Fernfachhochschule Schweiz, Brig
Studiengang Informatik

Features von ECMAScript-basierten Programmiersprachen

Eine vergleichende Analyse

2. Überarbeitete Fassung vom 20. Juli 2013

Bachelor-Thesis im Studiengang Informatik

von

Thomas Lahn

Ursprünglich eingereicht bei:	**Ursprünglicher Referent:**
Prof. Dr. Martin Sutter Departement Informatik Departementsleiter	Heinrich Zimmermann Prof. Dr. Dozent of HCI

Eingereicht: Zollikofen, 12. März 2012
Verteidigt: Bern, 26. April 2012

Zusammenfassung

ECMAScript ist ein Standard für objektorientierte Programmiersprachen. Seine Implementierungen, wie JavaScript und JScript, weisen oberflächlich betrachtet grosse Ähnlichkeiten auf. Sie werden daher oft verallgemeinernd diskutiert. Doch hält dieser Ansatz einer genaueren Untersuchung stand?

In dieser Arbeit werden Features von ECMAScript und gebräuchlichen ECMAScript-Implementierungen auf ihre Standardkonformität und Kompatibilität untereinander untersucht. Es wird eine Web-Applikation für die Erfassung von Features und dazugehörigen Testfällen erstellt. Mit der Web-Applikation werden die Testfälle in Web-Browsern ausgeführt und die Ergebnisse werden in einer Datenbank gespeichert. Die Ergebnisse werden in Relation zum Verbreitungsgrad und Herstellersupport dieser Web-Browser gesetzt. Features können so als für die Verwendung in Web-Browsern sicher definiert werden, und effizienzmindernde Kompatibilitätsmassnahmen können für diese entfallen.

Abstract

ECMAScript is a standard for object-oriented programming languages. Superficially, its implementations like JavaScript and JScript are very similar. They are therefore often discussed in a simplified way. But does this approach hold water against methodical testing?

In this work, features of ECMAScript and its common implementations are compared with regard to their standards compliance and compatibility. A web application is created for the registration of features and associated test cases. Using this application, the test cases are run in web browsers and the test results are stored in a database. The results are correlated with the market share of the used web browsers and their degree of vendor support. Thus, features can be defined as safe for use in web browsers, which makes efficiency-reducing compatibility measures unnecessary.

Ich möchte den Personen und Organisationen danken, ohne die diese Arbeit nicht möglich gewesen wäre:

Meiner Familie, die mich während des halben Arbeitsweihnachtens geduldig ertragen hat, für ihre immerwährende Unterstützung; meinem Betreuer Herrn Prof. Dr. Heinrich Zimmermann sowie Herrn Dr. Urs-Martin Künzi von der Fernfachhochschule Schweiz für ihre freundliche Unterstützung während des Bachelor-Studiums und dafür, dass ich ein eigenes Thema vollständig bearbeiten durfte; den Geschäftsführern und Mitarbeitern der Dynamic Solution AG in Lyss für ihre Unterstützung, insbesondere Herrn Thomas Kocher für die grosszügige Arbeitszeitregelung und Bereitstellung von Ressourcen; den Regulars der Newsgroups `comp.lang.javascript` und `de.comp.lang.javascript` für ihre Tests und anregenden Diskussionsbeiträge; STEGelectronics Zollikofen für den Notebook-Support in vorletzter Minute; Corsin Capol für seine LaTeX-Empfehlung; der Eclipse Foundation und dem TeXlipse-Team; Markus Kohm für KOMA-Script, Dr. Leslie Lamport für LaTeX, und Prof. Donald E. Knuth für TeX.

Ausserdem danke ich Christoph ‘Mehdorn’ Weber für den Server-Support, für seine guten Ratschläge und für seinen einmaligen Humor, der vieles leichter gemacht hat.

Meinem Vater gewidmet, der mich gelehrt hat, die richtigen Fragen zu stellen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface
CDATA Character Data (SGML/XML-Datentyp)
DBMS Database Management System
DOM Document Object Model
E4X ECMAScript for XML
HTML HyperText Markup Language
HTTP HyperText Transfer Protocol
IIS Internet Information Server/Services (Microsoft)
JIT Just-In-Time
JSON JavaScript Object Notation
KDE K Desktop Environment
KJS KDE JavaScript
LAMP Linux, Apache, MySQL, PHP
MIME Multipurpose Internet Mail Extensions
MDN Mozilla Developer Network
MSDN Microsoft Developer Network
MSHTML Microsoft HTML Component
MVC Model-View-Controller
NES Netscape Enterprise Server
ORM Object-Relational Mapper
PCRE Perl Compatible Regular Expressions
PDO PHP Data Objects
PDT PHP Development Tools
PHP PHP Hypertext Preprocessor
SGML Standard Generalized Markup Language
SQL Structured Query Language
TC Technical Committee
URI Uniform Resource Identifier (RFC 3986)
VM Virtual Machine
WWW World Wide Web
XML Extensible Markup Language

1. Einführung

Die ECMAScript Language Specification (*ECMAScript*) ist ein Standard für objektorientierte Programmiersprachen.[1] Basierend darauf gibt es mehrere *Implementierungen*; zu den bekanntesten zählen derzeit JScript, JavaScript, V8, JavaScriptCore und KJS.[2] Sie werden aufgrund ihrer Gemeinsamkeiten oft verallgemeinernd unter einem Sammelbegriff diskutiert.[3,4]

Die Praxis zeigt jedoch, dass ECMAScript-Implementierungen trotz vieler Gemeinsamkeiten auch einige Unterschiede in ihrem Funktionsumfang (*Features*) aufweisen. Die Ursachen hierfür liegen einerseits in der historischen Entwicklung, die zum Standard und seinen Implementierungen geführt hat, sowie in den Freiheiten, die der Standard seinen konformen Implementierungen lässt[5]. Andererseits sind – jenseits dieser Freiheiten – nicht alle Implementierungen in allen Features standardkonform.

Ein gutes Beispiel dafür sind Objekt-Initialisierer in JScript und JavaScript. In JScript stellt ein Komma am Ende eines `Object`-Initialisierers (`{ ..., }`) bis einschliesslich Version 5.7 einen Syntaxfehler dar, jedoch in keiner Version von JavaScript. Ein alleinstehendes Komma am Ende eines `Array`-Initialisierers (`[ ..., ]`) vergrössert in JScript-Versionen vor 9.0 (oder in JScript 9.0 im MSHTML[6]-Kompatibilitätsmodus) die Länge des Arrays um 1; in JScript 9.0 im standardkonformen Rendering-Modus und in allen JavaScript-Versionen jedoch nicht. Im Fall des `Object`-Initialisierers ist ECMAScript Edition 3 relevant, nach der das Komma an dieser Stelle syntaktisch falsch ist[7]; anders spezifiziert ist es seit

[1] ECMA INTERNATIONAL 1997.
[2] siehe Abschnitt 2.2 auf Seite 20
[3] CROCKFORD 2008; FLANAGAN 2011; GOODMAN u. a. 2007; RESIG 2006.
[4] SMITH 2010, Abschnitt “2.1 What is ECMAScript?”
[5] ECMA INTERNATIONAL 2011a, Abschnitt “2 Conformance”.
[6] Microsoft HTML Component
[7] ECMA INTERNATIONAL 2000, Abschnitt 11.1.5.

Edition 5[8]. Für `Array`-Initialisierer spezifizieren hingegen alle relevanten ECMAScript-Editionen, dass ein *einzelnes* Komma am Ende des Initialisierers keinen Einfluss auf die Länge des erzeugten Arrays hat.[9]

Das heisst, schon *ein* zusätzliches, vom Software-Entwickler vielleicht nur versehentlich eingegebenes Zeichen in einem ECMAScript-basierten Programm kann dazu führen, dass das Programm mit einer Implementierung nicht lauffähig ist – jedoch möglicherweise mit einer anderen –, oder dass es ein von der Implementierung abhängiges Laufzeitverhalten zeigt.

ECMAScript-Implementierungen sind jedoch *Skriptsprachen*. Das heisst, die für die Ausführung eines ECMAScript-basierten Programms (*Skript*) verwendete Implementierung wird durch die Umgebung bestimmt, in der das Programm ausgeführt wird (seine *Host-Umgebung*). Oft ist die Host-Umgebung ein Web-Browser.[10]

1.1. Motivation

Der Autor eines Skripts kann also nicht ohne weiteres davon ausgehen, dass das in einer Host-Umgebung positiv getestete Skript auch in einer anderen genauso oder überhaupt funktioniert. Dies ist *unabhängig* von möglichen Unterschieden in der Programmierschnittstelle (API), welche die Host-Umgebung gegebenenfalls für den Zugriff mit einer ECMAScript-Implementierung bereitstellt (wie z. B. dem DOM[11] in Web-Browsern).

Abweichungen vom Standard und Unterschiede zwischen den Implementierungen erfordern Kompatibilitätsmassnahmen für interoperable Programme. Solche Massnahmen führen jedoch zu weniger effizienten Programmen (siehe Abschnitt 2.3). Gleichzeitig führen unbekannte Abweichungen und nicht berücksichtigte Inkompatibilitäten zu weniger robusten Programmen und zu Programmen, die kaum oder nicht interoperabel sind.

[8] ECMA INTERNATIONAL 2009, Abschnitt 11.1.5.
[9] ECMA INTERNATIONAL 2000; ECMA INTERNATIONAL 2009; ECMA INTERNATIONAL 2011a, jeweils Abschnitt 11.1.4.
[10] ECMA INTERNATIONAL 2011a, Abschnitt “4 Overview”.
[11] Document Object Model

Diese Arbeit untersucht, inwieweit eine verallgemeinernde Diskussion unter einem der genannten Sammelbegriffe gerechtfertigt ist. In welchem Mass sind ECMAScript-Implementierungen standardkonform, und inwieweit unterscheiden sie sich in ihren Features voneinander? Müssen verschiedene ECMAScript-Implementierungen jeweils als eigene Programmiersprache betrachtet werden?

Wenn festgestellt werden kann, dass Features in verschiedenen relevanten Implementierungen zueinander kompatibel oder sogar standardkonform implementiert sind, dann können für diese Features die entsprechenden Kompatibilitätsmassnahmen entfallen. Dies führt zu *effizienteren* Programmen. Sollten andererseits jedoch bislang unbekannte Inkompatibilitäten festgestellt werden, so können dagegen *gezielt* Kompatibilitätsmassnahmen ergriffen werden. Dies führt zu *robusteren*, *interoperableren* Programmen.

1.2. Ansatz

Basierend auf einer früheren Arbeit[12] wird in dieser Arbeit ein neuartiger Ansatz verfolgt. Testfälle, welche geeignet sind, nicht nur die Existenz, sondern auch die *Funktionalität* von Features zu ermitteln, werden mit verschiedenen Versionen verschiedener Implementierungen in verschiedenen Web-Browsern ausgeführt. Dabei werden auch *proprietäre* Features einzelner Implementierungen berücksichtigt, die den Standard erweitern, soweit diese dokumentiert sind.

Die Testergebnisse werden für die spätere Auswertung in einer Datenbank *gespeichert*. Aus der Datenbank wird eine *Kompatibilitätsmatrix* in HTML[13] generiert. Diese gibt Auskunft über die Funktionalität von Features in der vom aktuellen Browser verwendeten ECMAScript-Implementierung sowie über die Testergebnisse mit anderen Implementierungen. Sollten Testfälle für ein Feature nachträglich geändert werden, dann werden die vorherigen Testergebnisse für dieses Feature automatisch invalidiert.

Durch die Kombination der Testergebnisse mit Informationen zur Verbreitung und Hersteller-Unterstützung der getesteten Browser-Versionen wird gezeigt, dass sich gebräuchliche

[12] LAHN 2011.
[13] HyperText Markup Language

ECMAScript-Implementierungen auch heute noch in wichtigen Features in ihrer Standardkonformität und Kompatibilität untereinander unterscheiden. Es gibt Features, welche bezüglich Kompatibilität im Web inzwischen als unbedenklich (*sicher*) betrachtet werden dürfen und deshalb in diesen Host-Umgebungen effizienzsteigernd direkt verwendet werden können; Features die weiterhin effizienzmindernder, aber die Interoperabilität von Programmen verbessernder Kompatibilitätsmassnahmen bedürfen, und Features, die zur Zeit höchstens eingeschränkt verwendet werden sollten.

1.3. Verwandte Arbeiten

Obwohl ECMAScript-Implementierungen inzwischen vielfältige Anwendungen gefunden haben[14], findet eine adäquate Diskussion der verschiedenen Implementierungen bisher kaum statt. Insbesondere findet sich aufgrund der eingangs erwähnten vereinfachenden Beschreibung in der wissenschaftlichen und sonstigen Fachliteratur so gut wie keine Information zur Standardkonformität von Implementierungen und zur Kompatibilität von Implementierungen untereinander.

Es gibt jedoch im WWW[15] bereits zwei zu dieser Arbeit ähnliche Untersuchungen zu ECMAScript-Implementierungen, die an dieser Stelle erwähnt werden sollen.

1.3.1. Wissenschaftliche Arbeiten

Wissenschaftliche Arbeiten zu ECMAScript und ECMAScript-Implementierungen konzentrieren sich – hauptsächlich unter dem Namen «JavaScript» diskutiert – vor allem auf Sicherheitsaspekte[16] und semantische Fragestellungen[17]. Es gibt anscheinend bisher keine Arbeiten, welche die Unterschiede zwischen ECMAScript-Implementierungen genauer untersuchen.

[14] u. a. NEUMANN 2005; BÜHLER u. a. 2004; VIGNA 2002; NEUBAUER u. a. 2004; RAULET u. a. 2008; WOLF u. a. 2009.
[15] World Wide Web
[16] YU u. a. 2007; GUARNIERI u. a. 2009; CHUGH u. a. 2009.
[17] ANDERSON u. a. 2005; MAFFEIS u. a. 2008; JENSEN u. a. 2009.

1.3.2. Sonstige Fachliteratur

Es muss an dieser Stelle festgehalten werden, dass in der sonstigen – auch in der neueren – Fachliteratur oft eine stark vereinfachende, und teilweise übersimplifizierende bis sachlich falsche Behandlung des Themas überwiegt. So findet etwa eine Unterscheidung zwischen dem ECMAScript-Standard und seinen Implementierungen JavaScript und JScript nicht oder nur unzureichend statt. Oft werden diese drei Begriffe als Synonyme für *eine* Programmiersprache vorgestellt, die lediglich verschiedene «Dialekte» in verschiedenen Browsern habe.

Zusätzlich findet oft eine Vermischung der Beschreibung der Programmiersprachen mit den APIs, die mit ihnen genutzt werden können (v. a. mit dem DOM) statt. Es überrascht daher nicht, dass in der FAQ der Newsgroup comp.lang.javascript nur zwei Bücher zum Thema *bedingt* empfohlen werden[18].

Als mögliche Ursachen hierfür dürfen die fehlende Trennung zwischen Sprache und API der Host-Umgebung in JavaScript bis einschliesslich Version 1.3 und die Tatsache, dass – dem ursprünglichen Ziel von JavaScript entsprechend[19] – besonders Programmieranfänger die Zielgruppe der entsprechenden Literatur sind, angenommen werden. Einige Werke seien der Vollständigkeit halber nachfolgend erwähnt.

CROCKFORD stellt in seinem Buch die “good parts” von “JavaScript” vor.[20] Darunter versteht er die weniger fehlerträchtigen Features *einer* Programmiersprache, die er in ECMAScript standardisiert sieht und mit JScript gleichsetzt. Eine Diskussion der “good parts”, die einige ECMAScript-Implementierungen zusätzlich zum Standard bereitstellen, wie etwa das Destructuring Assignment seit Mozilla JavaScript 1.7, und ein Vergleich von Features zwischen Implementierungen, findet daher nicht statt.

Auch FLANAGAN geht davon aus, dass es sich bei “JavaScript” um *eine* Programmiersprache handele, die von allen modernen Web-Browsern mittels “JavaScript interpreters” unterstützt werde.[21] Trotz der Aktualität des Werkes geht er nicht auf die Unterschiede

[18] SMITH 2010.
[19] siehe Abschnitt 2.1.2
[20] CROCKFORD 2008.
[21] FLANAGAN 2011.

zwischen aktuellen ECMAScript-Implementierungen – darunter einige, die zu nativem Maschinencode kompilieren (siehe Abschnitt 2.2.6) – ein.

GOODMAN gibt ebenfalls eine Einführung zu *einer* Sprache, die er unter dem Namen "JavaScript" subsubmiert.[22] In den einführenden Kapiteln erklärt er jedoch die Zusammenhänge zwischen JavaScript und JScript und geht auf die inzwischen auch in (Mozilla) JavaScript erfolgte Trennung zwischen Kernsprache und DOM API ein.

JAWORSKI vergleicht JavaScript bis einschliesslich Version 1.3, JScript 3.1 und 5 und ECMAScript Ed. 1 miteinander.[23] Unter "JScript" versteht er jedoch die Summe der Features von JScript und des MSHTML-DOMs, obwohl es sich hierbei um unterschiedliche *Arten* von Implementierungen handelt.

KEITH beschreibt ECMAScript-Implementierungen mit Erweiterungen als "JavaScript". Er beschreibt JavaScript 1.1 als *eine* von Netscape Navigator *und* Microsoft Internet Explorer unterstützte Programmiersprache[24], obwohl Internet Explorer die Implementierung Microsoft JScript verwendet.

RESIG beschreibt Ansätze für "Modern JavaScript Programming" und bezieht sich damit anscheinend nur auf Mozilla JavaScript.[25] Er beschreibt die Programmierung im Web-Kontext mittels DOM API[26]. Auf die Tatsache, dass (Mozilla) JavaScript nur eine von mehreren in Web-Browsern eingesetzten Implementierungen von ECMAScript ist, und auf die Unterschiede zwischen diesen Implementierungen, geht er nicht ein.

1.3.3. Webressourcen

Das ECMA-TC39 stellt auf seiner Website "ecmascript test262" Testfälle bereit, um die Standardkonformität der von dem aktuellen Browser verwendeten Implementierung zu testen.[27] Die Testfälle sind sehr umfangreich (zum Zeitpunkt der Erstellung dieser Arbeit

[22] GOODMAN u. a. 2007.
[23] JAWORSKI 1999.
[24] KEITH u. a. 2011.
[25] RESIG 2006.
[26] Application Programming Interface
[27] ECMA INTERNATIONAL 2012.

sind es 11563). Wie in dieser Arbeit werden die Testfälle gegeneinander isoliert. Dabei wird jedoch jeder Testfall nicht nur in einem eigenen `SCRIPT`-Element, sondern auch in einer eigenen Skript-Ressource ausgeführt. Anhand der Ergebnisse (Tab “Results”) ist ein direkter Vergleich der Unterstützung von Features in Implementierungen allerdings nur mit grösserem Aufwand möglich, da die Testergebnisse zur Ansicht aufbereitet, aber nicht gespeichert werden. Ermittelte Testergebnisse sind zudem vergleichsweise unzuverlässig, da die Testfälle aufgrund aktiver Entwicklung ständigen Veränderungen unterliegen können (“Because tests are being actively added and modified, tests results from different days or times may not be directly comparable.”[28]).

ZAYTSEV untersucht auf seiner Website die Kompatibilität von Features, die mit ECMAScript Edition 5 eingeführt wurden; von Features, die mit Edition 6 eingeführt werden sollen, und auch von proprietären Features.[29] Es wird jedoch die Kompatibilität von Features zwischen Browser-Versionen miteinander verglichen, nicht zwischen ECMAScript-Implementierungen. Der überwiegende Anteil der für den Vergleich verwendeten Tests sind Existenztests, keine Funktionstests (“Please note that these tests represent existence, not functionality or full conformance.”; vgl. Abschnitt 3.1.2.4 auf Seite 45). ZAYTSEV verwendet bei Funktionstests ebenfalls die `eval`-Methode, um die Testfälle gegeneinander zu isolieren. Es finden sich jedoch keine genaueren Angaben zu den getesteten Umgebungen und der verwendeten Testmethode.

1.4. Struktur dieser Arbeit

Kapitel 1 arbeitet die Motivation für diese Arbeit heraus und gibt eine kurze Zusammenfassung des Inhalts. Kapitel 2 stellt ECMAScript und die getesteten ECMAScript-Implementierungen vor. Kapitel 3 beschreibt die für die Vorbereitung und Durchführung der Tests erstellte Web-Applikation, sowie die Methoden, die bei der Erstellung der Testfälle und Durchführung der Tests angewendet wurden. Kapitel 4 stellt die wichtigsten Resultate vor. Kapitel 5 schliesslich diskutiert die Implikationen der Resultate und gibt einen Ausblick für die zukünftige Forschung auf diesem Gebiet.

[28] ECMA INTERNATIONAL 2012.
[29] ZAYTSEV 2012.

1.4.1. Textauszeichnung

In dieser Arbeit wird die folgende Textauszeichnung verwendet:

- *Hervorhebung/wichtiger Begriff*
- `Quelltext`
- *`Variable im Quelltext`*

2. ECMAScript und ECMAScript-Implementierungen

Dieses Kapitel beschreibt ECMAScript und zeigt die historische Entwicklung auf, die zu den zu vergleichenden ECMAScript-Implementierungen geführt hat.

2.1. ECMAScript

Die *ECMAScript Language Specification (ECMAScript)* ist ein unter der Registriernummer ECMA-262 von der Ecma International[1] veröffentlichter Standard[2] für eine Klasse objektorientierter Skriptsprachen mit prototypbasierter Vererbung und schwacher Typisierung, die grundsätzlich dem prozedural-imperativen Programmierparadigma folgen[3].

2.1.1. Allgemeines

Der Standard wird in *Editionen* veröffentlicht. Die aktuelle Edition ist Edition 5.1.[4] Heute basieren mehrere Programmiersprachen auf ECMAScript. Diese werden als ECMAScript-*Implementierungen* bezeichnet.[5]

[1] `http://www.ecma-international.org/`
[2] ECMA INTERNATIONAL 2011b.
[3] ECMA INTERNATIONAL 2011a, Abschnitt “4.2 Language Overview”.
[4] ECMA INTERNATIONAL 2011b.
[5] ECMA INTERNATIONAL 2011a, Abschnitt “2 Conformance”.

Die erste Edition von ECMAScript basierte jedoch ihrerseits auf gemeinsamen Features zweier implementierter Programmiersprachen: Netscape JavaScript, Version 1.1, und Microsoft JScript, Version 1.0.[6] Um zu verstehen, wie ECMAScript entstand und was es ausmacht, ist es daher erforderlich, zunächst die Ursprünge dieser beiden Programmiersprachen zu betrachten. (Auf die Merkmale von ECMAScript selbst und die Editionen von ECMAScript wird in Abschnitt 2.1.4 auf Seite 13 näher eingegangen.)

2.1.2. Netscape JavaScript 1.0 und 1.1 (1995/1996)

JavaScript ist eine von Brendan Eich ursprünglich bei der Netscape Communications Corporation (Netscape) entwickelte Programmiersprache. Ursprünglich war JavaScript nur als clientseitige Skriptsprache in Web-Browsern gedacht.[7]

2.1.2.1. Alternative zu Java

Mit *Java* von Sun Microsystems, Inc. (Sun)[8] gab es seit Version 1.0a2 (23. Mai 1995)[9] bereits eine objektorientierte Programmiersprache für Web-Browser. Sie hatte jedoch mehrere Nachteile. Unter anderem hatte sie eine sehr komplexe Syntax und war deshalb zu schwierig zu erlernen; Quelltext musste explizit kompiliert werden; für die Verwendung dieses Java-Programms (*Applet*) im Browser musste ein Plugin installiert werden, und dieses Applet lief ohne Bezug zum Rest des HTML-Dokuments, in das es eingebettet war.[10]

Brendan Eich sollte daher für Netscape eine einfachere Programmiersprache für Programmieranfänger entwickeln, um Browserfunktionen steuern und Websites ohne Verwendung eines Plugins dynamisch machen zu können.[11]

[6] ECMA INTERNATIONAL 1997, “Introduction”.
[7] KRILL 2008; HAMILTON 2008.
[8] heute Oracle America, Inc., vgl. `http://www.oracle.com/us/corporate/press/044428`
[9] BYOUS 1998.
[10] KRILL 2008; HAMILTON 2008.
[11] GOODMAN u. a. 2007, “Foreword”.

2.1.2.2. Name

Eich entwickelte die Sprache von Mai bis September 1995 unter den Namen «Mocha» und später «LiveScript».[12] Die Sprache wurde kurz darauf von Netscape in «JavaScript» umbenannt. Netscape wollte damit einerseits die Popularität von Java zu jener Zeit ausnutzen und andererseits JavaScript als Ergänzung zu Java etablieren.[13]

Um Rechtsstreitigkeiten zu vermeiden, schloss Netscape mit Sun, dem damaligen Inhaber des Warenzeichens «Java», Anfang Dezember 1995 ein Lizenzabkommen: Sun erhielt die Rechte am Warenzeichen «JavaScript», und Netscape durfte dieses Warenzeichen mit Hinweis auf Sun als Name für ihre Programmiersprache verwenden.[14] Der ähnliche Name führt auch heute noch immer wieder zu Verwechslungen zwischen JavaScript und Java.

