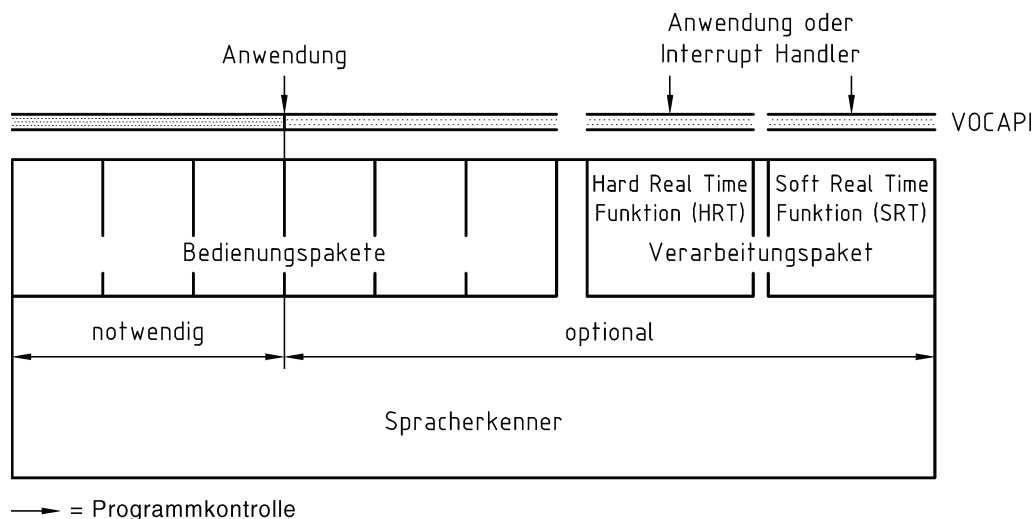


Bild 2 – Die VOCAPI-Pakete

Das Verarbeitungspaket wird nicht benötigt, wenn der implementierte Spracherkenner ein eigenständiges Bauteil mit eigenen Verarbeitungsfähigkeiten und Zugriff auf ein Audiogerät hat. Verfügt der Spracherkenner jedoch nicht über den alleinigen Zugriff auf Prozessor und/oder eine Audioeinheit, so liefert das Verarbeitungspaket wichtige Schnittstellenfunktionen, die den Erkennen mit Audiodaten versorgen und ihm Rechenzeit zuteilen. Eine solche Schnittstellenfunktion ist die Hard Real Time Funktion (HRT), die für harte Echtzeit-Bedingungen ausgelegt ist. Die HRT-Funktion nimmt Audiodaten an und weist diese den Eingangs-Ringpuffern zu. Die andere Schnittstellenfunktion ist die Soft Real Time Funktion (SRT), die weichen Echtzeit-Bedingungen genügt. Die SRT-Funktion führt der Spracherkenner-Software die Programmkontrolle und damit die Verarbeitungszeit zu. Die Verarbeitungsfunktionen können vom Anwendungsprogramm entweder synchron aufgerufen werden oder asynchron durch entsprechende Interrupts (Unterbrechungsbehandlungen) oder Event Handler (Ereignisverteiler).

3 Das Zustandsmodell der Schnittstelle

3.1 Allgemeines

Der Erkenner und damit die Schnittstelle des Erkenners können verschiedene Zustände einnehmen, die von der aktuellen Betriebsart abhängen. Für einige API-Funktionen muss der Zustand der Schnittstelle berücksichtigt werden, da manche diesen Zustand beeinflussen, andere wiederum nur in gewissen Zuständen korrekt funktionieren. Die Übergänge zwischen den API-Zuständen werden durch Ereignisse (Events) ausgelöst. Alle im Spracherkenner erzeugten Ereignisse werden in den Ereignis-Ringpuffer geschrieben, der zur Verfolgung der gesamten Ereignis-Historie von der Applikation ausgelesen werden kann.

3.2 Zustandsmodell bei synchroner Wiedergabe

3.2.1 Allgemeines

Bild 3 stellt das Zustandsdiagramm der Schnittstelle bei synchronem Wiedergabemodus dar. Der Zustand „Aufnahme“ (RECORDING) einschließlich der betreffenden Ereignisse ist irrelevant für Erkenner, die optionale Benutzerwortfunktionen (siehe 6.8) nicht unterstützen. Der Zustand „Wiedergabe“ (REPLAYING) und dessen mögliche Ereignisse sind hingegen irrelevant für Erkenner, die die Wiedergabe von Ansagen nicht unterstützen.