JavaScript 1.0 wurde im März 1996 mit dem Release von Netscape Navigator 2.0 beta 3 der Öffentlichkeit vorgestellt.[15]

2.1.2.3. Netscape JavaScript 1.1

JavaScript 1.1 wurde im August 1996 mit Netscape Navigator 3.0 released.[16] Zu den Neuerungen gegenüber Version 1.0 gehörten neue Objekte und Objekteigenschaften, sowohl was die Kernsprache als auch Host-Objekte betraf, welche der Netscape-Browser bereitstellte.[17] (Eine klare Trennung zwischen der Kernsprache und dem API der Host-Umgebung wurde erst mit JavaScript 1.4 vollzogen, siehe Abschnitt 2.2.1.)

In der Kernsprache wurden zwei neue Operatoren, `typeof` und `void`, eingeführt. (Aufgrund ihrer Abwärtskompatibilität werden diese Operatoren in den Testfällen dieser Arbeit häufig verwendet; vgl. Abschnitt 3.1 auf Seite 29.)

[12] KRILL 2008; HAMILTON 2008.
[13] FLANAGAN 2011.
[14] KRILL 2008; HAMILTON 2008.
[15] ANDREESSEN 1998.
[16] METZGER 2012.
[17] NETSCAPE COMMUNICATIONS CORP. 1996b.

Mit *LiveConnect*[18] war es nun erstmals möglich, dass JavaScript-Skripts und Java-Applets untereinander Daten austauschen bzw. sich gegenseitig steuern konnten. Dazu wurde JavaScript um eine Schnittstelle erweitert, die es ermöglichte, im Browser installierte Plug-ins zu erkennen (`navigator.plugins`). Schliesslich wurde "data tainting" eingeführt, um einerseits zu verhindern, dass Skripts normalerweise private Informationen wie die Browser-History lesen können und andererseits Autoren die Möglichkeit zu geben, auf bestimmte Komponenten eines Dokuments in sicherer Weise zuzugreifen[19].

(Zur Einhaltung einer chronologischen Reihenfolge wird nachfolgend zunächst die zweite Programmiersprache vorgestellt, deren Features in ECMAScript eingeflossen sind. Neuere Versionen von JavaScript als 1.1 werden daher im Abschnitt 2.2.1 auf Seite 21 diskutiert.)

2.1.3. Microsoft JScript 1.0 (1996)

Um gegenüber dem Konkurrenten Netscape wettbewerbsfähig zu bleiben, führte die Microsoft Corporation (Microsoft) mit Version 3.0 ihres Web-Browsers *Internet Explorer* ebenfalls eine JavaScript-ähnliche Skriptsprache ein, die letztlich ein Reverse-Engineering von JavaScript 1.0 darstellte. Um nicht das «JavaScript»Warenzeichen von Sun lizensieren zu müssen, nannte Microsoft die Sprache *JScript*[20].

Internet Explorer 3.0 und JScript 1.0 wurden fast zeitgleich mit Netscape Navigator 3.0 und JavaScript 1.1 im August 1996 released[21,22].

(Neuere JScript-Versionen werden im Abschnitt 2.2.2 auf Seite 23 diskutiert.)

[18] `https://developer.mozilla.org/en/LiveConnect`
[19] NETSCAPE COMMUNICATIONS CORP. 1996b.
[20] YUI THEATER 2007.
[21] MICROSOFT CORP. 2012.
[22] vgl. VEITCH 2001.

2.1.4. ECMAScript-Editionen (seit 1997)

2.1.4.1. Standardisierung

Edition 1 (1997) JavaScript (1.1) wurde im November 1996 von Netscape der Ecma International zur Standardisierung vorgeschlagen[23,24]. Microsoft war zu diesem Zeitpunkt jedoch ebenfalls Mitglied dieser Organisation, und war seinerseits an einer Standardisierung von JScript (1.0) interessiert. Die Folge war, dass – da man sich nicht auf einen Namen einigen konnte – der Standard, der aus gemeinsamen Features von JavaScript 1.1 und JScript 1.0 hervorging[25], *ECMAScript* genannt wurde[26]. Die erste Edition von ECMAScript wurde im Juni 1997 von der Ecma-Generalversammlung verabschiedet[27].

Edition 2 (1998) Edition 2 von ECMAScript unterscheidet sich von Edition 1 inhaltlich nicht. Änderungen sind "editorial in nature"[28].

Edition 3 (1999/2000) Edition 3 führt neue Syntaxelemente und Objekte ein[29], unter anderem:

- Operatoren
  - Typstrenge Vergleichsoperatoren (`===` und `!==`)
  - `in`-Operator zur genaueren Ermittlung von Eigenschaften eines Objekts
  - `instanceof`-Operator zur Ermittlung des Typs eines Objekts
- Ausdrücke
  - Anonyme und benannte Function Expressions (Lambdas)
  - Objekt-Initialisierer für `Array` und `Object`-Instanzen
  - Reguläre Ausdrücke
  - `undefined`-Wert

[23] NETSCAPE COMMUNICATIONS CORP. 1996a.
[24] ECMA INTERNATIONAL 1998, Abschnitt "Brief History", Seite 5.
[25] ECMA INTERNATIONAL 1997, Abschnitt "Brief History", Seite 4.
[26] KRILL 2008.
[27] ECMA INTERNATIONAL 1998, Abschnitt "Brief History", Seite 5.
[28] Ebd.
[29] ECMA INTERNATIONAL 2000.

  - Konsequente Unicode-Unterstützung
- Anweisungen
  - `do...while`
  - Benannte Anweisungen (labelled statements)
  - `switch...case...default`
  - `throw` und `try...catch...finally` (siehe Objekte)
- Objekte und Methoden
  - Zusätzliche Methoden von `Array.prototype`
  - Zusätzliche Methoden von `Date.prototype` vor allem zur Lokalisierung von Datumsangaben
  - Das `Error`-Objekt für die Behandlung von Exceptions (siehe Anweisungen)
  - Zusätzliche Methoden, um Funktionen als Methoden eines anderen Objekts aufrufen zu können (`Function.prototype.call` und `Function.prototype.apply`)
  - Flexiblere mathematische Methoden
  - Methoden für die reflexive Programmierung (u. a. `Object.prototype.hasOwnProperty` und `Object.prototype.isPrototypeOf`)
  - Das `RegExp`-Objekt (siehe Ausdrücke)
  - `String`-Methoden für reguläre Ausdrücke
  - Globale Methoden zur URI[30]-Kodierung und Dekodierung (`encodeURI`, `encodeURIComponent`, `decodeURI`, `decodeURIComponent`)

Edition 4 ECMAScript Edition 4 erweitert Edition 3 unter anderem um klassenbasierte Vererbung, modulare Programmierung und starke Typisierung[31].

Edition 4 liegt nur als Vorschlag an die Ecma International vor, da sich die Mitglieder des Ecma-TC39[32] nicht auf einen Kompromiss bezüglich dieser neuen Features einigen konnten[33]. Sie ist also kein offizieller Standard. Jedoch wurde ein Teil der darin spezifizierten Features in JScript .NET/10.0 auf Microsoft IIS[34] und ActionScript 3.0 in Adobe Flash

[30] Uniform Resource Identifier (RFC 3986)
[31] ECMA INTERNATIONAL 2007.
[32] `http://www.ecma-international.org/memento/TC39-M.htm`
[33] EICH 2008b.
[34] Internet Information Server/Services

implementiert[35].

Edition 5 (2009) Unter dem Arbeitstitel «ECMAScript 3.1» entwickelt[36], erweitert Edition 5 die eingebauten Objekte um das `JSON`-Objekt[37], neue «statische» Methoden (von `Object` und `Array`)[38] sowie Prototyp-Methoden von `Array`-Instanzen[39]. Damit wurden einige Features standardisiert, die bereits zuvor in JavaScript implementiert waren (vgl. Abschnitt 2.2.1) und von anderen Implementierungen übernommen wurden.

Zu den syntaktischen Änderungen gehört die Möglichkeit, `String`-Literale auf der nächsten Zeile fortsetzen zu können[40]. Ausserdem wird ein optionaler strenger Modus (*"strict mode"*, mit der Anweisung `"use strict"`) eingeführt. Dieser macht einige, auch sicherheitstechnisch relevante, proprietäre Erweiterungen unbrauchbar; er hilft ausserdem dabei, häufige Programmierfehler (z. B. eine Zuweisung an eine nicht deklarierte Variable) durch automatisches Auslösen von Laufzeitfehlern zu vermeiden[41].

Edition 5.1 (2010) Edition 5.1 enthält Korrekturen und Klarstellungen zum Text von Edition 5. Es werden keine neuen Features spezifiziert[42].

ECMAScript.next (Zukunft) Unter den Bezeichnungen "ECMAScript.next" und "Harmony" werden öffentlich Vorschläge zur nächsten Edition von ECMAScript diskutiert[43].

2.1.4.2. Klassifizierung

Typ ECMAScript ist eine *Skriptsprache*, denn es handelt sich um «eine Programmiersprache, die benutzt wird, um die Funktionen eines existierenden Systems zu beeinflussen,

[35] MICROSOFT CORP. 2011a; GROSSMAN u. a. 2006.
[36] EICH 2008b.
[37] ECMA INTERNATIONAL 2009, Abschnitt 15.12.
[38] Ebd., Abschnitte 15.2.3 und 15.4.3.
[39] Ebd., Abschnitte 15.4.4 und 15.4.3.
[40] Ebd., Abschnitt 7.3.
[41] Ebd., "Annex C".
[42] ECMA INTERNATIONAL 2011a, "Annex F", Seite 255.
[43] `http://wiki.ecmascript.org/`

anzupassen und zu automatisieren» (“a programming language that is used to manipulate, customise, and automate the facilities of an existing system”[44]).

Wie auch bei anderen Skriptsprachen (vgl. z. B. PHP, Perl und Python) üblich, erfordert die Ausführung eines Programms zuvor keine explizite Kompilierung. Stattdessen erfolgt die Kompilierung in der Regel implizit bei der Ausführung (JIT[45]): Das Programm wird zunächst auf syntaktische Korrektheit überprüft und in äquivalenten Bytecode umgewandelt. Anschliessend wird dieser Bytecode von einer Virtuellen Maschine (VM) interpretiert. Diese Form der Ausführung ist für ECMAScript nicht spezifiziert, wird jedoch von mehreren Implementierungen umgesetzt[46]. SquirrelFish Extreme und Google V8 hingegen JIT-kompilieren ein Programm direkt zu nativem Maschinencode (siehe Abschnitte 2.2.5 und 2.2.6).

Syntax ECMAScript verwendet viele Syntaxelemente von C. Funktionskörper (`function`) und Anweisungen (z. B. `if`, `for`, `while` und `switch`) werden in Blöcken strukturiert, die mit geschweiften Klammern (`{...}`) begrenzt werden. Der Gültigkeitsbereich einer Variablen (scope) erstreckt sich jedoch bei Anweisungen über den Block hinaus (lexikalisches Scoping).

Typisierung ECMAScript-Implementierungen sind schwach typisiert[47]. Eine Variable oder Eigenschaft kann zu jedem Zeitpunkt jeden Wert unbhängig von seinem Typ annehmen[48]. Da es sich um Skriptsprachen handelt, sind sie in der Regel ausserdem dynamisch typisiert, d. h. die Typüberprüfung findet erst zur Laufzeit statt.

[44] ECMA INTERNATIONAL 2011a, Abschnitt 4, Seite 2.
[45] Just-In-Time
[46] EICH 2010a.
[47] mit Ausnahme der auf Edition 4 basierenden Implementierungen, siehe Seite 14.
[48] Es sei denn, das Objekt hat für diese Eigenschaft einen Setter, der dies verhindert; einen Getter, der eine entsprechende transparente Konvertierung vornimmt; oder die Eigenschaft ist schreibgeschützt.

2.1.4.3. Programmierparadigmen

Prozedural-imperativ ECMAScript folgt grundsätzlich dem prozedural-imperativen Programmierparadigma: Ein Programm besteht aus Anweisungen, die in Quelltextreihenfolge ausgeführt werden.

Objektorientiert ECMAScript ist eine *objektorientierte* Programmiersprache: Daten werden als Eigenschaften von Objekten gespeichert. Die Objektorientierung ist *prototypbasiert*, d. h. Objekte erben Eigenschaften von anderen Objekten, ihren *Prototypen*. Der Zugriff auf Objekte erfolgt dabei nicht direkt, sondern über eine Eigenschaft eines anderen Objekts, welche als Wert eine *Referenz* auf das erstere Objekt hat. Das heisst, ein Objekt kann von verschiedenen Eigenschaften anderer Objekte (oder sich selbst) referenziert werden, wobei die Dereferenzierung automatisch erfolgt.

Subroutinen werden als Methoden von Objekten implementiert. Unter *Methoden* werden spezielle Eigenschaften von Objekten verstanden, deren Werte aufrufbare Objekte – *Funktionen* – referenzieren.

Funktional In dieser Sprache handelt es sich also bei Funktionen um Objekte erster Klasse: Funktionen können (über Referenzen) als Werte zugewiesen werden (d. h. auf der rechten Seite einer Zuweisung stehen), sie können über Argumente an andere Methoden übergeben werden, und sie können selbst Eigenschaften haben. Dies ermöglicht auch die Anwendung von Elementen des funktionalen Programmierparadigmas, wie z. B. Closures, Callbacks und Currying in ECMAScript-Implementierungen.

2.1.4.4. Besonderheiten

Dieser Abschnitt beschreibt besondere Merkmale von ECMAScript.

Datentypen Es wird zwischen primitiven und Objekt-Datentypen unterschieden.

Werte primitiver Datentypen werden bei Verwendung in einem Objekt-Kontext (z. B. beim Eigenschaftszugriff temporär in Werte von Objekt-Datentypen konvertiert. Somit ist etwa folgender Ausdruck möglich, der den (gerundeten) Wert von π in Binärschreibweise ergibt:

```
1 Math.PI.toString(2)
```

Hier wird der `Number`-Wert der Eigenschaft `Math.PI` implizit in eine `Number`-Instanz konvertiert, und dann die von `Number.prototype` geerbte `toString`-Methode mit dieser Instanz als `this`-Wert aufgerufen.

Ein wichtiger primitiver Datentyp ist `Undefined`, dessen einziger Wert `undefined`[49] auch das Ergebnis von Zugriffen auf nicht existente Objekteigenschaften ist[50]. Dies wird in den Testfällen dieser Arbeit genutzt, um die Existenz von Eigenschaften zu testen, deren Typ als verschieden von `Undefined` angenommen wird.

Numerische Datentypen In ECMAScript (ausser in dem Vorschlag für Edition 4) ist bisher nur *ein* numerischer Datentyp spezifiziert: `Number`. Dieser implementiert Gleitkommazahlen mit «doppelter Genauigkeit» (“double precision”, kurz “Double”) nach IEEE-754, d. h. 64 Bit Breite[51]. Dabei werden 1 Bit für das Vorzeichen, 11 Bit für den Exponenten und 52 Bit für die Mantisse benutzt[52]. IEEE-754 entsprechend gibt es die `Number`-Werte `Infinity` und `-Infinity` (Unendlichkeiten), und `NaN` (“not a number” – «keine Zahl»)[53], wobei `Infinity` bzw. `-Infinity` das Ergebnis der hier zulässigen Division `x / 0` ist, und `NaN` ungleich allen Werten ist, auch sich selbst. Zur Überprüfung, ob es sich um einen `NaN`-Wert handelt, gibt es die globale Funktion `isNaN()`. Ebenso wird (intern) zwischen den Werten $+0$ und -0 unterschieden.

[49] ECMA INTERNATIONAL 2011a, Abschnitt 8.1.
[50] Ebd., Abschnitt 11.2.1.
[51] Ebd., Abschnitte 4.3.19, 4.3.20 und 8.5.
[52] INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS 2008.
[53] ECMA INTERNATIONAL 2011a, Abschnitt 4.3.22 und 4.3.23.

Intern, z. B. bei den Operatoren <<< (shift left, Bitverschiebung nach links) und >> bzw. >>> (shift right, Bitverschiebung nach rechts) wird jedoch auch 32-Bit Ganzzahl-Arithmetik verwendet[54].

Reguläre Ausdrücke ECMAScript-Implementierungen basierend auf Edition 3 oder neuer unterstützen reguläre Ausdrücke, die PCRE[55] ähneln[56].

Deklarationen Das Objekt, welches eine Eigenschaft besitzt, muss bzw. kann nicht immer explizit referenziert werden. So führt die Deklaration von Variablen implizit zur Erstellung von Eigenschaften eines in der Regel nicht referenzierbaren[58] Variablenobjekts, welche den deklarierten Bezeichner als Namen und den Wert der Variablen als Wert haben. Hierbei wird nicht zwischen Variablen- und Funktionsdeklarationen unterschieden.

Diese Variableninstantiierung – ab ECMAScript Edition 5 «Instantiierung der Deklarationsbindung» ("Declaration Binding Instantiation"[59]) – erfolgt in Quelltextreihenfolge, jedoch vor der Ausführung der Anweisungen eines Ausführungskontextes (z. B. Zuweisungen)[60]. Das heisst es ist möglich, eine Variable nach der Deklaration explizit zu initialisieren.

Nicht initialisierte Variablen erhalten implizit den Wert `undefined`. Funktionsvariablen sind jedoch immer mit einer Referenz auf das somit erstellte Funktionsobjekt (`Function`-Instanz) initialisiert.

Parameter und Argumente Formale *Parameter* einer Funktion können im Funktionskopf deklariert werden. Der betreffenden Funktion kann aber beim Aufruf eine beliebige Anzahl Werte (*Argumente*) übergeben werden. Dies macht diese Sprachen einerseits sehr

[54] ECMA INTERNATIONAL 2011a, Abschnitt 11.7.1, 11.7.2 und 11.7.3.
[55] `http://www.pcre.org/`
[56] Über den `RegExp`-Konstruktor können einige Features von PCRE[57] in ECMAScript-Implementierungen emuliert werden, siehe auch `http://PointedEars.de/scripts/regexp.js`
[58] Eine Ausnahme ist das Variablenobjekt des globalen Ausführungskontextes. Dieses ist das *Globale Objekt*, welches mit `this` im globalen Kontext referenziert werden kann. Durch Zuweisung von `this` an eine globale Variable oder global verfügbare Eigenschaft kann das Globale Objekt auch im lokalen Kontext referenziert werden.
[59] ECMA INTERNATIONAL 2011a, Abschnitt 10.5.
[60] ECMA INTERNATIONAL 2000, Abschnitt 10.1.3.

flexibel (optionale Argumente), erschwert aber auch aufgrund der schwachen Typisierung das Schreiben von Funktionen, da diese Sonderfälle berücksichtigt werden müssen.

Konstruktoren In ECMAScript-basierten Programmiersprachen dienen Funktionen als Konstruktoren für Objekte. Mit dem `new`-Operator kann jede Funktion – und damit theoretisch jede Methode – auch als Konstruktor aufgerufen werden. Einige eingebaute Objekte (z. B. `RegExp`) haben oft zusätzlich die Eigenschaft, dass sie sich ohne `new` aufgerufen genauso verhalten, als wären sie als Konstruktor aufgerufen worden[61].

Objekt-Initialisierer `Object`-, `Array`- und `RegExp`-Instanzen können statt mit Konstruktor-Aufrufen auch mit Objekt-Initialisierern bzw. -Literalen erzeugt werden. Mit einem Initialisierer kann eine `Object`-Instanz zusätzlich mit Eigenschaften und Eigenschaftswerten initialisiert werden. `Object`- und `Array`-Initialisierer können auch ineinander verschachtelt werden. Dies ermöglicht die Erstellung komplexer Datenstrukturen in einem Ausdruck und hat zum JSON[62]-Austauschformat geführt[63].

2.2. ECMAScript-Implementierungen

Dieser Abschnitt gibt einen kurzen Überblick über die in dieser Arbeit untersuchten ECMAScript-Implementierungen. Für eine detailliertere Beschreibung wird auf die angegebene Fachliteratur und Webressourcen verwiesen.

[61] ECMA INTERNATIONAL 2011a, Abschnitte 15.4.1, 15.10.3 und 15.11.1.
[62] JavaScript Object Notation
[63] CROCKFORD 2008, ; siehe auch `http://json.org/`.

2.2.1. Netscape/Mozilla JavaScript (seit 1996)

2.2.1.1. Netscape JavaScript 1.2 bis 1.4

JavaScript 1.2 JavaScript 1.2 wurde mit Netscape Navigator 4.0 im August 1996 eingeführt[64]. Zu den neuen Features gegenüber Version 1.1 gehören unter anderem Unterstützung für reguläre Ausdrücke, signierte Skripte und die `switch`-Anweisung[65]. JavaScript 1.2 ist noch nicht vollständig ECMAScript-konform[66]. Dies äussert sich z. B. darin, dass der Wert einer Zuweisung in JavaScript 1.2 ein boolescher Erfolgswert ist und nicht der Wert der rechten Seite der Zuweisung.

JavaScript 1.3 JavaScript 1.3 wurde zusammen mit Netscape Navigator 4.06 als Bestandteil der Internet-Suite Netscape Communicator am 17. August 1998 released[67]. Dabei handelte es sich vor allem um Korrekturen an JavaScript 1.2, die JavaScript konform zu ECMAScript Edition 2 machen sollten[68].

JavaScript 1.4 JavaScript 1.4 liegt nur in einer serverseitigen Variante vor, die für den Netscape Enterprise Server (NES) entwickelt wurde. Diese Version ist von historischer Bedeutung, weil mit der serverseitigen Variante eine Trennung der Kernsprache vom Netscape-Browser-API notwendig wurde. Die Referenz zu JavaScript 1.4 trägt entsprechend den Titel "*Core* JavaScript Reference", wohingegen die Referenz zu JavaScript 1.3 noch den Titel "*Client-Side* JavaScript Reference" trägt[69].

[64] METZGER 2012.
[65] MOZILLA.ORG u. a. 2012b.
[66] NETSCAPE COMMUNICATIONS CORP. 1999.
[67] WILSON 2005.
[68] NETSCAPE COMMUNICATIONS CORP. 1999.
[69] NETSCAPE COMMUNICATIONS CORP. 1998a; NETSCAPE COMMUNICATIONS CORP. 1999.

2.2.1.2. Mozilla JavaScript 1.5 bis 1.8.5 (2002–2011)

1998 veröffentlichte Netscape den Quelltext von Netscape Communicator[70] und damit auch den Quelltext von SpiderMonkey, der originalen JavaScript-Implementierung in C/C++[71]. Dies führte über das Mozilla-Projekt zur Veröffentlichung von JavaScript 1.5[72].

JavaScript 1.5 JavaScript 1.5, released mit Mozilla 1.0 am 5. Juni 2002, ist die erste JavaScript-Version, die auf ECMAScript Edition 3 basiert. Zu den Neuerungen in dieser Version gehören daher Exceptions, Zahlenformatierung und vielseitigere reguläre Ausdrücke. Ausserdem proprietäre Features: Bedingte Funktionsdeklarationen ("function statements"), mehrere `catch`-Klauseln für gezielteres Abfangen von Fehlern und die Deklaration von Konstanten[73].

JavaScript 1.6 JavaScript 1.6, released mit Mozilla Firefox 1.5 am 29. November 2005[74], implementierte erstmals Features, die in ECMAScript for XML (E4X, ECMA-357)[75] spezifiziert sind, wie die `for-each`-Anweisung und `CDATA`-Literale[76]. Ausserdem wurden in dieser Version neue Array-Methoden eingeführt[77], die später in ECMAScript Edition 5 standardisiert wurden (vgl. Abschnitt 2.1.4).

JavaScript 1.7 JavaScript 1.7, released mit Firefox 2.0 am 24. Oktober 2006[78], stellt gegenüber Version 1.6 eine Reihe neuer, derzeit noch proprietärer Features[79] bereit, die ihren Ursprung in Python haben[80]: Generatoren und Iteratoren, Array Comprehension (vgl. List Comprehension in Python), blockweises Scoping mit `let` und das Destructuring

[70] NETSCAPE COMMUNICATIONS CORP. 1998b.
[71] MOZILLA.ORG u. a. 2011b.
[72] MOZILLA.ORG u. a. 2012a.
[73] MOZILLA.ORG u. a. 2012c.
[74] MOZILLA.ORG u. a. 2005.
[75] ECMA INTERNATIONAL 2005.
[76] `var x = <![CDATA[ foo bar ]]>;`
[77] MOZILLA.ORG u. a. 2011c.
[78] MOZILLA.ORG u. a. 2009.
[79] ORENDORFF 2008; EICH 2008a; DYER 2008.
[80] PYTHON SOFTWARE FOUNDATION 2012; EICH 2010b; EICH 2008a.

Assignment[81]. Einige dieser Features können nur benutzt werden, wenn JavaScript 1.7 explizit deklariert wird[82]. (Daher hat die für diese Arbeit erstellte Datenbank ein Feld `alt_type` in der Tabelle `testcase`, um genau dies zu ermöglichen; siehe Abschnitt 3.1.1.3 auf Seite 30.)

JavaScript 1.8 JavaScript 1.8 wurde als Bestandteil von Gecko 1.9 mit Firefox 3.0 am 24. Oktober 2006[83] released. Diese Version führt Expression Closures, Generator-Ausdrücke und weitere Array-Methoden ein[84], wobei letztere inzwischen in ECMAScript Edition 5 bzw. 5.1 standardisiert wurden[85]. Für Generator-Ausdrücke muss JavaScript 1.8 explizit deklariert werden.

JavaScript 1.8.1 JavaScript 1.8.1 wurde als Bestandteil von Gecko 1.9.1 mit Firefox 3.5, Thunderbird 3.0 und SeaMonkey 2.0 am 30. Juni 2009 released. Es stellt neu native Unterstützung für JSON bereit (im Dezember 2009 in ECMAScript Edition 5 standardisiert)[86].

JavaScript 1.8.5 JavaScript 1.8.5 wurde als Bestandteil von Gecko 2 mit Firefox 4, Thunderbird 3.3 und SeaMonkey 2.1 am 22. März 2011 released[87]. Es implementiert neue Features, die in ECMAScript Edition 5 standardisiert wurden, und für zukünftige Editionen vorgeschlagen wurden, wie Proxy-Objekte[88] für die Meta-Programmierung[89].

2.2.2. Microsoft JScript (seit 1996)

JScript ist, wie zuvor erwähnt, die ECMAScript-Implementierung von Microsoft. Parallel zu JavaScript wurde auch JScript weiterentwickelt, so dass sich JavaScript und

[81] Mit einem Destructuring Assignment ist z. B. der Austausch der Werte zweier Variablen `a` und `b` ohne Hilfsvariable möglich: `[a, b] = [b, a]` ist *semantisch* äquivalent zu `var tmp = a; a = b; b = tmp`.
[82] MOZILLA.ORG u. a. 2012d.
[83] MOZILLA.ORG u. a. 2008.
[84] MOZILLA.ORG u. a. 2012e.
[85] ECMA INTERNATIONAL 2009; ECMA INTERNATIONAL 2011a, jeweils Abschnitt 15.4.4.
[86] MOZILLA.ORG u. a. 2010.
[87] MOZILLA.ORG u. a. 2011a.
[88] VAN CUTSEM 2012; VAN CUTSEM 2010.
[89] MOZILLA.ORG u. a. 2010.

JScript heute stärker unterscheiden als früher. JScript ermöglicht z. B. mittels spezieller Schlüsselwörter (versions)bedingte Kompilierung (“Conditional Compilation”) und stellt mit `ActiveXObject` eine Schnittstelle zu Windows-spezifischen COM-Objekten bereit[90].

Mit JScript .NET gibt es ausserdem eine überwiegend serverseitig eingesetzte JScript-Version, die auf Netscapes Vorschlag für ECMAScript 4 basiert[91] und somit klassenbasierte Vererbung und starke Typisierung unterstützt.

2.2.3. Opera ECMAScript (seit 1997)

Opera ECMAScript ist die ECMAScript-Implementierung von Opera Software ASA für die Desktop-, Mini-[92] und Mobile-Versionen des Opera Browsers. Sie wird stellenweise von Opera auch als “JavaScript“ bezeichnet. Die erste Opera-Version mit Skript-Unterstützung war Opera 3.0, released am 1. Dezember 1997.[93] Die aktuelle ECMAScript-Implementierung in Opera 11.61 (released am 24. Januar 2012) ist gemäss Herstellerangaben konform zu ECMAScript Edition 3, und teilweise konform zu Edition 5. Dazu gehört auch native Unterstützung für JSON.[94].

2.2.4. KDE JavaScript (seit 2000)

KDE JavaScript (*KJS*) ist die ECMAScript-Implementierung, die für den Web-Browser Konqueror (Open Source)[95] entwickelt wurde und hauptsächlich dort eingesetzt wird. Die erste Version von KJS wurde am 23. Oktober 2000 mit KDE[96] 2.0 released[97]. KJS lieferte 2002 die Vorlage für Apple JavaScriptCore[98] (siehe Abschnitt 2.2.5).

[90] MICROSOFT CORP. 2011b.
[91] MICROSOFT CORP. 2011a.
[92] OPERA SOFTWARE ASA 2011.
[93] OPERA SOFTWARE ASA 2012b.
[94] OPERA SOFTWARE ASA 2012a.
[95] `http://konqueror.kde.org/`
[96] K Desktop Environment
[97] KDE E.V. 2000.
[98] STACHOWIAK 2002.

Seitdem wird KJS im Rahmen des KDE-Projekts weiterentwickelt, hat jedoch gegenüber JavaScriptCore und anderen Implementierungen an Bedeutung verloren. Die derzeit neueste in Debian GNU/Linux “sid” (instabil) verfügbare Version von Konqueror, 4.6.5, unterstützt zum Beispiel im Unterschied zu aktuellen Versionen anderer Implementierungen noch nicht alle Methoden, die in ECMAScript Edition 5 (Dezember 2009) eingeführt wurden (siehe Kapitel 4 auf Seite 59).

2.2.5. Apple JavaScriptCore (seit 2002)

JavaScriptCore ist die ECMAScript-Implementierung in WebKit[99]. Sie wird in Desktop- und Mobile-Versionen des Browsers Safari von Apple Inc. verwendet. JavaScriptCore wurde mit WebKit am 13. Juni 2002 released[100]. Am 2. Juni 2008 erfolgte im Rahmen des WebKit-Projekts ein Rewrite von JavaScriptCore namens *SquirrelFish*, das wie auch andere ECMAScript-Implementierungen einen Bytecode-Interpreter enthält und so gegenüber der Vorversion verbesserte Leistung zeigt[101]. Mit der Weiterentwicklung *SquirrelFish Extreme* wird Quelltext direkt zu Maschinencode kompiliert[102] (vgl. Google V8).

2.2.6. Google V8 (seit 2006)

V8 ist die ECMAScript-Implementierung von Google Inc. und wird vom Browser-Projekt Chromium (Open Source) sowie dem darauf basierenden Browser Google Chrome verwendet[103]. Ausserdem wird V8 von der Browserkomponente von Google Android[104] und für das serverseitige Framework node.js[105] verwendet[106]. Die erste Version von V8 wurde 2006 released[107].

[99] WEBKIT-PROJEKT 2011.
[100] STACHOWIAK 2002.
[101] GAREN 2008.
[102] STACHOWIAK 2008.
[103] GOOGLE INC. 2012.
[104] `http://code.google.com/android/`
[105] `http://nodejs.org/`
[106] GOOGLE INC. 2012.
[107] MORITA 2009.

V8 erreicht gegenüber den meisten anderen Implementierungen eine höhere Geschwindigkeit durch JIT-Kompilierung zu nativem Maschinencode (statt zu von einer VM zu interpretierenden Bytecode), Inline-Caching, aggressive Garbage Collection und weitere Optimierungen[108].

2.3. Kompatibilitätsmassnahmen

Wenn bekannt ist, dass ein Feature in einer Implementierung nicht unterstützt wird, so kann bestehender Quelltext in Abhängigkeit von der Art des Features kompatibel gemacht werden.

2.3.1. Syntaxelemente

Zu den *Syntaxelementen* gehören *Operatoren*, *Punktuatoren* und *Schlüsselwörter*. Operatoren können entweder Punktuatoren sein (z. B. `!==`) oder Schlüsselwörter (z. B. `instanceof`). Nicht alle Punktuatoren sind auch Operatoren; z. B. ist `(...)` der Gruppierungsoperator (*grouping operator*), die Eigenschaftszugriffsyntax `[...]` (*bracket property accessor*) hingegen nicht. Ebenso sind nicht alle Schlüsselwörter auch (in jedem Kontext) Operatoren (z. B. ist `in` in einem Ausdruck ein Operator; `for` hingegen nicht, und auch `in` in einer `for-in`-Anweisung nicht).

Neue Syntaxelemente sind nicht abwärtskompatibel, d. h. die Verwendung eines Syntaxelements in einer Implementierung, die dieses nicht unterstützt, führt zu einem Syntaxfehler.

Hier kann man sich die Eigenschaft der globalen `eval`-Funktion zunutze machen, beliebigen Quelltext innerhalb eines String-Literals als Programm im Programm auszuwerten. Ist ein Syntaxelement nicht implementiert, so ist dieses Teilprogramm in der nicht kompatiblen Implementierung syntaktisch fehlerhaft. Die Auswertung des Stringausdrucks ergibt dann entweder kein positives Ergebnis oder der Aufruf von `eval(...)` löst eine

[108] KIRSCH 2010.

`SyntaxError`-Exception aus. Ein fehlendes positives Ergebnis kann als negatives Ergebnis gewertet werden, d. h. die Implementierung unterstützt das Syntaxelement nicht. Eine ausgelöste `SyntaxError`-Exception kann behandelt werden, wenn die Implementierung das syntaktische Konstrukt `try... catch` oder die Host-Umgebung die proprietäre `window.onerror`-Eigenschaft unterstützt.

Aufgrund eines negativen Rückgabewerts, einer ausgelösten Exception oder dem Aufruf der Funktion, deren Referenz der `window.onerror`-Eigenschaft zuvor zugewiesen wurde, können dann alternative Programmabläufe und Funktionen (Fallbacks) genutzt werden. Der Nachteil dieses Ansatzes ist die reduzierte Laufzeiteffizienz, da das «Programm im Programm» bei jeder Verwendung des Features neu kompiliert werden muss.

Daher ist es nützlich, im Voraus zu wissen, welche Features zwischen welchen Implementierungen kompatibel sind. So kann das Risiko, dass das ECMAScript-basierte Programm mit einer nicht kompatiblen Implementierung konfrontiert wird, besser eingeschätzt werden. Die geringere Effizienz muss dann nur in den Fällen in Kauf genommen werden, wo der Kompatibilität Vorrang gegenüber der Effizienz eingeräumt wird.

2.3.2. Eingebaute Eigenschaften

Wie zuvor erwähnt, spezifizieren die Editionen von ECMAScript einige eingebaute Eigenschaften. Die Anzahl eingebauter Eigenschaften hat mit der Zeit stetig zugenommen. Implementierungen von früheren ECMAScript-Editionen unterstützen daher einige eingebaute Eigenschaften nicht, die in späteren Editionen spezifiziert wurden. Ebenso wurden die Implementierungen parallel zum Standardisierungsprozess weiterentwickelt und stellen neue Eigenschaften bereit. Besonders wenn es sich bei den Eigenschaften um Methoden handelt, ermöglichen diese neuen Methoden elegantere und effizientere Programmierung.

So ist es z. B. mit der Methode `Array.prototype.indexOf()`, die `Array`-Instanzen erben, möglich, mit nur einem Aufruf dieser Methode zu ermitteln, ob die gekapselte Array-Datenstruktur einen bestimmten Wert enthält. Wenn diese Methode nicht verfügbar ist, muss dafür eine `for`-Schleife programmiert werden. Diese ist tendenziell langsamer

als die native, vorkompilierte Implementierung, insbesondere wenn es sich um “sparse arrays”, d. h. Datenstrukturen, wo die Indizes der Elemente nicht fortlaufend vergeben sind, handelt. Denn in einer `for`-Schleife muss mittels `in`-Operator oder Aufruf von `Object.prototype.hasOwnProperty` erst überprüft werden, ob das Array ein Element mit einem entsprechenden Index hat.

Aufgrund der dynamischen Natur von ECMAScript-Implementierungen können einige native Methoden mit der jeweiligen ECMAScript-Implementierung selbst emuliert werden, z. B. in dem obiger Ansatz verallgemeinert wird und dem Objekt, welches `Array.prototype` referenziert, eine Methode namens `indexOf` hinzugefügt wird. Jedoch ist eine solche Implementierung aufgrund des zusätzlichen Methodenaufrufs in der Erstellung aufwändiger und in der Ausführung weniger effizient als würde man direkt eine `for`-Anweisung benutzen. Bei nativen Methoden fallen diese Unterschiede weniger ins Gewicht.

Auch weil einige native Methoden nicht mit geringem Aufwand in dieser Weise emuliert werden können, es daher oft sinnvoll, einen Fallback zu implementieren: Ergibt ein Laufzeittest (siehe Abschnitt 3.1.2.5), dass die native Methode anscheinend unterstützt wird, so kann der effizientere native Lösungsweg beschritten werden. Der Test selbst kostet jedoch auch wieder Laufzeit. Auch hier ist es daher nützlich zu wissen, ob es nötig und – je nach Kompatibilitätsgrad des Features – noch sinnvoll ist, die Fallunterscheidung an dieser Stelle zu machen und den Test durchzuführen.

3. Material und Methoden

Um ECMAScript-Implementierungen hinsichtlich ihrer Features miteinander zu vergleichen, wurden auf einzelne Features ausgerichtete Testfälle in verschiedenen Web-Browsern ausgeführt, die Ergebnisse gespeichert und ausgewertet.

Dieses Kapitel beschreibt im Detail die Hard- und Software, die bei den Tests zum Einsatz kamen (Material), sowie die Methoden, die hierbei angewendet wurden.

3.1. Material

Wie in Kapitel 1 dargelegt, bestimmt die Host-Umgebung die verwendete ECMAScript-Implementierung. Es wurden clientseitige Implementierungen miteinander verglichen. Um die Testergebnisse zu analysieren, ist jedoch ihre strukturierte persistente Speicherung erforderlich. Daher besteht die Testumgebung aus einer serverseitigen Komponente (*Backend*) und einer clientseitigen Komponente (*Frontend*).

3.1.1. Backend

Das Backend übernimmt die folgenden Aufgaben:

- Speicherung der Daten zu Implementierungen, Versionen, Host-Umgebungen, Features und Testfällen;
- Generierung des Frontends und
- Speicherung der vom Frontend übermittelten Testergebnisse für die spätere Auswertung.

Als Backends wurde eine LAMP[1]-Umgebung verwendet, die nachfolgend genauer beschrieben wird.

Die für diese Arbeit entwickelte Webapplikation implementiert den Datenbankzugriff und die Generierung des Frontends. Ihr serverseitiger Teil wird daher in Abschnitt 3.1.1.4 auf Seite 39 und ihr clientseitiger Teil in Abschnitt 3.1.2 auf Seite 42 beschrieben.

3.1.1.1. Betriebssystem

Als Betriebssystem für das Backend wurde Debian GNU/Linux[2] 5.0.9 mit Linux 2.6.18-238.12.1.el5.028stab091.1ent verwendet.

3.1.1.2. Webserver

Als Webserver wurde der Apache HTTP Server[3], Version 2.2.16-6+squeeze6 (Debian-Paket `apache2-mpm-prefork`) verwendet.

3.1.1.3. Datenbank

Für den Vergleich der Implementierungen wurde mit phpMyAdmin[4] eine MySQL[5]-Datenbank erstellt und verwendet. Das Datenbankdesign erfolgte hauptsächlich mit MySQL Workbench[6].

Die folgende Software wurde verwendet:

- MySQL-Server 5.1.49-3 (Debian-Paket `mysql-server-5.1`)
- MySQL-Client 5.1.49-3 (Debian-Paket `mysql-client-5.1`)
- phpMyAdmin 3.3.7-7 (Debian-Paket `phpmyadmin`)

[1] Linux, Apache, MySQL, PHP
[2] `http://www.debian.org/`
[3] `http://httpd.apache.org/`
[4] `http://www.phpmyadmin.net/`
[5] `http://www.mysql.com/`
[6] `http://www.mysql.com/products/workbench/`

- MySQL Workbench SE 5.2.37 für Ubuntu Linux 10.04 (x86, 32-bit), DEB

Aufgaben Die Datenbank erfüllt folgende Aufgaben:

- Registrierung von Host-Umgebungen und damit assoziierten Versionen von ECMAScript-Implementierungen;
- Speicherung von Informationen zu Features und den dazugehörigen Testfällen;
- *Persistente* Speicherung der Testergebnisse und somit
- Bereitstellung der Daten für die Auswertung.

Datenbankstruktur Die Datenbank besteht aus folgenden Tabellen (Details in Abbildung 3.1 auf Seite 32 und Tabelle 3.1 auf Seite 31):

- `implementation` zur Erfassung der Implementierungen;
- `version` zur Erfassung der Versionen von Implementierungen;
- `environment` zur Identifikation der Host-Umgebungen, in denen getestet wurde;
- `feature` für die zu testenden Features;
- `testcase` für die Testfälle zu einem Feature und
- `result` für die Testergebnisse.

Tabelle	Speicher-Engine	Kollation
`implementation`	InnoDB	`utf8_general_ci`
`version`	InnoDB	`utf8_general_ci`
`environment`	InnoDB	`utf8_general_ci`
`feature`	InnoDB	`utf8_general_ci`
`testcase`	InnoDB	`utf8_general_ci`
`result`	InnoDB	`utf8_general_ci`

Tabelle 3.1.: Datenbanktabellen

Speicher-Engine Als Speicher-Engine (storage engine) für die Tabellen wurde InnoDB gewählt, weil diese (z. B. im Unterschied zur MyISAM-Engine[7]) Fremdschlüssel nativ

[7] ORACLE CORP. AND/OR ITS AFFILIATES 2012a.

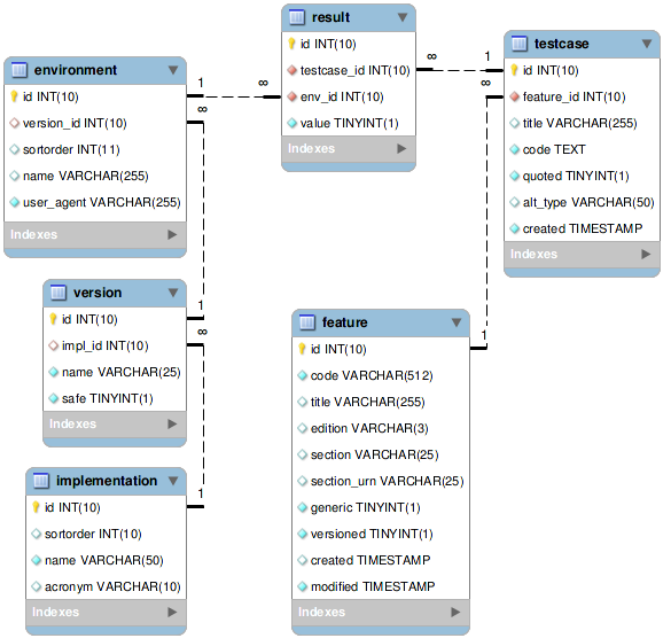

Abbildung 3.1.: Datenbankstruktur

unterstützt[8].

Fremdschlüssel werden benötigt, um die referentielle Integrität der Daten sicherzustellen (vgl. Abbildung 3.1 auf Seite 32): So muss z. B. ein Testergebnis einem Testfall und der Host-Umgebung, in welcher der Test durchgeführt wurde, zugeordnet werden. Eine Host-Umgebung wiederum muss einer Version einer Implementierung zugeordnet sein, um dem Testergebnis in der Auswertung eine Bedeutung zuordnen zu können. Testergebnisse müssen invalidiert und sollten automatisch aus der Datenbank entfernt werden, wenn ein Testfall nachträglich geändert oder gelöscht wird (siehe Tabelle 3.11 auf Seite 39).

Kollation Als Kollation für Tabellen und Textfelder wurde `utf8_general_ci` gewählt. Somit können Zeichen aus dem Unicode-Zeichensatz ohne Maskierung, in der Codierung UTF-8, gespeichert werden, und Textdaten werden ohne Berücksichtigung von Gross- und Kleinschreibung (case-insensitive) sortiert.

Nachfolgend werden die Tabellen der Datenbank im Detail beschrieben. Die MySQL-Anweisungen zu ihrer Erstellung sind im Anhang A.1.1 zu finden.

Tabelle `implementation` Die Tabelle `implementation` speichert die zu testenden Implementierungen. Sie hat folgende Felder (Details siehe Tabelle 3.2):

- `id` für den Primärschlüssel;
- `sortorder` für die Spaltenreihenfolge im Frontend (siehe Abschnitt 3.1.2 auf Seite 42);
- `name` für die Bezeichnung (Spaltenüberschrift im Frontend) und
- `acronym` für eine mögliche Kurzbezeichnung (wird, wenn angegeben, anstelle der Bezeichnung im Frontend angezeigt).

Tabelle `version` Die Tabelle `version` speichert die Versionen der Implementierungen. Sie hat folgende Felder (Details siehe Tabelle 3.3):

- `id` für den Primärschlüssel;

[8] ORACLE CORP. AND/OR ITS AFFILIATES 2012b.

Feld	Typ	NULL?	Key	Standard	Extra
id	INT(10) UNSIGNED	Nein	PRIMARY		AUTO_INC.[a]
sortorder	INT(10) UNSIGNED	Nein		0	
name	VARCHAR(50)	Nein	UNIQUE		
acronym	VARCHAR(10)	Ja		NULL	

[a] AUTO_INCREMENT: Wert wird bei neuen Datensätzen automatisch erhöht

Tabelle 3.2.: Struktur der Tabelle implementation

- impl_id für die assoziierte Implementierung (Fremdschlüssel);
- name für die Versionsnummer und
- safe für die Angabe, ob ein Feature auch dann als sicher gilt, wenn die Version dieses Feature gemäss Testfällen nicht oder unvollständig unterstützt. Veraltete Versionen von Implementierungen, deren Host-Umgebungen ihr *end-of-life*[9] erreicht haben, können so für die Auswertung markiert werden.

Feld	Typ	NULL?	Key/Index	Standard	Extra
id	INT(10) UNSIGNED	Nein	PRIMARY		AUTO_INC.
impl_id	INT(10) UNSIGNED	Ja	MULTI-COL.[b]	NULL	
name	VARCHAR(25)	Ja	MULTI-COL.		
safe	TINYINT(1) UNSIGNED	Nein		0	

[b] MULTI-COLUMN: mehrspaltiger Index

Tabelle 3.3.: Struktur der Tabelle version

Die Tabelle version steht über Fremdschlüssel in Beziehung mit der Tabelle implementation (Tabelle 3.4).

[9] Ende des Lebens- bzw. Wartungszyklus einer Software. Die Software wird ab diesem Zeitpunkt vom Hersteller nicht mehr unterstützt, und der Hersteller empfiehlt in der Regel den Wechsel auf eine neue Version.

Feld	Foreign-Key Constraint
impl_id	implementation.id ON DELETE CASCADE ON UPDATE CASCADE

Tabelle 3.4.: Beziehungen der Tabelle version

Tabelle environment Die Tabelle environment speichert Informationen zu den Host-Umgebungen, in denen getestet wurde. Sie hat folgende Felder:

- id für den Primärschlüssel;
- version_id für die zugeordnete Version einer Implementierung;
- sortorder für die Reihenfolge der Darstellung im Frontend (siehe Abschnitt 3.1.2 auf Seite 42);
- name für die Bezeichnung der Umgebung, i. d. R. zusammengesetzt aus Produktname und Version;
- user_agent für den Wert des; UserAgent-HTTP-Headerfelds beim Test mit der Host-Umgebung.

Feld	Typ	NULL?	Key	Std.	Extra
id	INT(10) UNSIGNED	Nein	PRIMARY		AUTO_INC.
version_id	INT(10) UNSIGNED	Ja	INDEX	NULL	
sortorder	INT(10) UNSIGNED	Nein		0	
name	VARCHAR(255)	Ja	UNIQUE	NULL	
user_agent	VARCHAR(255)	Nein	UNIQUE		

Tabelle 3.5.: Struktur der Tabelle environment

Das Feld name hat einen *eindeutigen* Index (UNIQUE), damit es bei der nachträglichen manuellen Benennung der Umgebung keine Duplikate geben kann.

Das Feld user_agent hat einen eindeutigen Index, damit dieselbe Host-Umgebung nur einmal in der Datenbank ausgeführt ist. Es wird in dieser Arbeit davon ausgegangen, dass der Wert des User-Agent-Headerfelds eine Host-Umgebung eindeutig identifiziert. Abweichungen wurden bei den Tests nicht festgestellt.

Das Feld `version_id` hat einen *einfachen* Index (`INDEX`), weil zwar die Möglichkeit besteht, dass verschiedene Umgebungen dieselbe Implementierung unterstützen, das Feld aber als Fremdschlüssel einen Index haben muss[10] (siehe Tabelle 3.6).