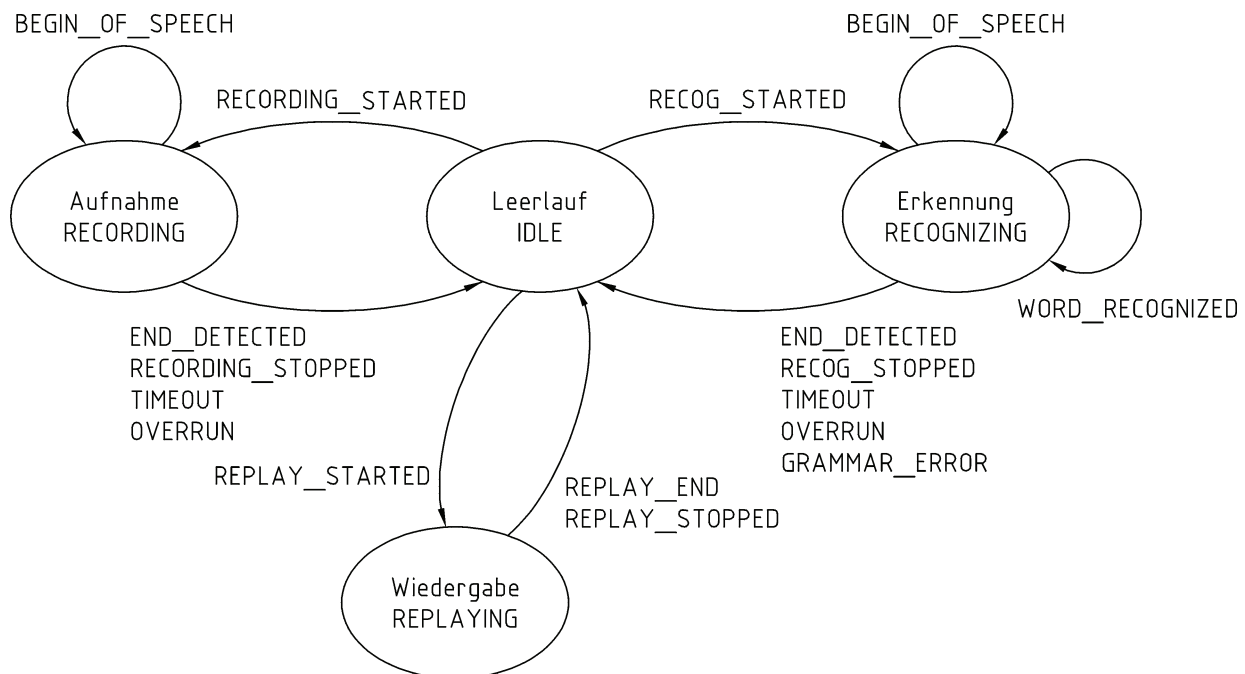


Bild 3 – Das Zustandsdiagramm der Schnittstelle bei synchronem Wiedergabemodus

In 3.2.2 bis 3.2.5 wird das Zustandsdiagramm erklärt. Die in diesem Zusammenhang erwähnten API-Funktionen werden in Abschnitt 6 beschrieben.

3.2.2 Der Leerlauf-Zustand IDLE

Wenn die Schnittstelle durch einen Aufruf der Funktion Init() initialisiert wird, so wird sie im Leerlauf (IDLE-Zustand) gestartet.

Im Leerlauf können Erkennungsparameter verändert werden; Vokabulare können ein- oder ausgelesen und bearbeitet werden. Diese Funktionen sind in anderen Zuständen nicht zulässig, da sie den Erkennungs- oder Trainingsprozess stören würden.

3.2.3 Der Erkennungs-Zustand RECOGNIZING

Ein Aufruf der Funktion RecogStart() löst ein RECOG_STARTED Ereignis aus, das wiederum die Schnittstelle in den Zustand RECOGNIZING (Erkennung) schaltet. In diesem Zustand erwartet der Spracherkenner eine Spracheingabe. Daher muss die Applikation oder ein entsprechender Interrupt Handler die HRT()-Funktion immer dann aufrufen, wenn ein Puffer mit neuen Audiodaten verfügbar ist. Die SRT()-Funktion muss ebenfalls regelmäßig aufgerufen werden, um den Erkennungsprozess voranzutreiben.

Im Zustand RECOGNIZING sucht der Spracherkennungsalgorithmus nach dem Anfang einer sprachlichen Äußerung. Sobald Sprache wahrgenommen wird, wird das Ereignis BEGIN_OF_SPEECH generiert. Die Schnittstelle bleibt im Zustand RECOGNIZING.

Der aktive Spracherkenner verarbeitet Spracheingabe im natürlichen Redefluss. Sobald der Erkenner ein Wort mit genügender Sicherheit erkannt hat, legt er es im Ringpuffer für erkannte Wörter ab und generiert das Ereignis WORD_RECOGNIZED. Der Erkenner ist weiterhin mit der Verarbeitung neuer Spracheingabe beschäftigt, demzufolge bleibt die Schnittstelle im Zustand RECOGNIZING.

Ein Endpunkt-Detektor findet das Ende einer Sprechfolge (ein Satz) und zeigt dieses durch das Ereignis END_DETECTED an. Das System stoppt den Erkennungsprozess. Die Schnittstelle wird in den Zustand IDLE (Leerlauf) zurückgesetzt.

Um den Sprachprozess zu unterbrechen, bevor er automatisch endet, kann das steuernde Programm die Funktion RecogStop() aufrufen. Diese erzeugt das RECOG_STOPPED-Ereignis, das den Erkennungsprozess sofort beendet und die Schnittstelle in den Zustand IDLE (Leerlauf) zurücksetzt. Die bis dahin verarbeiteten Ergebnisse der Erkennung sind im Wortgraphen verfügbar, können aufgrund der Zwangsunterbrechung jedoch nur eingeschränkt interpretiert werden, da über ihre Genauigkeit nichts bekannt ist.

Ein Parameter zur Zeitbegrenzung kann gesetzt werden, um den Erkennungsprozess nach Ablauf einer bestimmten Zeit zu beenden. Falls die im Timer gesetzte Zeit abläuft, bevor der Endpunkt-Detektor angesprochen wird, wird das Ereignis TIMEOUT generiert. Der Erkennungsprozess wird gestoppt und die Schnittstelle in den Zustand IDLE (Leerlauf) geschaltet. Der Wortgraph zeigt die Ergebnisse der Erkennung, die bis zum Timeout verarbeitet wurden.

Es gibt einige Überlastfälle, die zur Beendigung des Erkennungsprozesses führen. Dazu gehören insbesondere das Überlaufen interner Datenpuffer, des Ringpuffers für erkannte Wörter oder des Platzes für die Wortgraphenerzeugung. Der wahrscheinlichste Grund für ein Überlaufen ist mangelnde Verarbeitungsgeschwindigkeit oder ein zu seltenes Aufrufen der Verarbeitungsfunktion SRT(). Bei allen Überlastfällen wird das Ereignis OVERRUN erzeugt, der Erkennungsprozess gestoppt und die Schnittstelle in den Leerlauf (IDLE-Zustand) geschaltet.

Inkonsistenzen in der Beschreibung des Sprachmodells können ebenfalls zur Beendigung des Erkennungsprozesses führen. Inkonsistenzen sind z. B. nicht nachvollziehbare Übergänge zwischen Grammatikzuständen, Wörter, die im Sprachmodell erwähnt werden, aber nicht im Vokabular enthalten sind, usw. Werden solche Bedingungen während des Erkennungsprozesses entdeckt, wird das Ereignis GRAMMAR_ERROR generiert, der Erkennungsprozess gestoppt und die Schnittstelle in den Leerlauf zurückgeschaltet.

3.2.4 Der Aufnahme-Zustand RECORDING

Um ein neues Benutzerwort zu trainieren, wird die Funktion TrainRecordStart() aufgerufen, wenn die Schnittstelle im Leerlauf ist. Diese erzeugt das Ereignis RECORDING_STARTED, das System startet den Aufnahmeprozess für das Worttraining, die Schnittstelle wird in den Zustand RECORDING geschaltet. Der Erkenner erwartet nun die Eingabe von Audiodaten. Daher muss die Applikation oder der entsprechende Interrupt Handler die Funktionen HRT() und SRT() auch für den Zustand RECORDING aufrufen.

Im Zustand RECORDING wird das Ereignis BEGIN_OF_SPEECH erzeugt, sobald Sprache wahrgenommen wird. Dieses Ereignis startet die eigentliche Aufnahme. Die Schnittstelle bleibt weiterhin im Zustand RECORDING.

Die Erzeugung eines END_DETECTED-, RECORDING_STOPPED-, TIMEOUT- oder OVERRUN-Ereignisses folgt denselben Regeln wie beim Erkennungsprozess (siehe Bild 3). Der einzige Unterschied besteht im Fehlen des WORD_RECOGNIZED-Ereignisses, das spezifisch für die Erkennung ist. Nach jedem der vier Ereignisse wird der Aufnahmeprozess gestoppt und die Schnittstelle in den Leerlauf zurückgeschaltet.

3.2.5 Der Wiedergabe-Zustand REPLAYING

Der Zustand REPLAYING tritt nur ein, wenn der Erkenner das Abspielen von Ansagen (announcements) unterstützt und diese synchron abgespielt werden.

Soll eine Ansage abgespielt werden, so wird im Zustand IDLE (Leerlauf) die Funktion AudioPlay() aufgerufen. Das Ereignis REPLAY_STARTED wird erzeugt. Bei der synchronen Wiedergabe wird die Schnittstelle in den Zustand REPLAYING geschaltet.

Nachdem eine Ansage komplett von Anfang bis Ende abgespielt wurde, wird die Wiedergabe automatisch gestoppt. Das Ereignis REPLAY_END zeigt das Ende der Wiedergabe an. Die Schnittstelle wird vom Zustand REPLAYING in den IDLE-Zustand (Leerlauf) geschaltet.

Mit der Funktion AudioStop() kann das Abspielen einer Ansage unterbrochen werden. Die Funktion erzeugt das Ereignis REPLAY_STOPPED. Bei der synchronen Wiedergabe wird die Schnittstelle vom Zustand REPLAYING in den Leerlauf geschaltet.

3.3 Zustandsmodell bei asynchroner Wiedergabe

In Bild 4 wird das Zustandsdiagramm der Schnittstelle für Erkennen gezeigt, die das asynchrone Wiedergeben von Ansagen unterstützen. Solche Erkennen stellen den Parameter `ASYNC_AUDIO` bereit, der für den asynchronen Wiedergabemodus auf `TRUE` gesetzt ist (siehe 5.2 und Tabelle 6).