Die Tabelle `environment` steht über Fremdschlüssel in Beziehung mit der Tabelle `version`:

Feld	Foreign-Key Constraint
`version_id`	`version.id ON DELETE NO ACTION ON UPDATE NO ACTION`

Tabelle 3.6.: Beziehungen der Tabelle `environment`

Tabelle `feature` Die Tabelle `feature` speichert die zu testenden Features. Sie hat folgende Felder:

- `id` für den Primärschlüssel;
- `code` für die Bezeichnung oder ein Quelltextbeispiel des Features;
- `title` für eine Kurzbeschreibung des Features;
- `edition` für die Edition der ECMAScript-Spezifikation, in der ein Feature erstmals beschrieben ist;
- `section` für den Abschnitt der Edition, in der ein Feature erstmals beschrieben ist;
- `section_urn` für den URN des Abschnitts in der Edition[11];
- `generic` gibt an, ob eine Objekteigenschaft absichtlich generisch spezifiziert ist, d. h. vom ursprünglichen Objekt auf andere Objekte übertragen werden kann;
- `versioned` gibt an, ob ein Feature die Deklaration der Version einer Implementierung erfordert, damit es genutzt werden kann;
- `created` für Datum und Zeit, an denen das Feature der Datenbank hinzugefügt wurde und

[10] ORACLE CORP. AND/OR ITS AFFILIATES 2012c.

[11] Das Feld `section_urn` soll später im Frontend für die Linksetzung verwendet werden. Da eine ECMAScript-Edition immer wieder referenziert werden muss, hat sich in der Vergangenheit die Verwendung eines eigenen URN-Schemas als sinnvoll erwiesen. Siehe auch [LAHN 2011].

- `modified` für Datum und Zeit, an denen das Feature zuletzt geändert wurde.

Feld	Typ	NULL?	Key	Standard	Extra
id	INT(10) UNSIGNED	Nein	PRI.		A._I.
code	VARCHAR(512)	Nein			
title	VARCHAR(255)	Ja		NULL	
edition	VARCHAR(3)	Ja		NULL	
section	VARCHAR(25)	Ja		NULL	
section_urn	VARCHAR(25)	Ja		NULL	
generic	TINYINT(1) UNSIGNED	Nein		0	
versioned	TINYINT(1) UNSIGNED	Nein		0	
created	TIMESTAMP	Ja		NULL	
modified	TIMESTAMP	Nein		CUR._TIMEST.[c]	

[c] CURRENT_TIMESTAMP: aktueller Zeitstempel

Tabelle 3.7.: Struktur der Tabelle `feature`

Tabelle `testcase` Die Tabelle `testcase` enthält die Testfälle für jedes Feature. Sie hat folgende Felder:

- `id` für den Primärschlüssel;
- `feature_id` für die Zuordnung eines Testfalls zu einem Feature (Fremdschlüssel);
- `title` für den Titel des Testfalls;
- `code` für den Quelltext des Testfalls;
- `quoted` gibt an, ob der Quelltext für den Test in doppelte Anführungszeichen eingeschlossen werden soll und entsprechend maskiert werden muss. Dieser Ansatz erleichtert später die Eingabe der Testfälle (siehe Abschnitt 3.1.2 auf Seite 42) und reduziert den für diese benötigten Speicherplatz.
- `alt_type` gibt optional einen alternativen MIME-Typ für einen Testfall an. Somit können auch Features getestet werden, die in einer Implementierung nur bei Angabe dieses MIME-Typs verfügbar sind; und
- `created` für Datum und Zeit, an denen der Testfall erstellt wurde.

Feld	Typ	NULL?	Key	Standard	Extra
id	INT(10) UNSIGNED	Nein	PRI.		AUTO_INC.
feature_id	INT(10) UNSIGNED	Nein	INDEX		
title	VARCHAR(255)	Ja		NULL	
code	TEXT	Nein			
quoted	TINYINT(1) UNSIGNED	Ja		0	
alt_type	VARCHAR(50)	Ja		NULL	
created	TIMESTAMP	Nein		CUR._TIMEST.	

Tabelle 3.8.: Struktur der Tabelle testcase

Die Tabelle testcase steht über Fremdschlüssel in Beziehung mit der Tabelle feature (siehe Tabelle 3.9).

Feld	Foreign-Key Constraint
feature_id	feature.id ON DELETE CASCADE ON UPDATE CASCADE

Tabelle 3.9.: Beziehungen der Tabelle testcase

Tabelle result Die Tabelle result speichert die Testergebnisse für die spätere Auswertung. Sie hat folgende Felder (Details siehe Tabelle 3.10):

- id für den Primärschlüssel;
- testcase_id für die Angabe des verwendeten Testfalls (Fremdschlüssel);
- env_id für die Angabe der verwendeten Host-Umgebung (Fremdschlüssel) und
- value für das Ergebnis des Testfalls.

Der eindeutige mehrspaltige Index über die Felder testcase_id und env_id verhindert, dass mehrere Testergebnisse für den selben Testfall und die selbe Host-Umgebung gespeichert werden können.

Die Tabelle result steht über Fremdschlüssel in Beziehung mit den Tabellen testcase und environment:

Feld	Typ	NULL?	Key/Index	Std.	Extra
id	INT(10) UNSIGNED	Nein	PRIMARY		AUTO_INC.
testcase_id	INT(10) UNSIGNED	Nein	MULTI-C.		
env_id	INT(10) UNSIGNED	Nein	MULTI-C.		
value	TINYINT(1)	Nein			

Tabelle 3.10.: Struktur der Tabelle result

Feld	Foreign-Key Constraint
testcase_id	testcase.id ON DELETE CASCADE ON UPDATE CASCADE
env_id	environment.id ON DELETE CASCADE ON UPDATE CASCADE

Tabelle 3.11.: Beziehungen der Tabelle result

Der Constraint ON DELETE CASCADE sorgt dafür, dass ein Testergebnis automatisch gelöscht wird, wenn der dazugehörige Testfall oder die dazugehörige Host-Umgebung aus der Datenbank gelöscht wird. Somit werden ungültige Testergebnisse ohne zusätzliche Programmierung vermieden.

3.1.1.4. Webapplikation

Für diese Arbeit wurde mit Eclipse PHP Development Tools (PDT)[12] eine MVC[13]-basierte, datenbankunterstützte[14] Webapplikation entwickelt, die auf einer früheren Arbeit[15] aufbaut. Die Webapplikation ermöglicht die effiziente Verwaltung der für Tests und Auswertung benötigten Ausgangsdaten (Zuordnungen und Testfälle), sowie die effiziente Verarbeitung und Auswertung der Testergebnisse.

[12] http://www.eclipse.org/projects/project.php?id=tools.pdt
[13] Model-View-Controller
[14] siehe Abschnitt 3.1.1.3
[15] LAHN 2011.

Laufzeitumgebung Als Programmiersprache für die Applikation wurde PHP 5.3.3-7+squeeze7 als Apache-2-Modul (Debian-Paket `libapache2-mod-php5`) verwendet.

Es kamen folgende wesentlichen PHP-Erweiterungen zum Einsatz:

- `pdo.so` – PHP Data Objects (*PDO*)[16]
- `pdo_mysql.so` – MySQL-Unterstützung für PDO[17]
- `suhosin.so` – *Suhosin*-Patch[18]
- `xdebug.so` – *Xdebug* (Debugger und Profiler)[19]

PDO wurde verwendet, weil seine Klassenbibliothek einen einfachen und universellen Zugriff auf verschiedene relationale DBMS[20], darunter MySQL, ermöglicht. Auf PDO basierende Applikationen können daher mit wenig Aufwand an andere DBMS angebunden werden. Ausserdem unterstützt es Prepared Statements (Applikationssicherheit) und bietet ein einfaches Interface für Transaktionen (Erhaltung der Datenintegrität bei zusammengehörigen Datenbankoperationen).

Die Verwendung des *Suhosin*-Patches wird aus Sicherheitsgründen empfohlen; Details hierzu sind auf der Suhosin-Website zu finden. Es wird jedoch davon ausgegangen, dass die Applikation auch ohne diese Erweiterung voll funktionsfähig ist.

Für die Funktionsfähigkeit der fertigen Applikation ist die *Xdebug*-Erweiterung ebenfalls nicht erforderlich; sie erwies sich jedoch während der Entwicklung als sehr nützlich. Xdebug stellt neben einem Remote-Debugging-fähigen Debugger (der mit Eclipse PDT genutzt werden kann) unter anderem auch eine verbesserte Ausgabe für die PHP-Funktion `var_dump()` bereit. Dies erleichtert die Fehlersuche besonders in komplexeren Applikationen.

Software-Architektur Die Webapplikation basiert auf einem für diese Arbeit entwickelten Framework, welches das MVC[21]-Entwicklungsmuster implementiert. Das Framework ist an

[16] `http://php.net/manual/en/book.pdo.php`
[17] `http://php.net/manual/en/ref.pdo-mysql.php`
[18] `http://www.hardened-php.net/suhosin/`
[19] `http://xdebug.org/`
[20] Database Management System
[21] Model-View-Controller

das Zend Framework[22] angelehnt, ohne dessen gesamten Funktionsumfang abzubilden (da dieser vorerst nicht benötigt wurde). Es wird hier kurz beschrieben. Der relevante Quelltext befindet sich in Anhang A.1.2.1.

Das vom Servermodul aufgerufene *Start-Script* (hier: `index-db.php`) instantiiert die *Applikation* (Klasse `Application`), definiert die Standard-Datenbankverbindung (*Datenbank-Objekt*) und startet die Applikation (`Application::run()`).

Die *Applikation* verarbeitet in ihrer `run`-Methode die eingehenden HTTP-Requests. Sie instantiiert eine *Controller*-Klasse (Subklasse von `Controller`) entsprechend des URI-Parameters `controller`. Fehlt dieser Parameter, wird ein *Default-Controller* (Vorgabe: `IndexController`) instantiiert.

Der *Controller* instantiiert eine entsprechende *View*-Klasse (Vorgabe: `View`) für die spätere Verwendung. Der Controller wertet ausserdem den URI-Parameter `action` aus, um seine dazugehörige *Action-Methode* (Bezeichner-Suffix: `Action`) aufzurufen. Fehlt der `action`-Parameter, wird eine *Default-Action-Methode* (Vorgabe: `indexAction`) aufgerufen.

In seiner *Action-Methode* benutzt der Controller eine oder mehrere Methoden von *Mapper*-Klassen (Subklassen von `Mapper`), um die der Aktion und den übrigen URI-Parametern (z. B. `id`) entsprechenden Datenbankzugriffe durchzuführen, und die Ergebnisse dieser Zugriffe speziellen *Template*-Variablen des zuvor instantiierten Views zuzuweisen (`assign`-Methode).

Der *Mapper* übernimmt den Datenbankzugriff (CRUD[23]). Dabei bedient er sich *Model*-Instanzen (Instanzen von Subklassen von `AbstractModel`) und eines *Tabellenobjekts* (Instanz einer Subklasse von `Table`), wobei es für jede verwendete Datenbanktabelle jeweils eine Model-Klasse und eine `Table`-Subklasse gibt.

Die Eigenschaften mit Sichtbarkeit `protected` einer Model-Instanz bilden für spätere Verwendung einen Datensatz objektorientiert ab. Das *Tabellenobjekt* stellt die Verbindung

[22] `http://framework.zend.com/`
[23] Create, Read, Update, Delete

zu einem *Datenbank-Objekt* (Instanz einer Subklasse von `Database`) her, stellt Metainformation zur Datenbanktabelle bereit (z. B. den Tabellennamen) und delegiert an das Datenbank-Objekt tabellenspezifische Datenbankoperationen.

Der Konstruktor einer *Model*-Klasse setzt die Eigenschaften der Model-Instanz über ihre Setter und Getter. Dabei findet das Mapping von Namen von Datenbank-Feldern zu Eigenschaften der Instanz über die Methode `Model::map()` statt, die von Subklassen überschrieben werden kann. Im Setter schliesslich findet die Bereichsüberprüfung und Typkonvertierung statt, während ein Getter den öffentlichen Zugriff auf `protected`-Eigenschaften ermöglicht.

(Alternativ zu diesem Model-Mapper-Ansatz wurde auch mit der Implementierung eines ORM[24]-Ansatzes begonnen. Dieser wurde jedoch vorerst nicht weiterentwickelt, weil er sich für diese Applikation als zu unflexibel erwies. So muss zum Beispiel für die effiziente Auswertung eine Auswahlabfrage über sechs miteinander verknüpfte Tabellen ausgeführt werden, die sich mit einem ORM nicht ohne weiteres abbilden lässt.)

Der *View* übernimmt die Darstellung, d. h. das Generieren des Frontends (siehe Abschnitt 3.1.2) unter Verwendung von Templates. Die Templates enthalten HTML-Quelltext sowie minimale PHP-Programmlogik unter Verwendung der zuvor zugewiesenen Template-Variablen.

3.1.2. Frontend

Die Tests werden im Frontend durchgeführt. Kurze Programme, die geeignet sind, das Laufzeitverhalten von Features von ECMAScript-Implementierungen zu testen (*Testfälle*), werden clientseitig ausgeführt. Die clientseitige Auswertung der Testergebnisse führt clientseitig zur Generierung von HTML-Steuerelementen, deren Name und Wert an das Backend per Formular übermittelt wird.

[24] Object-Relational Mapper

3.1.2.1. Darstellung

Das Frontend besteht im Wesentlichen aus einer mehrzeiligen und mehrspaltigen HTML-Tabelle (Abbildung 3.2). Diese listet in den Zeilen die zu testenden Features und in den Spalten die Testergebnisse für dieses Feature auf.

Ausserdem gibt es eine Spalte, welche den Abschnitt der ECMAScript-Edition angibt, in der das Feature erstmals spezifiziert wurde. Somit können einfacher Vergleiche zwischen ECMAScript-Editionen und Versionen von Implementierungen angestellt werden, wie es in der Einleitung dargelegt wurde.

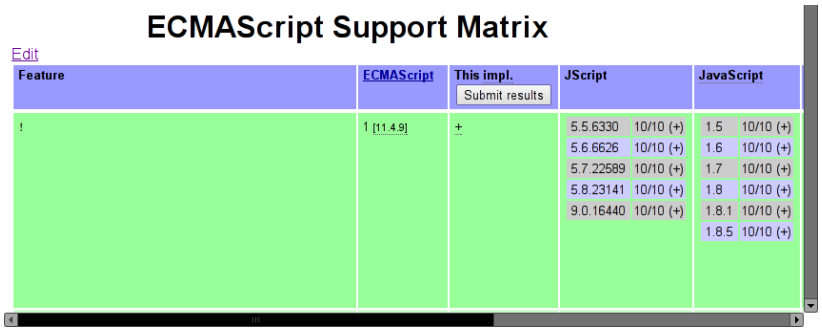

Abbildung 3.2.: Normalmodus

3.1.2.2. Benutzerschnittstelle

Der Benutzer hat zunächst die Möglichkeit, zwischen Normalmodus und Bearbeitungsmodus (Abbildung 3.3) zu wechseln. Im initialen Normalmodus werden alle Elemente der Benutzeroberfläche ausgeblendet, die für die Bearbeitung benötigt werden. Sowohl im Normal- als auch im Bearbeitungsmodus hat der Benutzer jedoch die Möglichkeit, Testergebnisse an das Backend zu übermitteln (Schaltfläche “Submit results”, siehe Abschnitt 3.2).

Im Bearbeitungsmodus können Features hinzugefügt, bearbeitet und gelöscht, sowie Implementierungen bearbeitet und hinzugefügt werden.

Ob sich der Benutzer im Normal- oder im Bearbeitungsmodus befindet, wird in einer Session-Variable gespeichert. (Ein Authentifizierungsmechanismus für die Bearbeitung wurde noch nicht implementiert. Der URL der Webapplikation wurde während der Erstellung dieser Arbeit nicht veröffentlicht und eine mögliche Brute-Force-Indizierung der Webapplikation durch gebräuchliche Suchmaschinen mittels `robots.txt` unterbunden[25].)

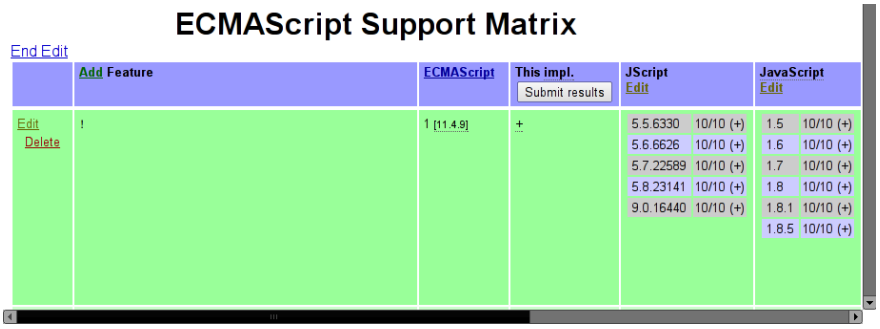

Abbildung 3.3.: Bearbeitungsmodus

3.1.2.3. Features

Zuerst wurden die zu testenden Features definiert. Dabei konnte die Information zu 257 Features aus den als PHP-Array vorliegenden Daten einer früheren Arbeit[26] mittels der in Abschnitt 3.1.1.4 beschriebenen Applikation in die Datenbank (Abschnitt 3.1.1.3) importiert werden. Siehe hierzu `IndexController::importAction()` in Anhang A.1.2.2 (inzwischen sind die betreffenden Zeilen auskommentiert, um versehentliches Überschreiben der geänderten Importdaten zu verhindern).

Diese Importdaten wurden unter Verwendung der Webapplikation (siehe Abbildungen 3.4 und 3.5) aus den folgenden Quellen angepasst und ergänzt:

- ECMAScript Language Specification, Edition 1[27]

[25] KOSTER 1995, 1997, 2007.
[26] LAHN 2011.
[27] ECMA INTERNATIONAL 1997.

- ECMAScript Language Specification, Edition 2[28]
- ECMAScript Language Specification, Edition 3 Final[29]
- ECMAScript Language Specification, Edition 5.1[30]
- MDN[31]: Core JavaScript Reference[32]
- MSDN[33] Library: JScript Language Reference[34]

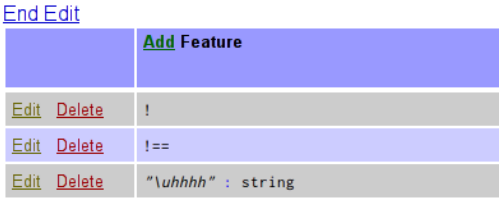

Abbildung 3.4.: Ausgangsdarstellung für das Hinzufügen von Features

Für die Eingabe der Features und Testfälle wurde Google Chrome 18.0.1025.45 beta (Debian-Paket `google-chrome-beta`) auf Debian GNU/Linux 6.0.4 mit dem benutzerkonfigurierten Kernel Linux 2.6.39.3-pe verwendet.

3.1.2.4. Testfälle

Die Testfälle wurden für jedes Feature vom Backend aus der Datenbank ausgelesen und im Frontend innerhalb von `SCRIPT`-Elementen generiert. In den `SCRIPT`-Elementen wurde die im nachfolgenden Abschnitt beschriebene Testfunktion, mit dem Quelltext des Testfalls als Argument, ausgeführt.

[28] ECMA INTERNATIONAL 1998.
[29] ECMA INTERNATIONAL 2000.
[30] ECMA INTERNATIONAL 2011a.
[31] Mozilla Developer Network
[32] MOZILLA.ORG u. a. 2011d.
[33] Microsoft Developer Network
[34] MICROSOFT CORP. 2011b.

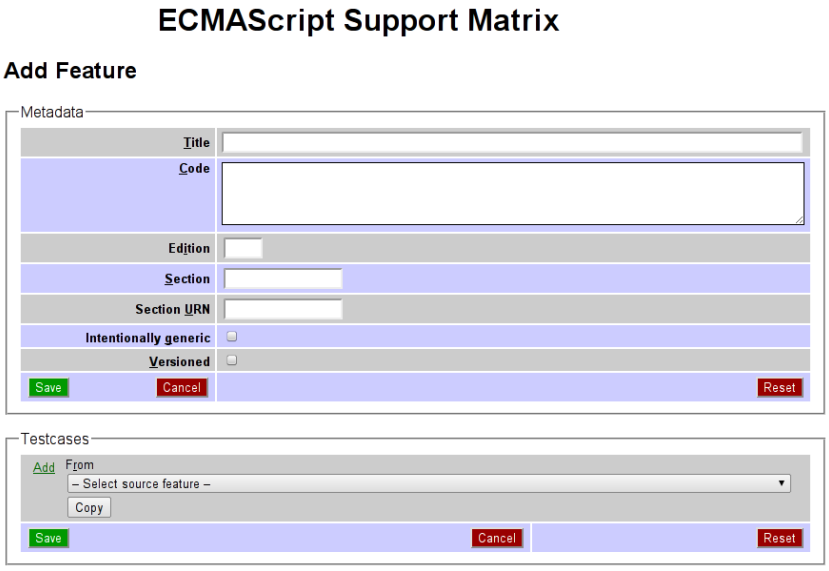

Abbildung 3.5.: Dialog “Add Feature”“ (Feature hinzufügen)

In Kooperation mit dem Backend konnten die Testfälle für jedes Feature einzeln im Frontend bearbeitet werden: es konnten für jedes Feature Testfälle hinzugefügt, geändert und gelöscht werden (Abbildung 3.6).

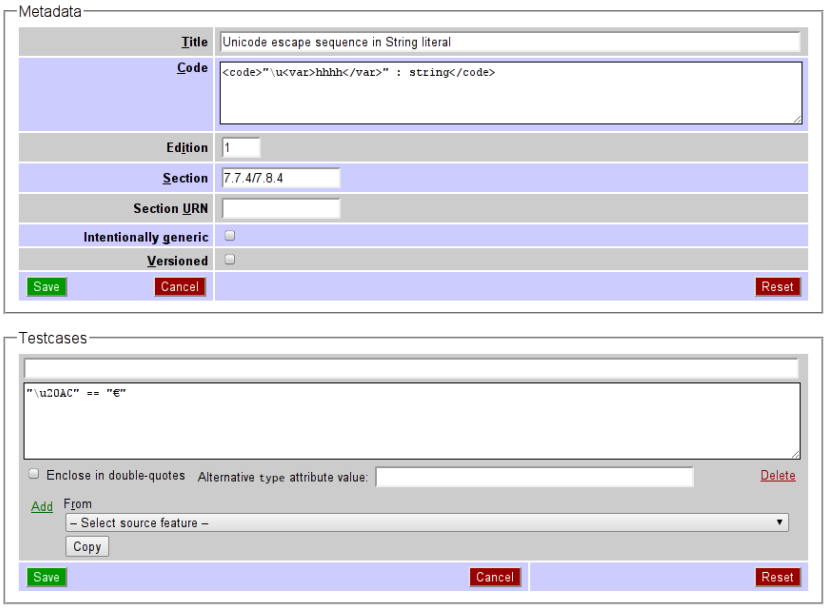

Abbildung 3.6.: Dialog “Edit Feature” (Feature/Testfälle bearbeiten)

Wie bereits in Abschnitt 3.1.1.3 erwähnt, kann für jeden Testfall ausgewählt werden, ob der Quelltext in Anführungszeichen gesetzt wird und welcher alternativer MIME[35]-Typ gegebenenfalls verwendet werden soll.

Für die Dialoge “Add Feature” und “Edit Feature” wird dasselbe Template verwendet (`layouts/feature/edit.phtml`). Die Action-Methode der `FeatureController`-Instanz bestimmt die Darstellung über den Wert der Template-Variable `feature`. Ist keine Feature-ID angegeben (es gilt also `$this->feature->id === null`), so wird der Dialog für das Hinzufügen konfiguriert, andernfalls für das Bearbeiten.

[35] Multipurpose Internet Mail Extensions

Die Änderungen werden mit der “Save”-Schaltfläche übernommen, und mit der “Cancel”=Schaltfläche verworfen (`history.back()`). Mit der “Reset”-Schaltfläche werden die Originalwerte wiederhergestellt, ohne den Bearbeitungsmodus zu verlassen.

Die “Save”-Schaltfläche in der Box “Testcases” löscht vor dem Speichern der neuen Testfälle automatisch die alten Testfälle für das Feature und somit auch die bisherigen Testergebnisse für das Feature (vgl. Abschnitt 3.1.1). Die “Save”-Schaltfläche in der Box “Metadata” hingegen speichert nur die Metadaten für ein Feature, ohne die Testfälle und Testergebnisse zu verändern.

Bei den Testfällen werden folgende Arten unterschieden:

- Syntaktisch
  - Einfacher Test
  - Ausgewerteter Test
- Funktionell
  - Syntaxtest
  - Test auf Existenz
  - Test auf Funktionalität

Die einzelnen Arten werden nachfolgend genauer beschrieben.