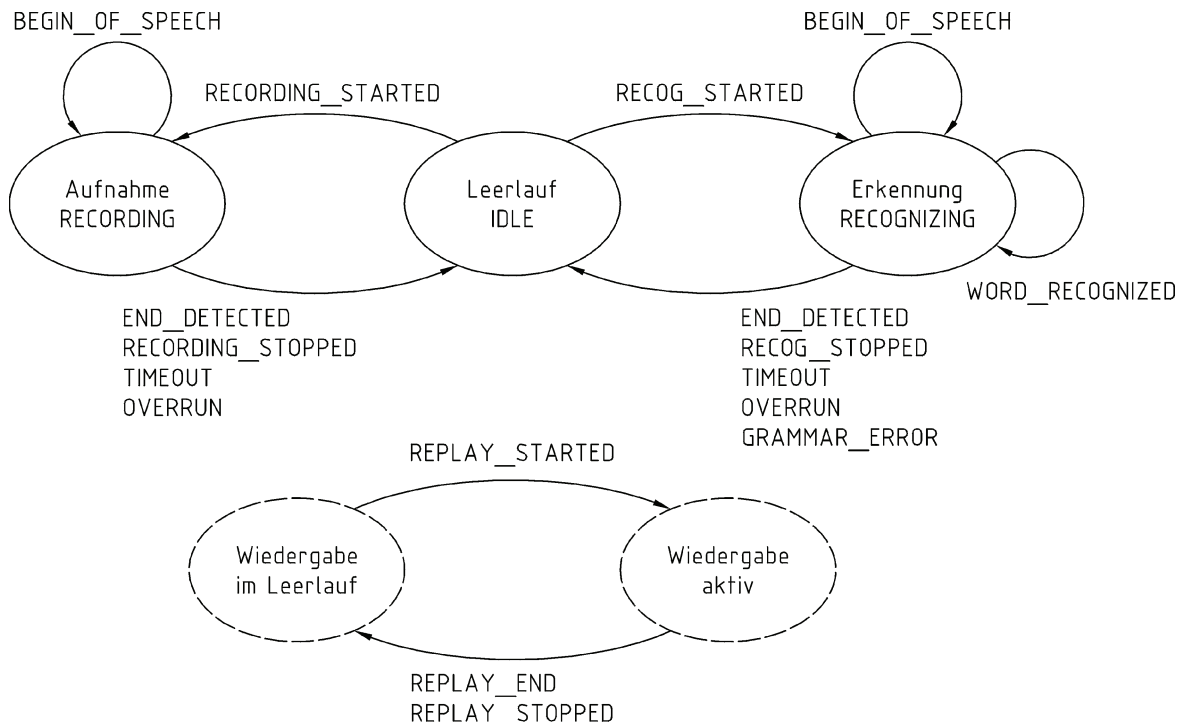


Bild 4 – Das Zustandsdiagramm der Schnittstelle bei asynchronem Wiedergabemodus

Das Zustandsmodell der Schnittstelle wird für den Betrieb mit asynchroner Wiedergabemöglichkeit durch zwei unabhängige Automaten beschrieben. Dabei bleibt der erste Automat mit den Zuständen `IDLE`, `RECOGNIZING` und `RECORDING` weiterhin über die Schnittstelle vollständig sichtbar. Es gelten für diese Zustände auch hier die Beschreibungen aus 3.2. Vom zweiten, unabhängigen Automaten sind über die Schnittstelle nur die Ereignisse `REPLAY_STARTED`, `REPLAY_END` und `REPLAY_STOP` sichtbar. Das Abspielen einer Ansage kann unabhängig vom Zustand des ersten Automaten gestartet werden. Ein `REPLAY_END`- oder `REPLAY_STOP`-Ereignis ist, wiederum unabhängig vom ersten Automaten, nur zu erwarten, wenn eine Wiedergabe zuvor gestartet wurde.

4 Begrenzungen, Konstanten und Datentypen

4.1 Begrenzungen

Wo und welche Begrenzungen zu setzen sind, hängt im Wesentlichen von der Implementation der Schnittstelle ab. Die folgende Auflistung zeigt, wo Begrenzungen erwartet werden können; sie definiert weder die exakten Grenzen, noch erhebt sie den Anspruch auf Vollständigkeit:

- Maximale Vokabularanzahl
- Maximale Anzahl von Wörtern je Vokabular
- Maximale Länge der Parameternamen
- Maximale Länge der Wortnamen
- Maximale Länge der Zeichenkette des Versionsnamens
- Maximaler Timeout-Wert
- Maximale Größe des Ereignispuffers
- Maximale Größe des Puffers für erkannte Wörter
- Maximale Anzahl von aktiven Wörtern
- Maximale Größe des Wortgraphen
- Maximale Anzahl an Grammatikzuständen
- Maximale Anzahl von Wörtern je Grammatikzustand
- usw.

4.2 Konstanten

In den meisten Fällen braucht die Applikation, die VOCAPI verwendet, keinerlei Konstanten des Spracherkenners zu kennen. Es gibt jedoch ein paar Ausnahmen (siehe Tabelle 1).

Tabelle 1 – Konstanten

Name	Beschreibung
DATASTRLEN	Je nach den Beschränkungen der Zielpattform kann es notwendig sein, die Maximalgröße von Datenfolgen zu beschränken. Wenn VOCAPI-Funktionen wie SetParamBlock(), GetParamBlock(), VocSet(), VocGet(), AudioSet() und AudioGet(), die möglicherweise mit langen Datenketten arbeiten, das Aufteilen von Datenfolgen unterstützen, so kann mit der Konstante DATASTRLEN die Größe der Datenfragmente definiert werden.
NO_OF_AUDIO_CHANNELS	Die Anzahl der Audiokanäle, die vom Spracherkennung unterstützt und von der Funktion HRT() übertragen werden.
SPELLINGLEN	Die Schreibweise eines Wortes wird als Zeichenkette gehandhabt. Ein Spracherkennung kann zwischen zwei Möglichkeiten wählen, wie diese Zeichenkette realisiert werden soll. Wenn die Konstante SPELLINGLEN nicht definiert ist, können dynamische Zeichenketten verwendet werden. Wird ein Wert von der Konstante SPELLINGLEN definiert, so gibt dieser die Länge von Datenfeldern mit fester Größe an, die für die Übertragung von Wortschreibweisen verwendet werden sollen.

4.3 Datentypen

Tabelle 2 gibt einen Überblick über alle Datentypen, die in der Schnittstellenbeschreibung verwendet werden. Aus den Tabellen 3 und 4 sind die genauen Definitionen der Aufzählungs- und Strukturtypen abzulesen.

Tabelle 2 – VOCAPI-Datentypen

Name	Beschreibung
anyType	Kann ein beliebiger der folgenden Datentypen sein
ApiStateCode	Aufzählung aller möglichen API-Zustände (siehe Tabelle 3)
BufferDescr	Eine Struktur zur Beschreibung eines Pufferinhalts (siehe Tabelle 4)
CallbackFunction	Referenz auf eine Funktion, die einem bestimmten Prototypen entspricht und vom Spracherkennung aufgerufen wird, wenn ein Ereignis aufgetreten ist
CompatType	Ein Datentyp, der eine nicht vorzeichenbehaftete Ganzzahl (unsigned integer) enthält, die den Kompatibilitätsstempel für das enthaltene Vokabulary darstellt
EventCode	Aufzählung aller möglichen Ereignisse des API-Zustandsmodells (siehe Tabelle 3)
flag	Enthält einen Booleschen Wert: TRUE oder FALSE
GraphEdge	Eine Struktur, in der die Kanten eines Wortgraphen beschrieben werden (siehe Tabelle 4)
GraphEdgeList	Eine Liste oder ein Datenfeld vom Typ „GraphEdge“
Id	Eine nicht vorzeichenbehaftete Ganzzahl (unsigned integer) zur Darstellung einer Vokabular-ID, einer Grammatikzustands-ID oder einer Puffer-ID. Es muss eine Definition für den Wert NO_ID geben, z. B. der höchste Ganzzahlenwert
IdList	Eine Liste oder ein Feld von ID's
int	Eine vorzeichenbehaftete Ganzzahl (integer)
intList	Eine Liste oder ein Feld von Ganzzahlen (integers)
real	Ein Typ, der Gleitkommazahlen (real values) enthält
RtnCode	Aufzählung der möglichen Rückgabewerte für API-Funktionen (siehe Tabelle 3)
Samples	Ein Feld der Größe „SAMPLE_BUFFER_SIZE“ (siehe 5.2 und Tabelle 7), das Audiosamples enthält
string	Eine Zeichenkette

Tabelle 2 (fortgesetzt)

Name	Beschreibung
uint	Eine nicht vorzeichenbehaftete Ganzzahl (unsigned integer)
WordId	Eine Struktur zur Identifizierung eines Wortes, bestehend aus einer Vokabular-ID und der Schreibweise des Wortes (siehe Tabelle 4)
WordIdList	Eine Liste von WordId-Strukturen
WordType	Eine Auflistung der möglichen Worttypen (siehe Tabelle 3)

Tabelle 3 – Aufzählungs-Datentypen

Name	Elemente	Beschreibung
ApiStateCode	NO_STATE IDLE RECOGNIZING RECORDING REPLAYING	API-Zustände (siehe auch Bild 3). NO_STATE bedeutet „API ist nicht initialisiert“.
EventCode	NO_EVENT BEGIN_OF_SPEECH RECOG_STARTED RECOG_STOPPED RECORDING_STARTED RECORDING_STOPPED REPLAY_STARTED REPLAY_END REPLAY_STOPPED WORD_RECOGNIZED END_DETECTED TIMEOUT OVERRUN GRAMMAR_ERROR LOST_EVENT	Ereignisse des API-Zustandsmodells (siehe Bild 3). NO_EVENT und LOST_EVENT sind Pseudo-Events des Ereignis-Ringpuffers.
RtnCode	OK NOT_ACTIVE WRONG_STATE DECODING_ERROR INCOMPLETE INVALID UNKNOWN_SPECIFIER PROTECTED OUT_OF_RANGE OUT_OF_MEMORY AUDIO_ERROR NOT_SUPPORTED	Aufzählung der Rückgabewerte für API-Funktionen; die jeweilige Bedeutung dieser Werte wird für jede Funktion in Abschnitt 6 erklärt.
WordType	SILENCE GARBAGE SPEAKER_DEPENDENT SPEAKER_INDEPENDENT ANY	Die Menge aller erlaubten Worttypen.

Tabelle 4 – Struktur-Datentypen

Name	Feldtyp	Feldname	Beschreibung
BufferDescr	id	bufferId	ID des Puffers
	int	start	Index des ersten Eintrags einer Folge relevanter Einträge
	int	end	Index des letzten Eintrags einer Folge relevanter Einträge

Tabelle 4 (fortgesetzt)