Syntaktisch

Einfacher Test Testfälle, die syntaktisch ausschliesslich Features aus den ersten JavaScript-Versionen oder ECMAScript-Editionen verwenden, können direkt ausgeführt werden. In diesem Fall bleibt das Kontrollkästchen “Enclose in double-quotes” (in doppelte Anführungszeichen einschliessen) leer und der Quelltext wird wie eingegeben der Testfunktion als Argument übergeben.

Ausgewerteter Test Ist hingegen aus der Praxis bekannt, dass die für einen Test verwendete Syntax wahrscheinlich nicht universell nutzbar ist (z. B. weil zuvor der Test auch mit den ältesten Versionen von Implementierungen fehlschlug), so wird das Kontrollkästchen “Enclose in double-quotes” markiert. Der Quelltext wird somit weiterhin unverändert in der Datenbank gespeichert, aber das Feld `quoted` wird für den betreffenden Testfall auf `1` gesetzt.

Dies sorgt in der Feature-Liste dafür, dass der für diesen Testfall gespeicherte Quelltext in doppelte Anführungszeichen eingeschlossen wird. Alle übrigen doppelten Anführungszeichen im Quelltext und alle Escape-Sequenzen werden dann maskiert, so dass der Quelltext als *`String`-Literal* an die Testfunktion übergeben wird (siehe Abschnitt 3.1.2.5).

Funktionell

Syntaxtest Bei einem Syntaxtest wird überprüft, ob der Testfall in der Implementierung als Programm kompiliert, d. h. nicht zu einem Syntaxfehler führt. Der Test wird so gestaltet, dass das Ergebnis des Programms `true` bzw. ein zu `true` konvertierbarer Wert ist, wenn der Quelltext ausgeführt werden kann. Kann das Programm nicht bis zu diesem Punkt ausgeführt werden, wird angenommen, dass die Implementierung das getestete syntaktische Konstrukt nicht unterstützt.

Test auf Existenz Bei einem Existenztest wird überprüft, ob eine eingebaute Eigenschaft aufgrund ihres Typs oder Wertes als vorhanden und *vermutlich funktionell* betrachtet werden kann.

Um implizite Typkonvertierung zu vermeiden, wird zunächst der `typeof`-Operator verwendet, der bereits seit JavaScript 1.1 und JScript 1.0 implementiert ist (siehe Abschnitt 2.1.4). Er wird daher als unterstützt vorausgesetzt. Die Praxis hat gezeigt, dass eine `typeof`-Operation für native Methoden den Wert `"function"` liefert und für nicht aufrufbare native Eigenschaften weder den Wert `"function"` noch den Wert `"undefined"`.

Host-Objekte werden in dieser Arbeit nicht berücksichtigt. Somit können nicht-implementierte Eigenschaften (`typeof`-Ergebnis `"undefined"`), sowie implementierte Methoden (Ergebnis `"function"`) und implementierte nicht-aufrufbare Eigenschaften (anderes Ergebnis) voneinander unterschieden werden.

Die Fallunterscheidung ist in der Methode `jsx.object.getFeature()` für nicht-aufrufbare Eigenschaften, sowie in der Methode `jsx.object.isNativeMethod()` für native aufrufbare Eigenschaften als Spezialfall der Methode `jsx.object.isMethod()` implementiert (siehe Anhang A.2.1). Die Verwendung von Funktionsdeklarationen statt Funktionsausdrücken soll es ermöglichen, damit auch Implementierungen zu testen, die keine Funktionsausdrücke unterstützen.

Diese Testfallvariante wird *alleinstehend* nur in Ausnahmefällen gewählt, etwa wenn eine Methode getestet werden muss, aber die Argumente, welche sie erwartet, nicht eindeutig spezifiziert sind. Das Feature gilt als unterstützt, wenn der Existenztest ein positives Ergebnis liefert.

Test auf Funktionalität Wenn aufgrund eines positiven Existenztests eine Eigenschaft als existent angenommen werden kann, dann kann im nächsten Schritt der Versuch unternommen werden, ihre *Funktionalität* zu testen.

Eine potentielle nicht-aufrufbare Eigenschaft ist funktionell, wenn sie das spezifizierte oder offiziell dokumentierte Werteverhalten zeigt. Eine potentielle Methode ist funktionell, wenn sie aufrufbar ist und sie für evtl. angegebene Argumente den erwarteten Rückgabewert liefert.

Bei einem Funktionstest werden die Algorithmen in der ECMAScript-Spezifikation oder in der Dokumentation des Features beim Hersteller berücksichtigt. Soweit der Aufwand vertretbar ist, werden alle Fälle, die sich aus den Algorithmen ergeben, getestet (z. B. Auslösen einer Exception, Verhalten bei fehlenden Argumenten oder Argumenten ausserhalb eines Wertebereichs). Für ein Feature werden dann *mehrere* voneinander *unabhängige* Testfälle für jeden dieser algorithmischen Spezialfälle und für die algorithmischen Normalfälle erstellt.

3.1.2.5. Testfunktion

Es wurde die in Anhang A.2.2 angegebene Testfunktion verwendet. Sie ermöglicht es, variablen Quelltext auszuführen, ohne dass dies im normalen Fehlerfall die Ausführung weiterer Testfälle verhindert.

Hierzu wird der Typ des Arguments überprüft. Handelt es sich um einen `String`-Wert, so wird dieser letztlich der `eval`-Funktion zur Ausführung als Programm übergeben. Handelt es sich bei dem Argument nicht um einen String-Wert, so wird überprüft, ob das Argument boolesch `true` oder `false` ist oder in einen primitiven `boolean`-Wert konvertiert werden kann.

`String`-Werte werden der Testfunktion nur übergeben, wenn es sich bei dem Wert *im Prinzip* um den Quelltext eines syntaktisch korrekten Programms handelt. Das heisst, das Programm sollte in einer konformen Implementierung mindestens einer der ECMAScript-Editionen fehlerfrei ausgeführt werden können. Somit werden falsch-negative Testergebnisse vermieden.

3.1.2.6. Testergebnis

Das Ergebnis der Testfunktion führt zur Generierung von versteckten HTML-Steuerelementen (`INPUT`-Elemente mit Attributspezifikation `type="hidden"`) innerhalb eines HTML-Formulars (`FORM`-Element). Der Wert des Steuerelements gibt das Ergebnis in der getesteten Implementierung für den jeweiligen Testfall an (`value="0"` oder `value=""`:

«nicht erfolgreich»; `value="1"`: «erfolgreich»; Details siehe Abschnitt 3.2). Durch Absenden des Formulars werden die clientseitig ermittelten Testergebnisse an das Backend zur Speicherung und weiteren Auswertung übermittelt.

3.1.3. ECMAScript-Implementierungen

Da die Host-Umgebung die verwendete Implementierung definiert, müssen die Features zur Laufzeit getestet werden. Als getestete Implementierung wird diejenige angenommen, die vom Hersteller der Host-Umgebung, in der getestet wurde, als solche dokumentiert wurde. Dies betrifft sowohl die Script-Engine als auch die angebene Version.

Demzufolge stehen die folgenden ECMAScript-Implementierungen für den Test zur Verfügung (vgl. Abschnitt 2.2 auf Seite 20):

- Netscape/Mozilla JavaScript 1.0 bis 1.8.5
- Microsoft JScript 5.5.6330 bis 9.0.16440
- Google V8 3.6.6.19 bis 3.8.9.5
- Apple JavaScriptCore 531.22.7 bis 7534.52.7
- KDE JavaScript 4.6.5

Genauere Angaben sind in Anhang A.2.3 zu finden.

3.2. Methoden

Es wurden ECMAScript-Implementierungen anhand ihrer Features auf Konformität zur Edition 5.1 der ECMAScript Language Specification getestet. Ausserdem wurde die Kompatibilität von Features, die für bestimmte ECMAScript-Implementierungen charakteristisch sind, aber nicht in ECMAScript explizit spezifiziert sind, überprüft.

3.2.1. Konformitätstests

Als Basis für die Konformitätstests wurde ECMAScript Edition 5.1 gewählt. Denn bei dieser Edition kann von einer stabilen Spezifikation ausgegangen werden (es handelt sich um eine Revision von Edition 5), und sie hat als derzeit neueste Edition die höchste Praxisrelevanz.

Andere Editionen von ECMAScript wurden nicht explizit berücksichtigt. Gleichwohl wurden implizit auch durch frühere Editionen spezifizierte Features getestet, sofern Edition 5.1 diese in gleicher Weise spezifiziert oder frühere Editionen lediglich präzisiert.

3.2.2. Einschränkungen bei Laufzeittests

Bei Laufzeittests müssen die Features einer Implementierung mit den Mitteln der Implementierung selbst überprüft werden. Verlässliche Testergebnisse als Grundlage für einen Vergleich sind jedoch nur möglich, wenn die Testmethode in allen Host-Umgebungen einheitlich ist.

Dies hat zur Folge, dass für den Test eines Features zusätzlich benötigte Features als unterstützt vorausgesetzt werden müssen. Ausserdem müssen die Testfälle gegeneinander isoliert werden, damit nicht ein Testfall den darauf folgenden beeinflusst.

3.2.2.1. Vorausgesetzte Features

Wird ein vorausgesetztes Feature nicht unterstützt, so kann der Testfall, der es benötigt, nicht ausgeführt werden. Die Nichtunterstützung eines vorausgesetzten Features führt also zu einem falsch-negativen Testergebnis für das andere, zu testende Feature (siehe Abschnitt 3.1.2.4). Daher wurden die vorausgesetzten Features auf ein notwendiges Minimum beschränkt, wobei insgesamt eine maximale Abwärtskompatibilität angestrebt wurde.

Die folgenden Features wurden als unterstützt vorausgesetzt:

- Literale[36]
  - `null`
  - Boolesche Literale (`true`, `false`)
  - Numerische Literale in einfacher Dezimaldarstellung (z. B. `42`)
  - String-Literale (z. B. `"foo"`)
- Ausdrücke
  - Eigenschaftszugriff (*x*`.`*y* und *x*`[`*y*`]`)[37]
  - Aufruf (*f*`(...)`)[38]
  - Operatoren[39]
    - Gruppierungsoperator (`(...)`)
    - `delete` *x*
    - `void` *x*
    - `typeof` *x*
    - Unäres Minus (`-`*x*)
    - Logisches NICHT (`!`*x*)
    - Relationale Operatoren (<, >, <= und >=)
    - Typkonvertierender Vergleich (*x* `==` *y*)
    - Negierter typkonvertierender Vergleich (*x* `!=` *y*)
    - Logisches UND (*x* `&&` *y*)
    - Logisches ODER (*x* `||` *y*)
    - Bedingungsoperator (*x* `?` *y* `:` *z*)
    - Einfache Zuweisung (*x* `=` *y*)
- Anweisungen[40]
  - Variablendeklaration mit optionalem Initialisierungsteil (`var` *x* `=` *y*)
  - Fallunterscheidung (`if (`*x*`)` *y*`;` `else` *z*)
  - `for (`*x* `in` *y*`)` *z*
  - Rücksprung aus Funktion (`return` *x*)
- Funktionsdeklaration (`function` *x* `(...) {...}`)[41]

[36] ECMA INTERNATIONAL 1997, Abschnitt 7.7.
[37] Ebd., Abschnitt 11.2.1.
[38] Ebd., Abschnitt 11.2.3.
[39] Ebd., Abschnitte 11.1.4, 11.4, 11.8, 11.9, 11.12 und 11.13.
[40] Ebd., Abschnitt 12.
[41] Ebd., Abschnitt 13.

- Eingebaute Objekte und Eigenschaften
  - Globales Objekt[42], referenziert mit `this` im globalen Kontext[43]
    - `NaN`
    - `eval(`*x*`)`
    - `isNaN(`*number*`)`
  - `Object`-Konstruktor[44]
  - `Array`-Konstruktor[45]
  - `String.prototype.indexOf(`*searchString*`)`[46]

Ferner wurden im Einzelfall einige Features als unterstützt vorausgesetzt, die

a) in der selben ECMAScript-Edition wie das getestete Feature;
b) in einer früheren ECMAScript-Edition als das getestete Feature oder
c) in der selben oder einer früheren Version der selben Implementierung wie das getestete Feature

erstmalig spezifiziert bzw. dokumentiert sind. Dies sind im einzelnen:

- Operatoren
  - Unäres Plus (`+`*x*)[47] für Tests der strengen Vergleichsoperatoren `===` und `!==`[48]
  - *x* `instanceof` *y*[49] für `try...catch` mit mehreren `catch`-Klauseln[50]
- Anweisungen
  - Block Scoping (`let`) und Generatoren (`yield`) für Array Comprehension[51]
- Eingebaute Objekte und Eigenschaften

[42] ECMA INTERNATIONAL 1997, Abschnitt 15.1.
[43] Ebd., Abschnitt 10.4.1.1.
[44] Ebd., Abschnitt 15.2.2.
[45] Ebd., Abschnitt 15.4.2.
[46] Ebd., Abschnitt 15.5.4.6.
[47] Ebd., Abschnitt 11.4.6.
[48] ECMA INTERNATIONAL 2000, Abschnitte 11.9.4 und 11.9.5.
[49] Ebd., Abschnitt 11.8.6.
[50] MOZILLA.ORG u. a. 2011e.
[51] MOZILLA.ORG u. a. 2012d.

- `Object.defineProperty(...)`[52] für `Object.getOwnPropertyDescriptor(`*`object`*`)`[53]
- `Function.prototype.call(`*`thisArg`*`)`[54] für den Test der internen `[[Class]]`-Eigenschaft des `arguments`-Objekts[55]
- `String.prototype.match(`*`regexp`*`)`[56] und `RegExp.prototype.test(`*`string`*`)`[57] für Features mit Bezug auf `RegExp`-Instanzen und -Literale[58]
- `Error`-Objekte[59] für das Testen von Exceptions, die von ECMAScript-Algorithmen ausgelöst werden können.

3.2.2.2. Isolation der Testfälle

Die Praxis hat gezeigt, dass sich in den für Tests zur Verfügung stehenden Host-Umgebungen (siehe Abschnitt 3.1.3) Skripts in HTML-`SCRIPT`-Elementen einen globalen Namensraum teilen; dass sie jedoch bezüglich Fehlern, die zu einem Programmabbruch führen, gegeneinander isoliert sind.

Ein Syntaxfehler in einem Skript in einem `SCRIPT`-Element hat somit keinen Einfluss auf die Ausführung eines Skripts in einem anderen `SCRIPT`-Element. Daher können die Testfälle gegeneinander isoliert werden, indem jeder Testfall in einem eigenen `SCRIPT`-Element ausgeführt wird:

```
<script type="text/javascript">
  /* Testfall 1 */
</script>
<script type="text/javascript">
  /* Testfall 2 */
</script>
```

[52] ECMA INTERNATIONAL 2011a, Abschnitt 15.2.3.6.
[53] Ebd., Abschnitt 15.2.3.3.
[54] Ebd., Abschnitt 15.3.4.4.
[55] Ebd., Abschnitt 10.6.
[56] ECMA INTERNATIONAL 2000, Abschnitt 15.5.4.1.
[57] Ebd., Abschnitt 15.10.6.3.
[58] Ebd., Abschnitt 15.10.
[59] Ebd., Abschnitt 15.11.

Ausserdem muss verhindert werden, dass ein erfolgreich ausgeführter Testfall den nachfolgenden Testfall beeinflusst. Dies wird durch die folgenden Massnahmen erreicht:

- Bei einfachen Tests findet in dem zu testenden Ausdruck keine Zuweisung zu einer Variablen statt.
- Ausgewertete Tests werden über die `eval`-Methode des Globalen Objekts als eigenes Programm ausgeführt. Dabei werden in diesem Programm alle neuen Bezeichner deklariert. Referenzen auf Bezeichner, sofern sie nicht global verfügbare Objekte betreffen, sind somit lokal zu diesem Ausführungskontext.
- Auf eine ansonsten in den verwendeten Skript-Bibliotheken (`object.js`) implementierte Erweiterung eingebauter Objekte wird über eine für diesen Zweck eingeführte Option explizit verzichtet (`jsx.options.emulate == false`). Ebenso werden von den Testfällen die eingebauten ECMAScript-Objekte (“Standard Built-in ECMAScript Objects”[60]) nicht verändert.

3.2.3. Untersuchungsgegenstand

Die getesteten Features von ECMAScript-Implementierungen lassen sich in eine von zwei Kategorien einordnen. Entweder handelt es sich um Syntaxelemente oder um eingebaute Eigenschaften (Built-ins).

3.2.3.1. Syntaxelemente

Die Implementierung von Syntaxelementen kann nicht direkt getestet werden, da die direkte Verwendung von nicht implementierten Syntaxelementen einen *Syntaxfehler* darstellt, der weitere zuverlässige Tests verhindert.

Hier machen wir uns die Eigenschaft der globalen `eval`-Funktion zunutze, beliebigen Quelltext innerhalb eines String-Literals als Programm im Programm auszuwerten. Ist ein Syntaxelement nicht implementiert, so ist dieses Teilprogramm in der getesteten Implementierung syntaktisch fehlerhaft. Die Auswertung des Ausdrucks ergibt dann entweder kein

[60] ECMA INTERNATIONAL 2011a, Abschnitt 15.

positives Ergebnis oder der Aufruf von `eval(...)` löst eine `SyntaxError`-Exception aus.

Ein fehlendes positives Ergebnis wird als negatives Ergebnis gewertet, d. h. die Implementierung unterstützt das Syntaxelement nicht. Eine ausgelöste `SyntaxError`-Exception kann behandelt werden, wenn die Implementierung das syntaktische Konstrukt `try...catch` oder die Host-Umgebung die proprietäre `window.onerror`-Eigenschaft unterstützt.

3.2.3.2. Eingebaute Eigenschaften

ECMAScript-Implementierungen verfügen über einige eingebaute Eigenschaften. Die Eigenschaften können Methoden sein, d. h. aufrufbare Objekte referenzieren. Diese Eigenschaften sind entweder in ECMAScript spezifiziert oder anderweitig dokumentiert (Sprachreferenz, Tutorials, Artikel etc.). Sofern möglich, wird die Quelle für jede Eigenschaft angegeben.

Nicht spezifizierte und nicht dokumentierte Eigenschaften werden nicht getestet. Die Detektion bislang unbekannter Eigenschaften ist nicht Bestandteil dieser Arbeit.

4. Resultate

Testfälle für ausgewählte in ECMAScript spezifizierte und proprietäre Features von ECMAScript-Implementierungen wurden in Web-Browsern ausgeführt. Ein Feature gilt in der von einem Web-Browser verwendeten Version einer Implementierung als unterstützt, wenn *alle* für es erstellten Testfälle ein positives Ergebnis liefern.

Eine Version einer Implementierung gilt als *sicher*, wenn es unwahrscheinlich erscheint, dass sie noch produktiv eingesetzt wird. Siehe hierzu Abschnitt 4.2.2. Ein Feature gilt als *sicher*, wenn es in *allen nicht-sicheren* Versionen von Implementierungen unterstützt wird.

4.1. Getestete Features

Es wurden die in Tabelle A.2 in Anhang A.2.4 angegebenen Features getestet. Dabei wurden die in Tabelle A.3 in Anhang A.2.5 angegebenen Testfälle ausgeführt.

4.2. Implementierungen

Nicht alle von den verfügbaren Implementierungen konnten getestet werden, weil das verwendete minimale Testframework noch immer zu hohe syntaktische Anforderungen an die Implementierungen stellte.

4.2.1. Getestete Implementierungen

Von den zur Verfügung stehenden Implementierungen (siehe Anhang A.2.3) konnten nur die folgenden Versionen – aufgrund von Syntaxfehlern – *nicht* getestet werden:

Netscape/Mozilla JavaScript			
Version	Browser	Browserversion	Betriebssystem
1.0	Netscape Navigator	2.02	Windows (Wine)
1.1		3.04	
1.2		4.04	
1.3	Netscape Communicator	4.8	

Tabelle 4.1.: Nicht getestete Implementierungen

In Netscape Communicator 4.8 konnten alle Testfälle trotz Laufzeitfehlern durchgeführt werden, sie lieferten jedoch ebenfalls Scriptfehler aufgrund der verwendeten Syntax und somit ausschliesslich falsch-negative Resultate. So ergab z. B. der automatische Test des `Object`-Initialisierers, dass jener nicht unterstützt nicht werde, obwohl ein manueller Test ergab, dass dieser Ausdruck von Netscape Communicator 4.8 durchaus unterstützt wird.

Auch in anderen Browsern, darunter Internet Explorer 7.0 und 9.0 sowie Konqueror 4.6.5 wurden stellenweise Laufzeitfehler während des Tests angezeigt. Diese waren jedoch nur lokal auf einen Testfall bezogen (etwa der Test JavaScript-proprietärer Syntax, die in JScript und KJS mit hoher Wahrscheinlichkeit zu einem Syntaxfehler führen *musste*) und beeinträchtigten den Test insgesamt somit nicht.

4.2.2. Sichere Versionen

Nach Ansicht dieses Autors können Versionen von Implementierungen als sicher betrachtet werden, wenn der Marktanteil der sie verwendenden Host-Umgebungen auf unter 2%

gefallen ist, oder wenn er unter 10% gefallen ist und die Host-Umgebung ihr End-of-Life erreicht hat.

Basierend auf statistischen Erhebungen von Net Applications (Anhang A.3.1) wurden die folgenden getesteten Versionen von ECMAScript-Implementierungen als sicher eingestuft; d. h. die Bedeutung der entsprechenden Host-Umgebungen ist nach Ansicht dieses Autors zu gering, als dass die Testergebnisse für die korrespondierenden Versionen von Implementierungen in Erwägung gezogen werden müssten:

- Netscape/Mozilla JavaScript bis einschliesslich Version 1.8.5 (alle getesteten Versionen)
- Microsoft JScript bis einschliesslich Version 5.6.6626
- Apple JavaScriptCore bis einschliesslich Version 7533.18.5
- Opera ECMAScript bis einschliesslich Version 11.61 (alle getesteten Versionen)

4.3. Sichere Features

Basierend auf den statistischen Daten wurden die entsprechenden Versionen in der Datenbank als «sicher» markiert. Die Auswertung der Darstellung der so aktualisierten Kompatibilitätsmatrix ergab somit, dass die in Tabelle A.5 (Anhang A.3.2) aufgelisteten Features derzeit als nicht sicher für die ungetestete Verwendung in Web-Browsern betrachtet werden müssen.

4.4. Web-Applikation und detaillierte Resultate

Die für diese Arbeit erstellte Web-Applikation und detaillierte Resultate sind unter `http://PointedEars.de/es-matrix/index-db` verfügbar.

Die Zeilen der basierend auf diesen Resultaten als sicher erachteten Features sind mit grüner Farbe hinterlegt; die entsprechenden `TR`-Elemente haben einen `class`-Attributwert,

der das Wort “safe” enthält. Die Darstellung kann über ein Benutzer-Stylesheet individuell angepasst werden.

5. Diskussion

Im Unterschied zu einer häufig vorkommenden, vereinfachenden Diskussion von ECMAScript-Implementierungen unter einem Sammelbegriff zeigt die nähere Beschäftigung mit diesen Programmiersprachen, insbesondere im professionellen Umfeld, beispielhaft die Unterschiede zwischen ihnen auf.

Das Ziel dieser Arbeit war daher eine systematische Untersuchung dieser Unterschiede, um konkrete Aussagen zur Kompatibilität ausgewählter Features machen zu können. Je genauere Kenntnisse über die Features dieser Programmiersprachen vorliegen, umso effizienter und robuster werden darin geschriebene Programme.

5.1. Ansatz

Basierend auf einer früheren Arbeit wurden ausgewählte, in ECMAScript spezifizierte und auch proprietäre Features in mehreren Web-Browsern untersucht. Spezifizierte Features wurden auf die Standardkonformität ihrer Implementierungen hin untersucht, und alle ausgewählten Features wurden auf ihre Kompatibilität zwischen verschiedenen Implementierungen hin untersucht.

Durch Ausnutzung von besonderen Eigenschaften von ECMAScript an sich und in seinen von Web-Browsern verwendeten Implementierungen wurden unter Zuhilfenahme einer zu diesem Zweck erstellten datenbankgestützten Web-Applikation für jedes Feature gegeneinander isolierte Testfälle erstellt, welche dessen Existenz, und wenn möglich auch Funktionalität, zur Laufzeit testen konnten. Die Testergebnisse wurden mit der Web-Applikation für die spätere Analyse gespeichert.

5.2. Auswertung

Aufgrund des Verbreitungsgrades der Host-Umgebungen, in denen getestet wurde, und der Unterstützung durch ihren Hersteller, konnten Rückschlüsse darauf gezogen werden, welche der Testergebnisse zum aktuellen Zeitpunkt relevant sind. Somit war es möglich, Features als für den ungetesteten Gebrauch «sicher» zu definieren. Dieses neue Wissen hat das Potential, die Effizienz der diese Features verwendenden Programme zu steigern, da auf Kompatibilitätsmassnahmen verzichtet werden kann.