Name	Feldtyp	Feldname	Beschreibung
GraphEdge	wordId	word	Das Wort, das auf die Kante des Graphen bezogen ist, einschließlich Pausen und Fülllaute
	real	score	Die akustische Bewertung bezogen auf die Kante, unabhängig von vorangehenden Kanten und Einflüssen von Sprachmodellen
	int	startNode	Der Knoten des Graphen, an dem die Kante beginnt
	int	endNode	Der Knoten des Graphen, an dem die aktuelle Kante endet
	int	startFrame	Die Nummer des Frame bezogen auf den Startknoten (–1, falls der Frame unbekannt ist)
	int	endFrame	Die Nummer des Frame bezogen auf den Endknoten (–1 falls der Frame unbekannt ist)
	flag	isConfirmed	TRUE: Die Bewertung des Wortes liegt über dem Grenzwert CONF_THRESHOLD FALSE: Die Bewertung des Wortes liegt zwischen den Grenzwerten REJ_THRESHOLD und CONF_THRESHOLD des Wortes
WordId	Id	vocId	ID des Vokabulars, aus dem das Wort stammt
	string	spelling	Schreibweise des Wortes

4.4 Konventionen

4.4.1 Präfix als Namenszusatz

Für den korrekten Einsatz der Schnittstelle in einer anderen Programmierumgebung ist es notwendig, die Namen von Funktionen, Typen und Konstanten um ein eindeutiges Präfix zu erweitern. Die in den folgenden Definitionen verwendeten Namen enthalten aus Gründen der Lesbarkeit kein Präfix.

4.4.2 Rückgabewerte

Alle Funktionen zeigen durch ihren Rückgabewert an, ob die Operation ausgeführt wurde. Die einzige Ausnahme bilden die Verarbeitungsfunktionen HRT() und SRT(); sie geben keinen Wert zurück. Die Definition des Typs „RtnCode“ gibt an, welche Rückgabewerte möglich sind. Rückgabeparameter werden niemals über den Funktionsrückgabewert zurückgegeben, sondern über Parameter. Die jeweilige Bedeutung des Rückgabewerts wird bei jeder Funktionsdefinition beschrieben. Außer den erwähnten Werten kann eine API-Implementation einen NOT_SUPPORTED-Wert zurückgeben, wenn der zugrunde liegende Spracherkenner eine bestimmte Funktion nicht unterstützt.

4.4.3 Rückgabeparameter

Die Ergebnisse von Funktionen werden dem Aufrufer stets durch Referenz zurückgegeben. In den folgenden Definitionen sind diese Parameter durch ein '&'-Zeichen gekennzeichnet, das „Übergabe per Referenz“ bedeutet. Dieses Zeichen kann für verschiedene Realisierungen interpretiert werden: In C kann dies ein Zeiger sein, für eine Nachrichten-Schnittstelle ist die Umsetzung als Rückgabeparameter ein geeignetes Mittel.

Rückgabeparameter sollten nicht verwendet werden, wenn der Rückgabewert der aufgerufenen Funktion kein „OK“ anzeigt. Je nach der für die Umsetzung gewählten Programmiersprache und -umgebung müssen Regeln für Speicherzuweisungen genau befolgt werden.

5 Parameter

5.1 Wortabhängige Parameter

Auf jedes Wort beziehen sich eine Reihe von Parametern, die mit VocWordSetParameter() und VocWordGetParameter() gesetzt und gelesen werden können. Wenn ein neues Wort trainiert wird, werden die Werte dieser Parameter automatisch bestimmt. Alle Wort-Parameter werden in Vokabularbeschreibungen eingefügt (siehe VocGet()- und VocSet()-Funktionen). Zusätzlich zu den Parametern, die in Tabelle 5 beschrieben sind, kann es weitere spracherkennerabhängige Parameter geben. Es wird den Herstellern empfohlen, die Namen zusätzlicher Parameter durch ein spezielles Präfix zu kennzeichnen.

Die wortbezogenen Standardparameter sind in Tabelle 5 zusammengefasst:

Tabelle 5 – Wortabhängige Parameter

Name	Beschreibung	
WORD_TYPE	WordType	Ob das Wort sprecherabhängig oder sprecherunabhängig ist, oder ob es als Fülllaut („Garbage“) oder Pause („Silence“) interpretiert werden soll.
REJ_THRESHOLD	real	Die niedrigste Bewertungsstufe, bei der das Wort noch akzeptiert wird.
CONF_THRESHOLD	real	Eine Bewertungsstufe, unterhalb derer das Wort bestätigt werden sollte.

5.2 Allgemeine Parameter

Das Verhalten des Spracherkenners kann durch eine Reihe allgemeiner Parameter beeinflusst werden. Für verschiedene Applikationen kann es notwendig sein, die Werte dieser Parameter zu ändern. Jeder Erkenner verfügt bereits über eine Standardeinstellung für alle allgemeinen Parameter. Diese kann von der Applikation mit der Funktion SetParameter() geändert werden. Die Werte allgemeiner Parameter können auch dadurch überschrieben werden, dass eine Parameterblockbeschreibung mit Hilfe der Funktion SetParamBlock() in den Erkenner eingegeben wird. Ein Erkenner kann bestimmte Parameter mit einem Schreibschutz versehen.

Tabelle 6 – Allgemeine Parameter

Name	Beschreibung	
ASYNC_AUDIO ^a	flag	Dieser Parameter steht Spracherkennern zur Verfügung, die ein asynchrones Abspielen von Audiodaten unterstützen. Ist dieser Parameter auf TRUE gesetzt, werden die Audiodaten asynchron wiedergegeben; ist er auf FALSE gesetzt, werden die Audiodaten synchron abgespielt.
AUDIO_BUFFER ^a	Id	ID eines Audiopuffers, der beim nächsten Erkennungs- oder Trainingsdurchgang zur Aufnahme des Audiostroms verwendet wird. Der Puffer kann von den Funktionen des Ansagepaketes (Announcement Package) genutzt werden. Die ID eines neuen Puffers kann mit der Funktion Audio-GetFreeBuffer() ermittelt werden. Default: NO_ID, d. h. keine Aufnahme von Audioströmen.
AUTO_ADAPTATION	flag	Ist dieses Flag auf TRUE gesetzt, passt sich der Erkenner automatisch an den Sprecher an. FALSE schaltet die automatische Adaption aus, ermöglicht jedoch eine überwachte Adaption, die durch die VocAdapt()-Funktion aufgerufen bzw. gesteuert werden kann.
GRAMMAR_START_STATE ^b	Id	Der Grammatikzustand, bei dem der Erkenner anfängt, wenn er gestartet wird. Nach einem Erkennungsdurchgang bestimmt der Übergang bezogen auf das letzte Wort, das in den Wort-Ringpuffer geschrieben wurde, den Wert des Parameters. Zu Beginn: NO_ID, wenn keine Grammatik aktiv ist, sonst die ID des Grammatikzustands, der als erster erzeugt wurde.
LEAD_TIME	integer	Dieser Parameter gibt die Anzahl der Audiosamples an, die vom Zeitpunkt des Aufrufs der Funktion RecogStart() aus rückwirkend für die Erkennung mit zu berücksichtigen sind (Vorlaufzeit). Ist eine Vorlaufzeit eingestellt, muss die HRT()-Funktion auch dann aufgerufen werden, wenn der Erkenner im Leerlauf (IDLE-Zustand) ist.

Tabelle 6 (fortgesetzt)

Name	Beschreibung
RECOG_INPUT ^a	BufferDescr Eine Struktur mit den folgenden Elementen: bufferId: ID des Puffers, von dem die Spracheingabe für den nächsten Erkennungsdurchgang genommen wird. Default: NO_ID, Eingabe über HRT(). start: Nummer des Abtastwertes im Puffer, von dem aus die Erkennung zu starten ist. Default: 0, d. h. erster Abtastwert. end: Nummer des Abtastwertes im Puffer, bei dem die Erkennung zu stoppen ist. Default: MAX_INT, d. h. letzter Abtastwert.
^a Parameter nur mit Ansagepaket verfügbar ^b Parameter nur mit Sprachmodellpaket verfügbar	

Die in Tabelle 7 aufgeführten Parameter legen einen Kanal (oder mehrere Kanäle) für die Audioeingabe fest.

Tabelle 7 – Parameter für die Audioeingabekanäle

Name	Beschreibung
AUDIO_CHANNEL_ID	Id ID des Audiokanals, für den die folgenden vier Parameter aktiviert sind. Der zulässige Bereich ist $0 \leq ID < NO_OF_AUDIO_CHANNELS$, Default ist 0.
SAMPLE_BUFFER_SIZE	integer Anzahl der Samples, die mit der Funktion HRT() in den Samplepuffer übertragen werden sollen. Dieser Parameter gilt für den mit AUDIO_CHANNEL_ID angegebenen Kanal.
SAMPLE_WIDTH	integer Größe eines Samples in Bits, z. B. 16 Bits. Dieser Parameter gilt für den mit Audio_Channel_Id angegebenen Kanal.
SAMPLES_PER_SRT	integer Anzahl der Samples, die durch Aufruf der Funktion SRT() abgearbeitet wurden. Dieser Parameter gilt für den mit AUDIO_CHANNEL_ID angegebenen Kanal.
SAMPLING_RATE	integer Frequenz in Hz, bei der die Audiodaten geholt werden, z. B. 8000. Dieser Parameter gilt für den mit Audio_Channel_Id angegebenen Kanal.

5.3 Spracherkennerspezifische Parameter

Es gibt eine Reihe von Parametern, die speziell für bestimmte Spracherkennung gelten. Die Schnittstelle ermöglicht es, solche zusätzlichen Parameter genau wie die oben erwähnten Parameter zu behandeln: Die Funktionen für die Parametersteuerung SetParameter() und GetParameter() verwenden eine Zeichenkette zur Identifizierung der Parameter. Es wird den Herstellern empfohlen, die Namen zusätzlicher spracherkennerspezifischer Parameter durch ein spezielles Präfix zu kennzeichnen.

Falls die Applikation unabhängig von dem zugrunde liegenden Spracherkennungssystem laufen soll, muss sie allgemeine und spezifische Parameter unterscheiden können. Allgemeine Parameter müssen von allen Spracherkennern unterstützt werden, die der VOC-API-Norm entsprechen. Spezifische Parameter sind systemgebunden. Die Applikation muss zumindest die Version der Schnittstelle prüfen (mit der GetAPIVersion()-Funktion), bevor sie versucht, spezifische Parameter zu manipulieren.

6 Die API-Funktionen

6.1 Allgemeines

Dieser Abschnitt definiert alle VOC-API-Funktionen. Die Funktionen sind in verschiedene Pakete aufgeteilt. Die ersten drei Pakete dienen allgemeinen Aufgaben, der Vokabular-Manipulation und Erkennung. Sie umfassen Grundfunktionen, die von allen Erkennenarten unterstützt werden müssen. Die anderen Pakete sind optional und können für Erkennen mit funktionalen Erweiterungen aktiviert werden, damit diese über die Schnittstelle auch zugänglich sind. Je nach Erkennen ist die Schnittstelle in der Lage, die Steuerung der Verarbeitung, ein Sprachmodell, sprecherabhängige Benutzerwörter und Audioausgaben (Ansagen) zu unterstützen.