Auf der anderen Seite zeigen die Resultate, dass ECMAScript-Implementierungen trotz einer zunehmenden Annäherung an den Standard und an andere Implementierungen bis auf weiteres voneinander unterschieden werden müssen, um Softwarequalität sicherzustellen. Es sind daher entweder gezielte Kompatibilitätsmassnahmen nötig, um die derzeit noch nicht sicheren Features in allen untersuchten Implementierungen nutzen zu können, oder auf die Nutzung dieser Features sollte in produktiven Umgebungen vorläufig zugunsten kompatiblerer Konstrukte verzichtet werden. Dies betrifft insbesondere Features, die einige Implementierungen schon seit einigen Jahren unterstützen, die jedoch erst in ECMAScript Edition 5 spezifiziert wurden.

5.2.1. Offene Punkte

Es konnten nicht alle Features und nicht alle gebräuchlichen Implementierungen getestet werden. Die Einbeziehung von weiteren Features, Implementierungen und Versionen von Implementierungen kann das Bild, welches hier vom aktuellen Zustand der Standardkonformität und Kompatibilität gewonnen wurde, weiter verschieben. Daher sollten keine schwerwiegenden Designentscheidungen nur aufgrund der Ergebnisse dieser Arbeit getroffen werden. Vielmehr sollten die Ergebnisse als Empfehlung vor allem an Web-Applikationsentwickler betrachtet werden, sich näher mit der hier diskutierten Problematik auseinanderzusetzen.

Einige Features konnten nur eingeschränkt getestet werden. So konnte die Funktionalität von Methoden des `Date`-Prototypobjekts nur eingeschränkt getestet werden, weil einige

Rückgabewerte zeitzonenabhängig und implementierungsabhängig spezifiziert sind[1].

Obwohl die Testfälle sehr sorgfältig erstellt wurden, ist es möglich, dass sie Fehler aufweisen, mithin also zu einem falsch-positiven oder falsch-negativen Ergebnis führen. Sie sollten einem gründlichen Peer Review unterzogen und gegebenenfalls korrigiert werden, um die Realität möglichst exakt abzubilden. Es sollten auch wo nötig weitere Funktionstests ergänzt werden. Dies betrifft insbesondere die Methode `JSON.stringify()`, um der zunehmenden Bedeutung von JSON gerecht zu werden.

5.3. Ausblick

ECMAScript und seine Implementierungen werden ständig weiterentwickelt und gewinnen stetig an Bedeutung nicht nur im Web-Umfeld. Neue Features werden den Sprachen hinzugefügt, alte fehlerträchtige oder mit Sicherheitsrisiken behaftete entfernt. Insbesondere für Mozilla JavaScript kann die Übernahme von neuen Features aus anderen Programmiersprachen, wie z. B. Python, beobachtet werden.

Der Vergleich von ECMAScript-Implementierungen kann daher mit dieser Arbeit nicht abgeschlossen sein. Vielfältige Anwendungszwecke dieser Programmiersprachen erfordern die Einbeziehung von weiteren clientseitigen und von browserfremden Implementierungen.

Schliesslich sollten vergleichende Untersuchungen auch über die Programmiersprachen hinaus auf damit verwendbare APIs wie dem DOM ausgedehnt werden. Diese Arbeit kann dazu als Grundlage dienen.

[1] ECMA INTERNATIONAL 2011a, Abschnitt 15.9.5.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

ANDERSON, CHRISTOPHER, PAOLA GIANNINI und SOPHIA DROSSOPOULOU (2005). «Towards Type Inference for JavaScript». In: *ECOOP 2005 - Object-Oriented Programming*. Hrsg. von ANDREW BLACK. Bd. 3586. Lecture Notes in Computer Science. 10.1007/11531142_19. Springer Berlin / Heidelberg, S. 733–733. ISBN: 978-3-540-27992-1. URL: http://dx.doi.org/10.1007/11531142_19.

ANDREESSEN, MARC (1998). «Innovators of the Net: Brendan Eich and JavaScript». In: *Netscape Columns. TechVision*. URL: http://web.archive.org/web/20080208124612/http://wp.netscape.com/comprod/columns/techvision/innovators_be.html (besucht am 13. 02. 2012).

BÜHLER, DIRK und STEFAN W. HAMERICH (2004). «Towards Embedding VoiceXML Applications Through Compilation». In: *Proceedings of the Berliner XML-Tage 2004*. Berlin, Germany. URL: http://www.hamerich.de/stefan/pub/vxml-xml04.pdf.

BYOUS, JOHN (1998). *Java Technology: An Early History*. URL: http://gcc.upb.de/www/WI/WI2/wi2_lit.nsf/7544f3043ee53927c12573e70058bbb6/abf8d70f07c12eb3c1256de900638899/$FILE/Java%20Technology%20-%20An%20early%20history.pdf (besucht am 13. 02. 2012).

CHUGH, RAVI u. a. (2009). «Staged information flow for javascript». In: *Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation*. PLDI '09. Dublin, Ireland: ACM, S. 50–62. ISBN: 978-1-60558-392-1. DOI: 10.1145/1542476.1542483. URL: http://doi.acm.org/10.1145/1542476.1542483.

CROCKFORD, DOUGLAS (2008). *JavaScript: The Good Parts*. O'Reilly. ISBN: 9780596517748. URL: http://books.google.de/books?id=PXa2bby0oQ0C.

DYER, JEFF (2008). «Destructuring assignment and binding forms». In: *ECMAScript Harmony Wiki*. URL: http://wiki.ecmascript.org/doku.php?id=proposals:destructuring_assignment (besucht am 15. 02. 2012).

ECMA INTERNATIONAL (1997). *Standard ECMA-262 – ECMAScript: A general-purpose, cross-platform programming language*. URL: http://www.mozilla.org/js/language/E262.pdf (besucht am 15. 12. 2003).

– (1998). *Standard ECMA-262. ECMAScript Language Specification*. 2nd Edition. URL: http://www.mozilla.org/js/language/E262-2.pdf (besucht am 22. 07. 2002).

– (2000). *ECMAScript Language Specification*. Edition 3 Final. URL: http://www.mozilla.org/js/language/E262-3.pdf (besucht am 22. 07. 2002).

– (2005). *Standard ECMA-357. ECMAScript for XML (E4X) Specification*. 2nd edition. URL: http://www.ecma-international.org/publications/standards/Ecma-357.htm (besucht am 15. 02. 2009).

– (2007). *Proposed ECMAScript 4th Edition – Language Overview*. URL: http://www.ecmascript.org/es4/spec/overview.pdf (besucht am 15. 02. 2012).

– (2009). *Standard ECMA-262. ECMAScript Language Specification*. 5th Edition. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf (besucht am 06. 12. 2009).

– (2011a). *Standard ECMA-262. ECMAScript Language Specification*. 5.1 Edition. URL: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (besucht am 27. 08. 2011).

– (2011b). *Standard ECMA-262. ECMAScript Language Specification*. URL: http://www.ecma-international.org/publications/standards/Ecma-262.htm (besucht am 19. 12. 2011).

– (2012). *ecmascript test262*. URL: http://test262.ecmascript.org/ (besucht am 12. 02. 2012).

EICH, BRENDAN (2008a). «Array Comprehensions». In: *ECMAScript Harmony Wiki*. URL: http://wiki.ecmascript.org/doku.php?id=harmony:array_comprehensions (besucht am 15. 02. 2012).

– (2008b). «ECMAScript Harmony». In: *es-discuss – Discussion of ECMAScript*. URL: https://mail.mozilla.org/pipermail/es-discuss/2008-August/003400.html (besucht am 15. 02. 2012).

EICH, BRENDAN (2010a). «Bytecode Standard In Browsers». In: *A Minute With Brendan*. URL: http://www.aminutewithbrendan.com/pages/20101122 (besucht am 13. 02. 2012).

– (2010b). «Paren-Free. The tl;dr version». In: *Blog*. URL: http://brendaneich.com/2010/11/paren-free/ (besucht am 15. 02. 2012).

FLANAGAN, DAVID (2011). *JavaScript: The Definitive Guide*. Definitive Guides. O'Reilly Media. ISBN: 9780596805524. URL: http://books.google.de/books?id=4RChxt67lvwC.

GAREN, GEOFFREY (2008). «Announcing SquirrelFish». In: *Surfin' Safari*. URL: http://www.webkit.org/blog/189/announcing-squirrelfish/ (besucht am 14. 02. 2012).

GOODMAN, DANNY, MICHAEL MORRISON und BRENDAN EICH (2007). *JavaScript Bible*. Sixth Edition. John Wiley & Sons. ISBN: 9780470146231. URL: http://books.google.de/books?id=W3eytUUwhEIC.

GOOGLE INC. (2012). *Google Chrome Releases*. URL: http://googlechromereleases.blogspot.com/ (besucht am 26. 02. 2011).

GROSSMAN, GARY und EMMY HUANG (2006). «ActionScript 3.0 overview». In: *Adobe Developer Connection. ActionScript Technology Center*. URL: http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html (besucht am 15. 02. 2012).

GUARNIERI, SALVATORE und BENJAMIN LIVSHITS (2009). «GATEKEEPER: mostly static enforcement of security and reliability policies for javascript code». In: *Proceedings of the 18th conference on USENIX security symposium*. SSYM'09. Montreal, Canada: USENIX Association, S. 151–168. URL: http://dl.acm.org/citation.cfm?id=1855768.1855778.

HAMILTON, NAOMI (2008). «The A-Z of Programming Languages: JavaScript». In: *Computerworld*. The A-Z of Programming Languages. URL: http://www.computerworld.com.au/article/255293/a-z_programming_languages_javascript/ (besucht am 31. 12. 2011).

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (2008). *754-2008 IEEE Standard for Floating-Point Arithmetic*. URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4610935 (besucht am 18. 02. 2012).

JAWORSKI, JAMES (1999). *Mastering JavaScript and JScript*. 1st. Alameda, CA, USA: SYBEX Inc. ISBN: 0782124925.

JENSEN, SIMON, ANDERS MØLLER und PETER THIEMANN (2009). «Type Analysis for JavaScript». In: *Static Analysis*. Hrsg. von JENS PALSBERG und ZHENDONG SU. Bd. 5673. Lecture Notes in Computer Science. 10.1007/978-3-642-03237-0_17. Springer Berlin / Heidelberg, S. 238–255. ISBN: 978-3-642-03236-3. URL: http://dx.doi.org/10.1007/978-3-642-03237-0_17.

KDE E.V. (2000). *KDE 2.0 Release Announcement*. URL: http://kde.org/announcements/announce-2.0.php (besucht am 14. 02. 2012).

KEITH, JEREMY u. a. (2011). «A Brief History of JavaScript». In: *DOM Scripting*. 10.1007/978-1-4302-3390-9_1. Apress, S. 1–6. ISBN: 978-1-4302-3390-9. URL: http://dx.doi.org/10.1007/978-1-4302-3390-9_1.

KIRSCH, CHRISTIAN (2010). «Chromes 'Kurbelwelle' optimiert JavaScript zur Laufzeit». In: *iX. News*. URL: http://www.heise.de/ix/meldung/Chromes-Kurbelwelle-optimiert-JavaScript-zur-Laufzeit-1149365.html (besucht am 15. 02. 2012).

KOSTER, MARTIJN (1995, 1997, 2007). *Robots in the Web: threat or treat?* URL: http://www.robotstxt.org/threat-or-treat.html (besucht am 11. 03. 2012).

KRILL, PAUL (2008). «JavaScript creator ponders past, future». In: *InfoWorld. Developer World*. URL: http://www.infoworld.com/d/developer-world/javascript-creator-ponders-past-future-704 (besucht am 22. 12. 2011).

LAHN, THOMAS (2011). *ECMAScript Support Matrix*. URL: http://PointedEars.de/scripts/test/es-matrix/.

MAFFEIS, SERGIO, JOHN MITCHELL und ANKUR TALY (2008). «An Operational Semantics for JavaScript». In: *Programming Languages and Systems*. Hrsg. von G. RAMALINGAM. Bd. 5356. Lecture Notes in Computer Science. 10.1007/978-3-540-89330-1_22. Springer Berlin / Heidelberg, S. 307–325. ISBN: 978-3-540-89329-5. URL: http://dx.doi.org/10.1007/978-3-540-89330-1_22.

METZGER, HOLGER (2012). *Geschichte Netscapes*. URL: http://www.holgermetzger.de/netscape/geschichte-netscapes/ (besucht am 14. 02. 2012).

MICROSOFT CORP. (2011a). *MSDN Library. Microsoft JScript Features - Non-ECMA (Visual Studio - JScript)*. URL: http://msdn.microsoft.com/en-us/library/894hfyb4.aspx (besucht am 31. 12. 2011).

– (2011b). *MSDN Library. JScript (ECMAScript3)*. URL: http://msdn.microsoft.com/en-us/library/hbxc2t98(VS.85).aspx (besucht am 31. 12. 2011).

– (2012). *A history of Internet Explorer. Highlights from the first 15 years*. URL: http://windows.microsoft.com/en-US/internet-explorer/products/history (besucht am 18. 02. 2012).

MORITA, HAJIME (2009). «Why Is the New Google V8 Engine So Fast? [Part 1]». In: *Nikkei Electronics Asia*. URL: http://techon.nikkeibp.co.jp/article/HONSHI/20090106/163615/ (besucht am 15. 02. 2012).

MOZILLA.ORG und CONTRIBUTORS (2005). *Firefox (1.5) Release Notes*. URL: http://www.mozilla.org/en-US/firefox/releases/1.5.html (besucht am 15. 02. 2012).

– (2008). *Firefox 3 Release Notes*. URL: http://www.mozilla.org/en-US/firefox/releases/3.0.html (besucht am 15. 02. 2012).

– (2009). *Firefox2. Bon Echo Planning Center*. URL: https://wiki.mozilla.org/Firefox2 (besucht am 15. 02. 2012).

– (2010). *Mozilla Developer Network. New in JavaScript 1.8.1*. URL: https://developer.mozilla.org/en/JavaScript/New_in_JavaScript/1.8.1 (besucht am 15. 02. 2012).

– (2011a). *Firefox 4 RELEASE NOTES*. URL: http://www.mozilla.org/en/firefox/4.0/releasenotes/ (besucht am 15. 02. 2012).

– (2011b). *Mozilla Developer Network. SpiderMonkey*. URL: https://developer.mozilla.org/en/SpiderMonkey (besucht am 15. 02. 2012).

– (2011c). *Mozilla Developer Network. New in JavaScript 1.6*. URL: https://developer.mozilla.org/en/JavaScript/New_in_JavaScript/1.6 (besucht am 15. 02. 2012).

– (2011d). *Mozilla Developer Network. JavaScript Reference*. URL: https://developer.mozilla.org/en/JavaScript/Reference (besucht am 15. 02. 2011).

MOZILLA.ORG und CONTRIBUTORS (2011e). *Mozilla Developer Network. Core JavaScript 1.5 Guide: Statements*. URL: https://developer.mozilla.org/en/Core_

JavaScript_1.5_Guide/Statements#try...catch_Statement (besucht am 10. 03. 2012).

– (2012a). *History of the Mozilla Project*. URL: http://www.mozilla.org/about/history.html (besucht am 15. 02. 2012).
– (2012b). *Mozilla Developer Network. New in JavaScript 1.2*. URL: https://developer.mozilla.org/en/JavaScript/New_in_JavaScript/1.2 (besucht am 13. 02. 2012).
– (2012c). *Mozilla Developer Network. New in JavaScript 1.5*. URL: https://developer.mozilla.org/en/JavaScript/New_in_JavaScript/1.5 (besucht am 15. 02. 2012).
– (2012d). *Mozilla Developer Network. New in JavaScript 1.7*. URL: https://developer.mozilla.org/en/JavaScript/New_in_JavaScript/1.7 (besucht am 15. 02. 2012).
– (2012e). *Mozilla Developer Network. New in JavaScript 1.8*. URL: https://developer.mozilla.org/en/JavaScript/New_in_JavaScript/1.8 (besucht am 15. 02. 2012).

NET APPLICATIONS.COM (2011). *NetMarketshare. Desktop Browser Version Market Share: February, 2012*. URL: http://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustomd=0 (besucht am 12. 03. 2011).

NETSCAPE COMMUNICATIONS CORP. (1996a). *INDUSTRY LEADERS TO ADVANCE STANDARDIZATION OF NETSCAPE'S JAVASCRIPT AT STANDARDS BODY MEETING*. URL: http://web.archive.org/web/19981203070212/http://cgi.netscape.com/newsref/pr/newsrelease289.html (besucht am 15. 02. 2012).

– (1996b). *JavaScript Guide for JavaScript 1.1. Features added after version 1*. The Internet Archive. URL: http://web.archive.org/web/20080529202716/http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/newfunc.htm (besucht am 13. 02. 2012).
– (1998a). *Core JavaScript (1.4) Reference*. Mozilla.org und Contributors. URL: http://devedge-temp.mozilla.org/library/manuals/2000/javascript/1.4/reference/ (besucht am 13. 02. 2012).

NETSCAPE COMMUNICATIONS CORP. (1998b). *NETSCAPE ANNOUNCES PLANS TO MAKE NEXT-GENERATION COMMUNICATOR SOURCE CODE AVAILABLE FREE*

ON THE NET. URL: http://web.archive.org/web/20021001071727/wp.netscape.com/newsref/pr/newsrelease558.html (besucht am 15. 02. 2012).

– (1999). *Client-Side JavaScript (1.3) Reference*. Oracle Corp. and/or its affiliates. URL: http://docs.oracle.com/cd/E19957-01/816-6408-10/ (besucht am 13. 02. 2012).

NEUBAUER, MATTHIAS und PETER THIEMANN (2004). «Haskell type browser». In: *Proceedings of the 2004 ACM SIGPLAN workshop on Haskell*. Haskell '04. Snowbird, Utah, USA: ACM, S. 92–93. ISBN: 1-58113-850-4. DOI: 10.1145/1017472.1017474. URL: http://doi.acm.org/10.1145/1017472.1017474.

NEUMANN, A. (2005). «USE OF SVG AND ECMASCRIPT TECHNOLOGY FOR E-LEARNING PURPOSES». In: *ISPRS Workshop Commissions VI/1 – VI/2 Tools and Techniques for E-Learning*. Potsdam, Germany: Institute of Cartography, ETH Zurich, Switzerland. URL: http://carto.net/neumann/papers/2005/potsdam_2005_use_of_svg_and_ecmascript_for_elearning.pdf (besucht am 11. 03. 2012).

OPERA SOFTWARE ASA (2011). *Opera Mini: web content authoring guidelines. JavaScript support*. URL: http://dev.opera.com/articles/view/opera-mini-web-content-authoring-guidelines/#javascript (besucht am 12. 03. 2011).

– (2012a). *ECMAScript support in Opera Presto 2.10*. URL: http://www.opera.com/docs/specs/presto2.10/ecmascript/ (besucht am 12. 03. 2011).

– (2012b). *Opera version history*. URL: http://www.opera.com/docs/history/ (besucht am 12. 03. 2011).

ORACLE CORP. AND/OR ITS AFFILIATES (2012a). *MySQL 5.1 Reference Manual. 13.5. The MyISAM Storage Engine*. URL: http://dev.mysql.com/doc/refman/5.1/en/myisam-storage-engine.html (besucht am 18. 02. 2012).

– (2012b). *MySQL 5.1 Reference Manual. 13.6. The InnoDB Storage Engine*. URL: http://dev.mysql.com/doc/refman/5.1/en/innodb-storage-engine.html (besucht am 18. 02. 2012).

ORACLE CORP. AND/OR ITS AFFILIATES (2012c). *MySQL 5.1 Reference Manual. 13.6.4.4. FOREIGN KEY Constraints*. URL: http://dev.mysql.com/doc/refman/

5.1/en/innodb-foreign-key-constraints.html (besucht am 18. 02. 2012).

ORENDORFF, JASON (2008). «Iterators and Generators». In: *ECMAScript Harmony Wiki*. URL: http://wiki.ecmascript.org/doku.php?id=proposals:iterators_and_generators (besucht am 15. 02. 2012).

PYTHON SOFTWARE FOUNDATION (2012). *Python v2.7.2 documentation. 5.2. Atoms*. The Python Language Reference. URL: http://docs.python.org/reference/expressions.html#atoms (besucht am 15. 02. 2012).

RAULET, MICKAËL u. a. (2008). «Validation of Bitstream Syntax and Synthesis of Parsers in the MPEG Reconfigurable Video Coding Framework». In: *IEEE Catalog No.: CFP08SIG-CDR*. Washington, D.C. Metro Area, U.S.A: IEEE, S. 1520–6130. URL: http://infoscience.epfl.ch/record/133360/files/MergePDFs(2).pdf.

RESIG, JOHN (2006). *Pro JavaScript techniques*. Apress Series. Apress. ISBN: 9781590597279. URL: http://books.google.de/books?id=GgJN2CC_2s4C.

SMITH, GARRETT, Hrsg. (2010). *comp.lang.javascript FAQ*. Version 32.2. URL: http://jibbering.com/faq/ (besucht am 31. 12. 2011).

STACHOWIAK, MACIEJ (2002). *[KDE-Darwin] JavaScriptCore, Apple's JavaScript framework based on KJS*. URL: http://web.archive.org/web/20060405170813/opendarwin.org/pipermail/kde-darwin/2002-June/000034.html (besucht am 13. 02. 2012).

– (2008). «Introducing SquirrelFish Extreme». In: *Surfin' Safari*. URL: http://www.webkit.org/blog/214/introducing-squirrelfish-extreme/ (besucht am 17. 02. 2012).

VAN CUTSEM, TOM (2010). *Changes to ECMAScript, Part 2: Harmony Highlights - Proxies and Traits*. URL: http://www.youtube.com/watch?v=A1R8KGKkDjU (besucht am 03. 02. 2012).

– (2012). «Direct Proxies». In: *ECMAScript Harmony Wiki*. URL: http://wiki.ecmascript.org/doku.php?id=harmony:direct_proxies (besucht am 15. 02. 2012).

VEITCH, MARTIN (2001). «Five years ago: Microsoft releases Internet Explorer 3.0 second beta». In: *ZDNet UK*. URL: http://www.zdnet.co.uk/news/it-strategy/

2001/07/17/five-years-ago-microsoft-releases-internet-explorer-30-second-beta-2091366/ (besucht am 13. 02. 2012).

VIGNA, SEBASTIANO (2002). «ERW: Entities and Relationships on the Web». In: *Poster Proc. of Eleventh International World Wide Web Conference*. Honolulu, Hawaii, USA. URL: http://vigna.di.unimi.it/ftp/papers/www2002c/.

WEBKIT-PROJEKT (2011). *The WebKit Open Source Project. JavaScript*. URL: http://www.webkit.org/projects/javascript/ (besucht am 31. 12. 2011).

WILSON, BRIAN (2005). *Browser Timelines. Netscape Navigator (Netscape Communications®)*. URL: http://www.blooberry.com/indexdot/history/netscape.htm (besucht am 18. 02. 2012).

WOLF, ERIC B. und KEVIN HOWE (2009). «Web-Client Based Distributed Generalization and Geoprocessing». In: *Proceedings of the 2009 International Conference on Advanced Geographic Information Systems & Web Services*. GEOWS '09. Washington, DC, USA: IEEE Computer Society, S. 123–128. ISBN: 978-0-7695-3527-2. DOI: 10.1109/GEOWS.2009.32. URL: http://dx.doi.org/10.1109/GEOWS.2009.32.

YU, DACHUAN u. a. (2007). «JavaScript instrumentation for browser security». In: *Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '07. Nice, France: ACM, S. 237–249. ISBN: 1-59593-575-4. DOI: 10.1145/1190216.1190252. URL: http://doi.acm.org/10.1145/1190216.1190252.

YUI THEATER (2007). *Douglas Crockford: The JavaScript Programming Language*. Ca. 00:04:00. URL: http://www.youtube.com/watch?v=v2ifWcnQs6M (besucht am 13. 02. 2012).

ZAYTSEV, JURIY (2012). *ECMAScript 5 compatibility table*. URL: http://kangax.github.com/es5-compat-table/ (besucht am 12. 02. 2012).

A. Anhang

A.1. Backend

A.1.1. MySQL-Anweisungen

A.1.1.1. Datenbankstruktur

es-matrix-structure.sql

```
-- phpMyAdmin SQL Dump
-- version 3.3.7deb7
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Erstellungszeit: 12. M rz 2012 um 21:23
-- Server Version: 5.1.61
-- PHP-Version: 5.3.3-7+squeeze8

SET FOREIGN_KEY_CHECKS=0;
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT=0;
START TRANSACTION;


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS
    */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Datenbank: `db_mw3020_1`
--

```

```
-- --------------------------------------------------------

--
-- Tabellenstruktur f r Tabelle `environment`
--

DROP TABLE IF EXISTS `environment`;
CREATE TABLE IF NOT EXISTS `environment` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `version_id` int(10) unsigned DEFAULT NULL,
  `sortorder` int(11) DEFAULT '0',
  `name` varchar(255) DEFAULT NULL,
  `user_agent` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `user_agent` (`user_agent`),
  UNIQUE KEY `name` (`name`),
  KEY `version_id` (`version_id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=106 ;

-- --------------------------------------------------------

--
-- Tabellenstruktur f r Tabelle `feature`
--

DROP TABLE IF EXISTS `feature`;
CREATE TABLE IF NOT EXISTS `feature` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `code` varchar(512) NOT NULL,
  `title` varchar(255) DEFAULT NULL,
  `edition` varchar(3) DEFAULT NULL,
  `section` varchar(25) DEFAULT NULL,
  `section_urn` varchar(25) DEFAULT NULL,
  `generic` tinyint(1) unsigned NOT NULL DEFAULT '0' COMMENT '
     Intentionally generic?',
  `versioned` tinyint(1) unsigned NOT NULL DEFAULT '0' COMMENT '
     Version needs to be declared in order to use this feature',
  `created` timestamp NULL DEFAULT NULL,
  `modified` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
     UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=289 ;

-- --------------------------------------------------------
```

```
66
--
68 -- Tabellenstruktur f r Tabelle `implementation`
--
70
DROP TABLE IF EXISTS `implementation`;
72 CREATE TABLE IF NOT EXISTS `implementation` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
74  `sortorder` int(10) unsigned DEFAULT NULL,
  `name` varchar(50) NOT NULL,
76  `acronym` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`id`),
78  UNIQUE KEY `name_UNIQUE` (`name`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=13 ;
80
-- --------------------------------------------------------
82
--
84 -- Tabellenstruktur f r Tabelle `result`
--
86
DROP TABLE IF EXISTS `result`;
88 CREATE TABLE IF NOT EXISTS `result` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
90  `testcase_id` int(10) unsigned NOT NULL,
  `env_id` int(10) unsigned NOT NULL COMMENT 'Host environment ID
      ',
92  `value` tinyint(1) NOT NULL,
  PRIMARY KEY (`id`),
94  UNIQUE KEY `testcase_id` (`testcase_id`,`env_id`),
  KEY `environment_id` (`env_id`)
96 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COMMENT='Test results'
    AUTO_INCREMENT=22099 ;

98 -- --------------------------------------------------------

100 --
-- Tabellenstruktur f r Tabelle `testcase`
102 --

104 DROP TABLE IF EXISTS `testcase`;
CREATE TABLE IF NOT EXISTS `testcase` (
106  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `feature_id` int(10) unsigned NOT NULL,
```

```
108   `title` varchar(255) DEFAULT NULL,
      `code` text NOT NULL,
110   `quoted` tinyint(1) unsigned NOT NULL DEFAULT '0',
      `alt_type` varchar(50) DEFAULT NULL COMMENT 'alternative type
          attribute value',
112   `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
      PRIMARY KEY (`id`),
114   KEY `feature_id` (`feature_id`)
    ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=3438 ;
116
    -- --------------------------------------------------------
118
    --
120 -- Tabellenstruktur f r Tabelle `version`
    --
122
    DROP TABLE IF EXISTS `version`;
124 CREATE TABLE IF NOT EXISTS `version` (
      `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
126   `impl_id` int(10) unsigned DEFAULT NULL,
      `name` varchar(25) NOT NULL,
128   `safe` tinyint(1) unsigned NOT NULL DEFAULT '0',
      PRIMARY KEY (`id`),
130   UNIQUE KEY `implementation_id` (`impl_id`,`name`),
      KEY `fk_version_implementation1` (`impl_id`)
132 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=38 ;
134 --
    -- Constraints der exportierten Tabellen
136 --
138 --
    -- Constraints der Tabelle `environment`
140 --
    ALTER TABLE `environment`
142   ADD CONSTRAINT `environment_ibfk_1` FOREIGN KEY (`version_id`)
          REFERENCES `version` (`id`) ON DELETE NO ACTION ON UPDATE
          NO ACTION;
144 --
    -- Constraints der Tabelle `result`
146 --
    ALTER TABLE `result`
```

```
148   ADD CONSTRAINT `result_ibfk_3` FOREIGN KEY (`testcase_id`)
          REFERENCES `testcase` (`id`) ON DELETE CASCADE ON UPDATE
          CASCADE,
      ADD CONSTRAINT `result_ibfk_4` FOREIGN KEY (`env_id`)
          REFERENCES `environment` (`id`) ON DELETE CASCADE ON UPDATE
           CASCADE;
150
--
152 -- Constraints der Tabelle `testcase`
--
154 ALTER TABLE `testcase`
      ADD CONSTRAINT `testcase_ibfk_2` FOREIGN KEY (`feature_id`)
          REFERENCES `feature` (`id`) ON DELETE CASCADE ON UPDATE
          CASCADE;
156
--
158 -- Constraints der Tabelle `version`
--
160 ALTER TABLE `version`
      ADD CONSTRAINT `version_ibfk_2` FOREIGN KEY (`impl_id`)
          REFERENCES `implementation` (`id`) ON DELETE CASCADE ON
          UPDATE CASCADE;
162 SET FOREIGN_KEY_CHECKS=1;
COMMIT;
```

A.1.2. Web-Applikation

A.1.2.1. Framework (Auszug)

[in dieser Version nicht enthalten; siehe Website]

A.1.2.2. Applikation (Auszug)

[in dieser Version nicht enthalten; siehe Website]

A.2. Frontend

A.2.1. Existenztest

```
/**
 * Determines whether an object is, or several objects are,
 * likely to be callable.
 *
 * @author (C) 2003-2010  <a href="mailto:js@PointedEars.de">
    Thomas Lahn</a>
 * @function
 * @param obj : Object which should be tested for a method, or
    checked
 *   for being a method if no further arguments are provided.
 *   <p>
 *   <em>NOTE: If you pass a primitive value for this argument,
 *   the properties of the object created from that value are
    considered.
 *   In particular, if you pass a string value containing
 *   a <i>MemberExpression</i>, the properties of the
    corresponding
 *   <code>String</code> instance are considered, not of the
    object that
 *   the <i>MemberExpression</i> might refer to.  If you need to
    use such
 *   a string to refer to an object (e.g., if you do not know
    whether it
 *   is safe to refer to the object), use the return value of
 *   {@link jsx#tryThis jsx.tryThis("<var>MemberExpression</var
    >")}
 *   as argument to this method instead.</em>
 *   </p>
 * @param prop : optional string|Array
 *   Path of the property to be determined a method, i.e. a
    reference to
 *   a callable object assigned as property of another object.
 *   Use a string argument for each component of the path, e.g.
 *   the argument list <code>(o, "foo", "bar")</code> for testing
     whether
 *   <code>o.foo.bar</code> is a method.
 *   If the last argument is an {@link Array}, all elements of
 *   this array are used for property names; e.g.
```

```
29  *   <code>(o, "foo", ["bar", "baz"])</code>.  This allows for
        testing
    *   several properties of the same object with one call.
31  * @return boolean
    *   <code>true</code> if all arguments refer to methods,
33  *   <code>false</code> otherwise.
    * @see jsx.object#isMethodType()
35  */
function jsx_isMethod_wrapper ()
37  {
      var areNativeMethods = null;
39
      function isMethod (obj, prop)
41    {
        var len = arguments.length;
43      if (len < 1)
        {
45        jsx.throwThis("jsx.InvalidArgumentError",
            ["Not enough arguments", "saw 0", "(obj : Object[, prop :
                string])"]);
47        return false;
        }
49
        /*
51       * Determine if we were apply'd by jsx.object.
             areNativeMethods;
         * NOTE: cache reference
53       */
        var checkNative =
55        (this == (areNativeMethods
                    || (areNativeMethods = jsx.object.
                        areNativeMethods)));
57
        var t = typeof obj;
59
        /* When no property names are provided, test if the first
            argument is a method */
61      if (len < 2)
        {
63        if (checkNative)
          {
65          return t == "function" && obj && true || false;
          }
67
```

```
      return t == "unknown" || (t == "function" || t == "object")
           && obj && true || false;
69  }

71  /* otherwise the first argument must refer to a suitable
       object */
    if (t == "unknown" || !obj)
73  {
      return false;
75  }

77  for (var i = 1; i < len; i++)
    {
79    prop = arguments[i];

81    /* NOTE: Handle null _and_ undefined */
      if (prop == null)
83    {
        return false;
85    }

87    var isLastSeg = (i == len - 1);
      if (isLastSeg)
89    {
        if (typeof prop.valueOf() == "string")
91      {
          prop = [prop];
93      }

95      var aProp = prop;
      }
97
      for (var j = (isLastSeg && aProp.length || 1); j--;)
99    {
        if (isLastSeg)
101     {
          prop = aProp[j];
103     }

105     t = typeof obj[prop];

107     /*
         * NOTE: Test for "unknown" required in any case;
109      * this order speeds up evaluation
```

```
          */
111       if (t == "unknown" || ((t == "function" || t == "object")
              && obj[prop]))
          {
113         if (i < len - 1)
            {
115           obj = obj[prop];
              if (!(typeof obj == "unknown" || obj))
117           {
                return false;
119           }
            }
121         else if (checkNative && t != "function")
            {
123           return false;
            }
125       }
          else
127       {
            return false;
129       }
        }
131   }

133   return true;
  }
135
  return isMethod;
137 }
jsx.object.isMethod = jsx.object.areMethods =
139 jsx.object.isHostMethod = jsx.object.areHostMethods =
    jsx_isMethod_wrapper();

141 /**
 * Determines whether an object is, or several objects are,
143  * likely to be a native method.
 *
145  * @author (C) 2011  <a href="mailto:js@PointedEars.de">Thomas
     Lahn</a>
 * @function
147  * @param obj : Object which should be tested for a method, or
     checked
 *   for being a method if no further arguments are provided.
149  *   <p>
```

```
 *   <em>NOTE: If you pass a primitive value for this argument,
151  *   the properties of the object created from that value are
     considered.
 *   In particular, if you pass a string value containing
153  *   a <i>MemberExpression</i>, the properties of the
     corresponding
 *   <code>String</code> instance are considered, not of the
     object that
155  *   the <i>MemberExpression</i> might refer to.  If you need to
     use such
 *   a string to refer to an object (e.g., if you do not know
     whether it
157  *   is safe to refer to the object), use the return value of
 *   {@link jsx#tryThis jsx.tryThis("<var>MemberExpression</var
     >")}
159  *   as argument to this method instead.</em>
 *   </p>
161  * @param prop : optional string|Array
 *   Path of the property to be determined a method, i.e. a
     reference to
163  *   a callable object assigned as property of another object.
 *   Use a string argument for each component of the path, e.g.
165  *   the argument list <code>(o, "foo", "bar")</code> for testing
      whether
 *   <code>o.foo.bar</code> is a method.
167  *   If the last argument is an {@link Array}, all elements of
 *   this array are used for property names; e.g.
169  *   <code>(o, "foo", ["bar", "baz"])</code>.  This allows for
     testing
 *   several properties of the same object with one call.
171  * @return boolean
 *   <code>true</code> if all arguments refer to methods,
173  *   <code>false</code> otherwise.
 * @see jsx.object#isMethodType()
175  */
jsx.object.isNativeMethod = jsx.object.areNativeMethods = (
    function() {
177   var areMethods = jsx.object.areMethods;

179   return function(obj, prop) {
      /* NOTE: Thread-safe, argument-safe code reuse -- 'this' is
          our ID */
181     return areMethods.apply(arguments.callee, arguments);
   };
```

```
183 }());

185 /* ... */

187 /**
     * Returns a feature of an object
189  *
     * @param obj : Object
191  * @return mixed
     *   <code>false</code> if <var>obj</var> does not have such a
         feature
193  */
    jsx.object.getFeature = function(obj) {
195   for (var i = 1, len = arguments.length; i < len; i++)
      {
197     var arg = arguments[i];
        if (typeof obj != "undefined" && typeof obj[arg] != "
            undefined" && obj[arg])
199     {
          obj = obj[arg];
201     }
        else
203     {
          return false;
205     }
      }
207
      return obj;
209 };
```

A.2.2. Testfunktion

```
/**
 * Clears the handler for the proprietary <code>error</code>
    event.
 *
 * @return boolean <code>true</code>
 */
jsx.clearErrorHandler = function() {
  if (typeof window != "undefined" && window.onerror)
  {
    window.onerror = null;
  }

  return true;
};

/**
 * Sets the handler for the proprietary <code>error</code> event.
 *
 * @function
 * @param fHandler : Callable
 * @return boolean
 *   <code>true</code> if the error handler could be assigned to
 *   successfully, <code>false</code> otherwise.  Note that one
    reason
 *   for failure can be that an event handler is no longer
    supported
 *   by the UA's DOM due to efforts towards adherence to Web
    standards.
 */
jsx.setErrorHandler = (function() {
  var
    jsx_object = jsx.object,
    jsx_clearErrorHandler = jsx.clearErrorHandler;

  return function(fHandler) {
    if (!jsx_object.isMethod(fHandler))
    {
      fHandler = jsx_clearErrorHandler;
    }

    if (typeof assertFalse == "function")
```

```
    {
      assertFalse(typeof fHandler == "undefined", false,
        "jsx.setErrorHandler(fHandler)");
    }

    if (typeof window != "undefined"
        && typeof window.onerror != "undefined"
        && typeof fHandler != "undefined")
    {
      window.onerror = fHandler;
    }

    return (typeof window.onerror != "undefined"
            && window.onerror == fHandler);
  };
}());

/**
 * Returns a value depending on whether an expression evaluates
    to
 * a true-value or a false-value.
 *
 * @param expression
 *   Expression to be evaluated.  Parsed as a <i>Program</i> if a
 *   <code>String</code>.
 * @param trueValue
 *   The value to be returned if <var>expression</var> evaluates
    to
 *   a true-value; the default is the empty string.
 * @param falseValue
 *   The value to be returned if <var>expression</var> evaluates
    to
 *   a false-value; the default is the empty string.
 * @return mixed
 *   <var>trueValue</var> if <var>expression</var> evaluates to
 *   a true-value; <var>falseValue</var> otherwise.
 * @see Global#eval(String)
 */
jsx.debug.test = function (expression, trueValue, falseValue) {
  var sDefault = "";
  var result = sDefault;

  jsx.setErrorHandler();
```

```
79    if (jsx.tryThis(expression))
      {
81      result = (arguments.length > 1 ? trueValue : sDefault);
      }
83    else
      {
85      result = (arguments.length > 2 ? falseValue : sDefault);
      }
87
      jsx.clearErrorHandler();
89    return result;
};
```

A.2.3. Getestete Implementierungen

Netscape/Mozilla JavaScript			
Version	Browser	Browserversion	Betriebssystem
1.0	Netscape Navigator	2.02	Windows (Wine)
1.1		3.04	
1.2		4.04	
1.3	Netscape Communicator	4.8	
1.5	Mozilla Firefox	1.0.8	
1.6		1.5.0.12	
1.7		2.0	
1.8		3.0.19	
1.8.1	Debian Iceape	2.0.11-9	Debian GNU/Linux
1.8.5	Debian Iceweasel	10.0.2	
Microsoft JScript			
5.5.6330	Internet Explorer	5.50.4807.2300	Windows XP (Wine)
5.6.6626		6.0.2800.1106	
5.7.22589		7.0.5730.13	Windows Server 2003
5.8.23141		8.0.6001.18702IC	Windows XP Pro 2002 SP3
9.0.16440		9.0.8112.16421	Windows 7 Professional 32 bit SP1
Google V8			
3.7.12.12	Chromium	17.0.963.78	Debian GNU/Linux
3.8.9.5	Google Chrome	18.0.1025.45 beta	
Apple JavaScriptCore			
531.22.7	Apple Safari	4.0.5 (531.22.7)	Windows XP (Wine)
7533.18.5		5.0.2 (7533.18.5)	
7534.52.7		5.1.2 (7534.52.7)	Windows 7 32-bit SP1
Opera ECMAScript			
7.02	Opera	7.02.2668	Windows XP (Wine)
8.0		8.0.7561	
9.27		9.27.8841	

9.52		9.52.10108	
9.62		9.62.10467	
9.64		9.64.10487	
10.53		10.53.3374	
10.54		10.54.3423	
11.61		11.61.1250	Debian GNU/Linux
KDE JavaScript			
4.6.5	Konqueror	4.6.5	Debian GNU/Linux

Tabelle A.1.: Getestete Implementierungen

A.2.4. Getestete Features

ID	Code	Beschreibung
261	`!`	Logical NOT operator
1	`!==`	Strict Not Equal/Nonidentity operator
2	`"\uhhhh": string`	Unicode escape sequence in String literal
3	`"foo\bar": string`	Escaped newline in String literal
288	`"use strict";`	Strict Mode
156	`(String in Object) : boolean`	'in' Operator
4	`+expression : number`	Unary plus (converts to Number)
10	`/(?!...)/ : RegExp`	Regular Expression with zero-width negative lookahead
11	`/(?:...)/ : RegExp`	Regular Expression with non-capturing parentheses
12	`/(?=...)/ : RegExp`	Regular Expression with zero-width positive lookahead
14	`/(...)\1/ : RegExp`	Regular Expression with backreferences
13	`/(...+?|...*?)/ : RegExp`	Regular Expression with non-greedy matching
5	`/[/]/ : RegExp`	RegExp literal with unescaped forward slash in character class
6	`/[^]/ : RegExp`	RegExp literal with empty negated character range
15	`/\uhhhh/ : RegExp`	Unicode escape sequence in RegExp literal
7	`/.../[g][i] : RegExp`	RegExp literal with only optional global and case-insensitive modifier
8	`/.../[g][i][m]: RegExp`	RegExp literal with optional multiline modifier
9	`/.../[g][i][m][y] : RegExp`	RegExp literal with optional sticky modifier
132	`= function identifier(...) {...} : Function`	Function expression
134	`= function (...) expression : Function`	
133	`= function (...) {...} : Function`	Anonymous function expression
18	`==`	Equals operator
19	`===`	Strict Equals/Identity operator

22	`[`*expression* `for each (`*propertyValue* `in` *Object*`) [if (`*condition*`)]] : Array`	Array comprehension
20	`[`*value*`, ...] : Array`	Array initializer
21	`[`*value*`, ] : Array`	Array initializer with trailing comma
23	`[var] [`*var1*`, [`*var2*`], `*var3*`, ...] = `*Array*	Destructuring assignment
29	`arguments`	
263	`arguments : Arguments`	
30	`arguments.callee : Function`	
32	`arguments.length : number|int`	
33	`Array(...)`	Array constructor/factory
34	`Array.every(`*iterable*`, `*callback* `: Function) : boolean`	
35	`Array.isArray(`*arg*`) : boolean`	
36	*array*`.length : number|int`	
38	`Array.prototype : Array`	
39	`Array.prototype.concat(...) : Array`	
40	`Array.prototype.constructor : Array`	
41	`Array.prototype.every(`*callback* `: Function[, `*thisValue*`]) : boolean`	
42	`Array.prototype.filter(`*callback* `: Function[, `*thisArg* `: Object]) : number|int`	
43	`Array.prototype.forEach(`*callback* `: Function[, `*thisArg* `: Object]) : number|int`	
44	`Array.prototype.indexOf(`*searchElement*`[, `*fromIndex* `: Number|int]) : number|int`	
45	`Array.prototype.join(`*separator* `: String) : string`	

```
46 Array.prototype.lastIndexOf(
   searchElement[, fromIndex
   : Number|int]) :
   number|int
47 Array.prototype.length :
   number|int
48 Array.prototype.map(callback
   : Function[, thisArg :
   Object]) : number|int
49 Array.prototype.pop()
50 Array.prototype.push([item1[,
   item2[, ...]]]) :
   number|int
51 Array.prototype.reduce(
   callback : Function[,
   initialValue]) : any
52 Array.prototype.reduceRight(
   callback : Function[,
   initialValue]) : any
53 Array.prototype.reverse()
   : Array
54 Array.prototype.shift()
55 Array.prototype.slice(start
   : Number|int[, end :
   Number|int]) : Array
56 Array.prototype.some(callback
   : Function[, thisValue]) :
   boolean
57 Array.prototype.sort(...)
   : Object
58 Array.prototype.splice(start
   : Number|int, deleteCount
   : Number|int[, item1[,
   item2[, ...]]]) : Array
59 Array.prototype.toSource()
   : string
60 Array.prototype.toString()
   : string
61 Array.prototype.unshift()
   : number|int
37 Array.some(iterable,
   callback : Function) :
   boolean
```

```
 62  boolean
 63  Boolean(...)
 64  Boolean.prototype :
     Boolean
 65  Boolean.prototype.toSource()
     : string
 66  Boolean.prototype.toString()
     : string
 67  Boolean.prototype.valueOf()
     : boolean
 68  byte
 70  char
286  Date.now()
 73  Date.prototype : Date
 74  Date.prototype.getFullYear()
     : number|int
 75  Date.prototype
     .getMilliseconds() :
     number|int
 76  Date.prototype.getUTCDate()
     : number|int
 77  Date.prototype.getUTCDay()
     : number|int
 78  Date.prototype
     .getUTCFullYear() :
     number|int
 79  Date.prototype.getUTCHours()
     : number|int
 80  Date.prototype
     .getUTCMilliseconds() :
     number|int
 81  Date.prototype.getUTCMinutes()
     : number|int
 82  Date.prototype.getUTCMonth()
     : number|int
 83  Date.prototype.getUTCSeconds()
     : number|int
 84  Date.prototype.getVarDate()
 85  Date.prototype.setFullYear(
     year : Number|int[, month
     : Number|int[, date :
     Number|int]]) : number|int
```

```
86 Date.prototype
   .setMilliseconds(Number|int)
   : number|int
87 Date.prototype.setUTCDate(
   Number|int) : number|int
89 Date.prototype
   .setUTCFullYear(year
   : Number|int[, month
   : Number|int[, date :
   Number|int]]) : number|int
90 Date.prototype.setUTCHours(
   hours : Number|int[,
   minutes : Number|int[,
   seconds : Number|int]],
   ms : Number|int]]) :
   number|int
91 Date.prototype
   .setUTCMilliseconds(
   Number|int) : number|int
92 Date.prototype
   .setUTCMinutes(minutes
   : Number|int[, seconds
   : Number|int[, ms :
   Number|int]]) : number|int
93 Date.prototype.setUTCMonth(
   month : Number|int[,
   date : Number|int]) :
   number|int
94 Date.prototype.setUTCSeconds(
   seconds : Number|int[,
   ms : Number|int]) :
   number|int
95 Date.prototype.toDateString()
   : string
96 Date.prototype.toGMTString()
   : string
97 Date.prototype.toISOString()
   : string
98 Date.prototype.toJSON([key])
   : string
99 Date.prototype
   .toLocaleDateString() :
   string
```

```
100 Date.prototype.toLocaleFormat(
    format : String) : string
101 Date.prototype
    .toLocaleString() : string
102 Date.prototype
    .toLocaleTimeString() :
    string
103 Date.prototype.toSource()
    : string
104 Date.prototype.toString()
    : string
105 Date.prototype.toTimeString()
    : string
106 Date.prototype.toUTCString()
    : string
109 decimal
110 decodeURI(String) : string
111 decodeURIComponent(String)
    : string
112 delete
114 double
113 do...while
115 encodeURI(String)
116 encodeURIComponent(String)
    : string
118 Enumerator(...)
119 Error([message : String])      Error constructor/factory
120 error.description : string
123 error.number : number|int
121 Error.prototype.message :
    string
122 Error.prototype.name :
    string
124 error.stack : string
128 for each ([var] identifier
    in Object)
129 Function([p1 : String[,        Function constructor/factory
    p2 : String[, ...], pn :
    String],] body : String) :
    Function
139 function.arguments :
    arguments
```

```
140  function.arguments.callee
     : Function
141  function.arguments.length
     : number|int
137  function.arity :
     number|int
143  function.caller :
     Function|null
144  function.length :
     number|int
264  function.prototype
136  Function.prototype :
     Function
138  Function.prototype.apply(...)
 31  Function.prototype.arguments
     .caller : Function|null
287  Function.prototype.bind(
     thisArg[, arg1[, arg2,
     ...]]) : Function
142  Function.prototype.call(...)
146  Function.prototype.toSource()
     : string
147  generator.close()
148  generator.next()
149  generator.send(expression)
150  generator.throw(expression)
151  GetObject(...)
152  Global object
 17  identifier : type              Type declaration
135  if (...) { function f() {     Function statement
     ... }; }
157  Infinity : number
158  instanceof
159  int
162  isFinite(...) : boolean
163  Iterator(Object) :
     Iterator
164  JSON.parse(text :
     String[,reviver :
     Function]) : Object
```