6.2 Überblick

Tabelle 8 – Liste der VOCAPI-Funktionen

Paket	Funktion
general	Init(callBackFunction) Exit() GetAPIVersion(&versionString, &compatibilityStamp) SetParameter(parameter, value) GetParameter(parameter, &value) SetParamBlock(paramBlock) GetParamBlock(¶mBlock, &length) GetState(&state, &attention) GetEvent(&event) ClearEventQueue()
vocabulary	StructGetVocs(&vocIdsSI, &vocIdsSD) VocGetVersion(vocId, &vocName, &compatibilityStamp) VocGetAllWords(vocId, &wordList) VocWordGetParameter(word, parameter, &value) VocWordSetParameter(word, parameter, value) VocSet(vocabulary, &vocId) VocGet(vocId, &vocabulary, &length) VocDelete(vocId) VocSetActiveWords(wordList) VocAdapt(vocId, wordList, bufferId)
recognition	RecogStart() RecogStop() RecogGetWord(&word) RecogGetGraph(&edges)
processing (optional)	HRT(samples, channelId) SRT() ClearSampleBuffers()
language model (optional)	FsgGetFreeState(&stateId) FsgNewState(stateId, wordList, targetStateList, frequencyList) FsgDeleteState(stateId) FsgAddWord(startStateId, word, targetStateId, frequency) FsgDeleteWord(stateId, word) FsgCheck(&stateId, &word, &targetStateId)
user words (optional)	VocNew(vocName, &vocId) TrainInit(word) TrainRecordStart() TrainRecordStop(accept) Train(accept) VocWordDelete(word)
announce- ments (optional)	AudioGetFreeBuffer(&bufferId) AudioSet(bufferId, audio) AudioGet(bufferId, &audio, &length) AudioPlay(bufferIdList) AudioStop() AudioDelete(bufferId)

Die in Tabelle 8 aufgelisteten Funktionen werden in 6.3.2 bis 6.3.11 ausführlich beschrieben. Die Beschreibungen entsprechen dem folgenden Schema. Für Syntaxausdrücke wird Pseudo-Code verwendet.

Name der Funktion

Syntax:	Return Type Function Name (type1 type2) parameter1, &parameter2)
Parameter:	parameter1: &parameter2: Erklärung des ersten Parameters. Erklärung des zweiten Parameters. Das ' & ' Zeichen vor dem Parameter bedeutet „Übergabe per Referenz“ anstatt „Übergabe des Parameterwertes“.
Rückgabewert:	Erklärung möglicher Rückgabewerte.
Beschreibung:	Verwendungszweck und Verhalten der Funktion.

6.3 Allgemeine Funktionen

6.3.1 Allgemeines

Die allgemeinen Funktionen umfassen Grundoperationen wie das Prüfen der Schnittstellenversion, das Öffnen und Schließen der Schnittstelle, die Handhabung von Erkennungsparametern und die Beobachtung von Zuständen der Schnittstellen sowie Erkennungsereignissen.

6.3.2 Init

Syntax:	RtnCode Init (CallbackFunction callbackFunction)
Parameter:	callbackFunction: Eine Referenz auf eine Funktion, die vom Erkennen ausgeführt wird, wenn ein Ereignis eintritt. Der Grund für den Aufruf, z. B. ein Wort im Wort-Ringpuffer, wird durch einen Parameter angezeigt, der an die Callback-Funktion weitergegeben wird (siehe nachfolgende Beschreibung). Eine Null-Referenz schaltet den Callback-Mechanismus aus.
Rückgabewert:	OK: OUT_OF_MEMORY: AUDIO_ERROR: Die Schnittstelle ist bereit. Zur Aktivierung des API fehlen Ressourcen. Audiozugriff kann nicht hergestellt werden.
Beschreibung:	Diese Funktion initialisiert die Schnittstelle. Die Funktion Init() muss vor allen anderen API-Funktionen aufgerufen werden; nur GetAPIVersion() kann auch vorher benutzt werden. Init() kann auch zum Reset der Erkennersoftware verwendet werden. Wenn eine Callback-Funktion aktiviert wurde, hat die angegebene Funktion dem folgenden Prototyp zu entsprechen: void callbackFunction(uint attention) Der „attention“-Parameter zeigt den Grund für das Aufrufen dieser Funktion. Er wird wie der „attention“-Parameter von GetState() interpretiert: Bit 0: Wenn gesetzt, enthält der Wortpuffer mindestens ein Wort. Bit 1: Wenn gesetzt, enthält der Ereignispuffer mindestens ein Ereignis.

6.3.3 Exit

Syntax:	RtnCode Exit (void)
Parameter:	Keine.
Rückgabewert:	OK: NOT_ACTIVE: Die Schnittstelle ist abgeschaltet. Die Schnittstelle wurde nicht aktiviert.
Beschreibung:	Diese Funktion schließt die Schnittstelle. Nach dem Aufrufen der Exit()-Funktion können außer der Funktion GetAPIVersion() und Init() keine anderen Funktionen mehr aufgerufen werden. Wird Exit() aufgerufen, während das API nicht im Leerlauf ist, kann die Abschaltung verzögert werden.