165	`JSON.stringify(value[, replacer[, space]]) : string`	
16	`label:`	Label
166	`let (assignment[, ...]) { [statements] }`	Block scoping: let statement
167	`let (assignment[, ...]) expression`	Block scoping: let expression
168	`let assignment[, ...]`	Block scoping: let definition
169	`long`	
170	`Math.max(a : Number, b : Number) : number`	
171	`Math.max(a : Number, b : Number, ...) : number`	
172	`Math.min(a : Number, b : Number) : number`	
173	`Math.min(a : Number, b : Number, ...) : number`	
174	`NaN : number`	Not-a-number value
28	`new ActiveXObject("serverName.typeName"[, location : String])`	
175	`Number([expression]) : number`	
176	`Number.MAX_VALUE : number`	
177	`Number.MIN_VALUE : number`	
178	`Number.NaN : number`	
179	`Number.NEGATIVE_INFINITY : number`	
180	`Number.POSITIVE_INFINITY : number`	
181	`Number.prototype : Number`	
182	`Number.prototype.toString() : string`	
183	`Object([expression]) : Object`	
262	`object.__noSuchMethod__ : Function`	__noSuchMethod__ method
266	`Object.create(obj : Object[, properties])`	

```
184 Object.defineProperties(o
    : Object, properties :
    Object) : Object
185 Object.defineProperty(o :
    Object, property : String,
    attr : Object) : Object
273 Object.freeze(o : Object)
    : Object
285 Object
    .getOwnPropertyDescriptor(obj
    : Object, property :
    String) : Object
186 Object.getOwnPropertyNames(o
    : Object) : Array
187 Object.getPrototypeOf(o :
    Object) : Object
284 Object.isExtensible(o :
    Object) : boolean
279 Object.isFrozen(o :
    Object) : boolean
277 Object.isSealed(o :
    Object) : boolean
268 Object.keys(o : Object) :
    Array
276 Object.preventExtensions(o
    : Object) : Object
188 Object.prototype : Object
189 Object.prototype
    .__defineGetter__(propertyName
    : String, getter :
    Function)
190 Object.prototype
    .__defineSetter__(propertyName
    : String, setter :
    Function)
192 Object.prototype.__proto__
193 Object.prototype.constructor
    : Function
194 Object.prototype
    .hasOwnProperty(
    propertyName : String) :
    boolean
```

```
195 Object.prototype
    .isPrototypeOf(o : Object)
    : boolean
196 Object.prototype
    .propertyIsEnumerable(
    propertyName : String) :
    boolean
197 Object.prototype.toSource()
    : string
269 Object.seal(obj : Object)
    : Object
201 print(string)
213 reArray.index
211 reArray.input
205 RegExp(...[, "[g][i]"]) :
    RegExp
206 RegExp(...[, "[g][i][m]"])
    : RegExp
207 RegExp.$integer
216 regExp.global
217 regExp.ignoreCase
218 regExp.multiline
214 RegExp.prototype.compile(...)
215 RegExp.prototype.exec(...)
219 regExp.source
208 RegExp{["$&"] |
    .lastMatch}
209 RegExp{["$'"] |
    .rightContext}
210 RegExp{["$+"] |
    .lastParen}
212 RegExp{["$`"] |
    .leftContext}
220 sbyte
221 ScriptEngine()
222 ScriptEngineBuildVersion()
223 ScriptEngineMajorVersion()
224 ScriptEngineMinorVersion()
226 short
227 String.fromCharCode(
    Number|uint)
228 String.prototype
```

```
229 String.prototype.charCodeAt(
    Number|uint)
230 String.prototype.concat(...)
231 String.prototype
    .localeCompare(string)
232 String.prototype.match(
    RegExp)
234 String.prototype.replace(
    string|RegExp, Function)
233 String.prototype.replace(
    string|RegExp, string)
235 String.prototype.search(
    RegExp)
236 String.prototype.slice(...)
265 String.prototype.split(
    separator :
    String|RegExp[, limit :
    Number|int])
237 String.prototype.split(
    String)
238 String.prototype.substr(
    start[, length])
239 String.prototype.trim() :
    string
240 string[Number|uint]            String subscripting
241 switch (expression) {
    case value: statements;
    [break;] ... default:
    statements; [break;] }
242 throw expression
243 try { [statements] }
    catch (identifier) {
    [statements] }
245 try { [statements] }
    catch (identifier) {
    [statements] } finally {
    [statements] }
246 try { [statements] }
    catch (identifier if
    expression) { [statements]
    } [catch (identifier) {
    [statements] }] [finally {
    [statements] }]
```

244	`try { [statements] } finally { [statements] }`	
247	`typeof expression`	
248	`undefined`	
24	`var \uhhhh`	Unicode escape sequence in Identifier
267	`var π`	Non-Latin Unicode letter in identifier
249	`VBArray.prototype.dimensions()`	
250	`VBArray.prototype.getItem(...)`	
251	`VBArray.prototype.lbound(...)`	
252	`window`	
257	`yield expression`	Generator expression
25	`{propertyName: propertyValue, ...} : Object`	Object initializer
26	`{propertyName: propertyValue, } : Object`	Object initializer with trailing comma

Tabelle A.2.: Getestete Features

A.2.5. Testfälle

Die Testfälle sind jeweils mit der ID des Features gekennzeichnet, um eine Zuordnung zu diesem zu ermöglichen (vgl. Anhang A.2.4). Sie sind für eine einfachere Referenzierung je Feature fortlaufend nummeriert.

ID	Nr.	Code
261	1	`typeof !void(0) == "boolean" && !void(0)`
	2	`typeof !null == "boolean" && !null`
	3	`typeof !false == "boolean" && !false`
	4	`typeof !true == "boolean" && !!true`
	5	`typeof !0 == "boolean" && !0`
	6	`typeof !-0 == "boolean" && !-0`
	7	`typeof !NaN == "boolean" && !NaN`
	8	`typeof !"" == "boolean" && !""`
	9	`typeof !"0" == "boolean" && !!"0"`
	10	`typeof !new Object() == "boolean" && !!new Object()`
1	1	`1 !== "1"`
	2	`var x; !(x !== x)`
	3	`!(null !== null)`
	4	`NaN !== 1`
	5	`1 !== NaN`
	6	`!(1 !== 1)`
	7	`!(+0 !== -0)`
	8	`!(-0 !== +0)`
	9	`1 !== 2`
	10	`("a" !== "b") && ("a" !== "ab")`
	11	`(true !== false) && (false !== true)`
	12	`(new Object()) !== (new Object())`
2	1	`"\u20AC" == "€"`
3	1	`(eval("'foo\\\nbar'") || "\n").indexOf("\n") < 0`
288	1	`eval('"use strict"; var x = 42;'); typeof x == "undefined"`
	2	`function f (a) { "use strict"; arguments[0] = 23; return a; } f(42) == 42`

```
3  function f () { "use strict"; return
   arguments.caller; }
   function f2 () { "use strict"; return f2.caller; }
   var result = jsx.tryThis(f, "e");
   var result2 = jsx.tryThis(f2, "e");
   result && result.name == "TypeError" && result2
   && result2.name == "TypeError"
4  function f () { "use strict"; return
   arguments.callee; }
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
5  function f () { "use strict"; return f.arguments;
   }
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
6  function f () {
   "use strict";
   var o = new Object();
   var desc = new Object();
   desc.value = 42;
   desc.configurable = false;
   Object.defineProperty(o, "answer", desc);
   delete o.answer;
   }
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
7  function f () { "use strict"; x = 42; }
   var result = jsx.tryThis(f, "e");
   result && result.name == "ReferenceError"
8  function f ()
   {
   "use strict";
   var o = new Object();
   var desc = new Object();
   desc.value = 42;
   desc.writable = false;
   Object.defineProperty(o, "answer", desc);
   o.answer = 23;
   }
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
```

	9	`function f2 () { return 42; }`
		`function f ()`
		`{`
		`"use strict";`
		`var o = new Object();`
		`var desc = new Object();`
		`desc["get"] = f2;`
		`Object.defineProperty(o, "answer", desc);`
		`o.answer = 23;`
		`}`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`

	10	`function f ()`
		`{`
		`"use strict";`
		`var o = new Object();`
		`var desc = new Object();`
		`desc.value = new Object();`
		`desc.extensible = false;`
		`Object.defineProperty(o, "child", desc);`
		`o.child.answer = 42;`
		`}`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`

	11	`typeof (function () { "use strict"; return this;` `}()) == "undefined"`

	12	`function string_foo () { "use strict"; return`
		`this; }`
		`String.prototype.foo = string_foo;`
		`typeof "".foo() == "string"`

156	1	`var o = new Object(); o.foo = "bar"; "foo" in o &&` `!("bar" in o)`
4	1	`+"42" == 42`
10	1	`"abac".match(/a(?!b)./) == "ac"`
11	1	`"ab".match(/a(?:b)/) == "ab"`
12	1	`"ab".match(/a(?=b)/) == "a"`
14	1	`("aaa".match(/^(a)\1/) || new Array(""))[0] ==` `"aa"`
13	1	`"aaa".match(/^aa*?/) == "a" &&` `"aaa".match(/^aa+?/) == "aa"`

5	1	`"/".match(/[/]/).length == 1`
6	1	`"\n".match(/[^]/).length == 1`
15	1	`/\u20AC/.test('€')`
7	1	`"aA".match(/a/gi).length == 2`
8	1	`"a\nA".match(/^a$/gim).length == 2`
9	1	`var rx = /^./y, s = "ab"; rx.exec(s) == "a" && rx.exec(s) == "b"`
132	1	`(function foo () { return 42; })() == 42`
134	1	`(function (x) x * x)(2) == 4`
133	1	`(function () { return 42; })() == 42`
18	1	`1 == "1"`
19	1	`!(1 === "1")`
22	1	`eval("function range(begin, end) { for (let i = begin; i < end; ++i) { yield i; } } var ten_squares = [i * i for each (i in range(0, 10)) if (i % 2 == 0)]; ten_squares[1] == 4")`
20	1	`[42, 23].length == 2`
21	1	`[42,].length == 1`
23	1	`var a = 1; var b = 2; [a, b] = [b, a]; a == 2 && b == 1`
	2	`function f () { return [1, 2]; } var [a, b] = f(); a == 1 && b == 2`
29	1	`function f() { return typeof arguments != "undefined" && arguments; }; var a = f(); a && typeof a != "undefined"`
263	1	`function f() { return typeof arguments != "undefined" && Object.prototype.toString.call(arguments) == "[object Arguments]"; }; f()`
30	1	`function f() { return typeof arguments != "undefined" && arguments; }; var a = f(); a && typeof a != "undefined" && typeof a.callee == "function"`

32	1	`function f () { return typeof arguments != "undefined" && arguments; }; var a = f(42); a && typeof a != "undefined" && typeof a.length == "number" && a.length == 1`
33	1	`jsx.object.isNativeMethod(jsx.global, "Array")`
	2	`var a = new Array(); a.length == 0 && typeof a[0] == "undefined";`
	3	`var a = new Array(2, 1); a.length == 2 && a[0] == 2 && a[1] == 1;`
	4	`var a = new Array(2); a.length == 2 && typeof a[0] == "undefined" && typeof a[1] == "undefined";`
	5	`var error = jsx.tryThis("new Array(-1);", "e"); error && error.name == "RangeError"`
	6	`var error = jsx.tryThis("new Array(Math.pow(2, 32))", "e"); error && error.name == "RangeError"`
34	1	`jsx.object.isNativeMethod(jsx.global, "Array", "every")`
	2	`function f (e) { return e < 3; } Array.every(new Array(0, 1, 2), f)`
	3	`function f (e) { return e < 3; } !Array.every(new Array(0, 1, 3), f);`
35	1	`jsx.object.isNativeMethod(jsx.global, "Array", "isArray")`
	2	`var result = Array.isArray(1); typeof result == "boolean" && !result`
	3	`var result = Array.isArray(new Array()); typeof result == "boolean" && result`
36	1	`var a = new Array(); a[0] = 42; a.length == 1`
38	1	`!!jsx.object.getFeature(jsx.global, "Array", "prototype")`
39	1	`jsx.object.isNativeMethod(jsx.global, "Array", "prototype", "concat")`
	2	`var a = new Array("foo").concat(new Array("bar", "baz"), new Array("bla")); a[0] == "foo" && a[1] == "bar" && a[2] == "baz" && a[3] == "bla"`

```
3  var a = new Array("foo").concat("bar", "baz",
   "bla");
   a[0] == "foo" && a[1] == "bar" && a[2] == "baz"
   && a[3] == "bla"
4  var a = new Array("foo").concat("bar", new
   Array("baz", "bla"));
   a[0] == "foo" && a[1] == "bar" && a[2] == "baz"
   && a[3] == "bla"
5  var len = Array.prototype.concat.length;
   typeof len == "number" && len == 1
6  var o = new Object();
   o.concat = Array.prototype.concat;
   var a = o.concat("foo", new Array("bar", "baz"));
   a && typeof a.concat == "function" && a.length ==
   4 && a[0] == o & a[1] == "foo" && a[2] == "bar" &&
   a[3] == "baz"
```

40

```
1  jsx.object.getFeature(jsx.global, "Array",
   "prototype", "constructor")
2  Array.prototype.constructor == Array
```

41

```
1  jsx.object.isNativeMethod(jsx.global, "Array",
   "prototype", "every")
2  function f () { a.every(42); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Array", "prototype", "every"))
   {
   var a = new Array(1, 2, 3);
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
3  function f (e) { return e == "f"; };
   new Array("f", "f", "f").every(f);
4  function f (e, i, a) { return a[i] == "f"; };
   var o = new Object();
   o[0] = "f";
   o[1] = "f";
   o.length = 2;
   new Array().every(f, o);
5  var len = Array.prototype.every.length;
   typeof len == "number" && len == 1
```

```
6  function f (e, i, a) { return e < 42; }
   var o = new Object();
   o[0] = 1;
   o[1] = 2;
   o.length = 2;
   o.every = Array.prototype.every;
   var result = o.every(f);
   typeof result == "boolean" && result
```

```
42 1  jsx.object.isNativeMethod(jsx.global, "Array",
      "prototype", "filter")
   2  function f () { a.filter(42); }
      if (jsx.object.isNativeMethod(jsx.global,
      "Array", "prototype", "filter"))
      {
      var a = new Array(1, 2, 3);
      var result = jsx.tryThis(f, "e");
      result && result.name == "TypeError"
      }
   3  function f (e, i, a) { return e == "bar"; }
      new Array("foo", "bar").filter(f)[0] == "bar"
   4  function f (e, i, a) { return this[i] == "bar"; }
      var o = new Object();
      o[0] = "foo";
      o[1] = "bar";
      new Array("x", "y").filter(f, o)[0] == "y"
   5  var len = Array.prototype.filter.length;
      typeof len == "number" && len == 1
   6  function f (e, i, a) { return e == "bar"; }
      var o = new Object();
      o[0] = "foo";
      o[1] = "bar";
      o.length = 2;
      o.filter = Array.prototype.filter;
      var result = o.filter(f);
      result && result.length == 1 && result[0] == "bar"
```

```
43 1  jsx.object.isNativeMethod(jsx.global, "Array",
      "prototype", "forEach")
```

```
2  function f () { a.forEach(42); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Array", "prototype", "forEach"))
   {
   var a = new Array();
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
3  function f (e, i, a) { a[i] = e.length; }
   var a = new Array("foobar", "baz");
   a.forEach(f);
   a[0] == 6 && a[1] == 3
4  function f (e, i, a) { this[i] = e; }
   var o = new Object();
   var a = new Array("x", "y");
   a.forEach(f, o);
   o[0] == "x" && o[1] == "y"
5  function f (e, i, a) { a[i] = e.length; }
   var a = new Array("foobar", "baz");
   typeof a.forEach(f) == "undefined"
6  var len = Array.prototype.forEach.length;
   typeof len == "number" && len == 1
7  function f (e, i, a) { a[i] = e.length; }
   var o = new Object();
   o[0] = "foobar";
   o[1] = "baz";
   o.length = 2;
   o.forEach = Array.prototype.forEach;
   o.forEach(f);
   o[0] == 6 && o[1] == 3
```

44

```
1  jsx.object.isNativeMethod(jsx.global, "Array",
   "prototype", "indexOf")
2  var i = new Array().indexOf("foo");
   typeof i == "number" && i == -1
3  var i = new Array("2", 2).indexOf(2);
   typeof i == "number" && i == 1
4  var i = new Array("foo", "foo").indexOf("foo", 1);
   typeof i == "number" && i == 1
5  var i = new Array("foo", "foo").indexOf("foo",
   "1");
   typeof i == "number" && i == 1
6  var i = new Array("foo").indexOf("bar")
   typeof i == "number" && i == -1
```

```
 7  var i = new Array("foo", "bar").indexOf("foo", 1);
    typeof i == "number" && i == -1
 8  var i = new Array("foo", "bar").indexOf("foo",
    "1");
    typeof i == "number" && i == -1
 9  var i = new Array("foo", "foo").indexOf("foo",
    -2);
    typeof i == "number" && i == 0
10  var i = new Array("foo").indexOf("foo", 1);
    typeof i == "number" && i == -1
11  var len = Array.prototype.indexOf.length;
    typeof len == "number" && len == 1
12  var o = new Object();
    o.indexOf = Array.prototype.indexOf;
    o[0] = 42;
    o[1] = "42";
    o[2] = 42;
    o.length = 3;
    o.indexOf(42) == 0
```

```
45  1  jsx.object.isNativeMethod(jsx.global, "Array",
       "prototype", "join")
    2  var a = new Array("foo", "bar");
       a.join() == "foo,bar"
    3  var a = new Array("foo", "bar");
       a.join("|") == "foo|bar"
    4  var o = new Object(), o2 = new Object();
       function str() { return this.foo; }
       o.toString = str;
       o.foo = "bar";
       o2.toString = str;
       o2.foo = "baz";
       new Array(o, o2).join() == "bar,baz"
    5  var o = new Object(), o2 = new Object();
       function str() { return this.foo; }
       o.toString = str;
       o.foo = "bar";
       o2.toString = str;
       o2.foo = "baz";
       new Array(o, o2).join("|") == "bar|baz"
```

```
6  var a = new Array();
   a[0] = "foo";
   a[2] = "bar";
   delete a[1];
   a.join("") == "foobar";
7  var a = new Array();
   a[0] = "foo";
   a[1] = null;
   a[2] = "bar";
   a.join("") == "foobar";
8  var len = Array.prototype.join.length;
   typeof len == "number" && len == 1
9  var o = new Object();
   o.join = Array.prototype.join;
   o[0] = "foo";
   o[1] = "bar"
   o.length = 2;
   o.join("") == "foobar"
```

46

```
1  jsx.object.isNativeMethod(jsx.global, "Array",
   "prototype", "lastIndexOf")
2  var i = new Array().lastIndexOf("foo");
   typeof i == "number" && i == -1
3  var i = new Array("2", 2).lastIndexOf(2);
   typeof i == "number" && i == 1
4  var i = new Array("foo", "foo").lastIndexOf("foo",
   0);
   typeof i == "number" && i == 0
5  var i = new Array("foo", "foo").lastIndexOf("foo",
   "0");
   typeof i == "number" && i == 0
6  var i = new Array("foo").lastIndexOf("bar")
   typeof i == "number" && i == -1
7  var i = new Array("bar", "foo").lastIndexOf("foo",
   0);
   typeof i == "number" && i == -1
8  var i = new Array("bar", "foo").lastIndexOf("foo",
   "0");
   typeof i == "number" && i == -1
9  var i = new Array("foo", "foo").lastIndexOf("foo",
   -2);
   typeof i == "number" && i == 0
10 var i = new Array("foo").lastIndexOf("foo", 1);
   typeof i == "number" && i == 0
```

```
11  var len = Array.prototype.lastIndexOf.length;
    typeof len == "number" && len == 1
12  var o = new Object();
    o.lastIndexOf = Array.prototype.lastIndexOf;
    o[0] = 42;
    o[1] = "42";
    o[2] = 42;
    o.length = 3;
    o.lastIndexOf(42) == 2
```

```
47  1  var a = jsx.object.getFeature(jsx.global, "Array",
       "prototype");
       a && typeof a.length == "number"
    2  var a = jsx.object.getFeature(jsx.global, "Array",
       "prototype");
       typeof a.length == "number" && a.length == 0
```

```
48  1  jsx.object.isNativeMethod(jsx.global, "Array",
       "prototype", "map")
    2  function f (e) { return e; }
       jsx.tryThis("new Array().map()", f).name ==
       "TypeError"
    3  function f () { count++; }
       var a = new Array();
       a.length = 2;
       var count = 0;
       a.map(f);
       count == 0
    4  function f () {}
       var a = new Array();
       a.map(f).constructor == Array
    5  function f () {}
       var a = new Array("foo");
       a.length = 2;
       a.map(f).length == 2
    6  function f () { return "bar"; }
       new Array("foo").map(f)[0] == "bar"
```

```
 7  var success = false;
    var a = new Array("foo");
    function f (e, i, a2)
    {
    !success && e == "foo" && i == 0
    && a2 == a && (success = true);
    }
    a.map(f);
    success
 8  function f (e, i, a) { return this[i] + "bar"; }
    var o = new Object();
    o[0] = "foo";
    var a = new Array("baz");
    var mapped = new Array("foo").map(f, o);
    mapped.constructor == Array && mapped[0] ==
    "foobar"
 9  function f (e, i, a) { return this[i] + "bar"; }
    var o = new Object();
    o[0] = "foo";
    var a = new Array("baz");
    var mapped = new Array("foo").map(f, o);
    mapped.constructor == Array && mapped[0] ==
    "foobar" && o[0] == "foo"
10  var len = Array.prototype.map.length;
    typeof len == "number" && len == 1
11  function f (e, i, a) { return e + "bar"; }
    var o = new Object();
    o[0] = "foo";
    o.length = 1;
    o.map = Array.prototype.map;
    var mapped = o.map(f);
    mapped.constructor == Array && mapped[0] ==
    "foobar" && o[0] == "foo"
```

```
49  1  jsx.object.isNativeMethod(jsx.global, "Array",
       "prototype", "pop")
    2  var a = new Array();
       a.pop();
       a.length == 0
    3  typeof new Array().pop() == "undefined"
    4  var a = new Array("foo", "bar");
       var len = a.length;
       a.pop();
       len == 2 && a.length == 1
```

```
5  new Array("foo", "bar").pop() == "bar"
6  var o = new Object();
   o[0] = "foo";
   o[1] = "bar";
   o.length = 2;
   o.pop = Array.prototype.pop;
   var len = o.length;
   o.pop();
   len == 2 && o.length == 1 && typeof o[1] ==
   "undefined"
```

50

```
1  jsx.object.isNativeMethod(jsx.global, "Array",
   "prototype", "push")
2  var a = new Array("foo");
   a.push("bar", "baz");
   a[0] == "foo" && a[1] == "bar" && a[2] == "baz"
   && a.length == 3
3  new Array("foo").push("bar", "baz") == 3
4  var len = Array.prototype.push.length;
   typeof len == "number" && len == 1
5  var o = new Object();
   o.length = 0;
   o.push = Array.prototype.push;
   o.push(42);
   typeof o.length == "number" && o.length == 1 &&
   typeof o[0] == "number" && o[0] == 42
```

51

```
1  jsx.object.isNativeMethod(jsx.global, "Array",
   "prototype", "reduce")
2  if (jsx.object.isNativeMethod(jsx.global,
   "Array", "prototype", "reduce"))
   {
   var result = jsx.tryThis("new Array().reduce()",
   "e");
   result && result.name == "TypeError"
   }
3  function f2 (prev, e, i, a) { return prev + e; }
   function f () { new Array().reduce(f2); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Array", "prototype", "reduce"))
   {
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
```

```
4  var a = new Array();
   a[0] = 1;
   a[1] = 2;
   a[2] = 3;
   var success = false;
   function f (prev, e, i, a) { !success && prev
   == 1 && e == 2 && i == 1 && (success = true); }
   a.reduce(f);
   success
5  var a = new Array();
   a[0] = 1;
   a[1] = 2;
   a[2] = 3;
   function f (prev, e, i, a) { return prev + e; }
   a.reduce(f) == 6
6  var len = Array.prototype.reduce.length;
   typeof len == "number" && len == 1
7  var o = new Object();
   o[0] = 1;
   o[1] = 2;
   o[2] = 3;
   o.length = 3;
   o.reduce = Array.prototype.reduce;
   function f (prev, e, i, a) { return prev + e; }
   o.reduce(f) == 6
```

```
52 1  jsx.object.isNativeMethod(jsx.global, "Array",
      "prototype", "reduceRight")
   2  if (jsx.object.isNativeMethod(jsx.global,
      "Array", "prototype", "reduceRight"))
      {
      var result = jsx.tryThis("new
      Array().reduceRight()", "e");
      result && result.name == "TypeError"
      }
   3  function f2 (prev, e, i, a) { return prev + e; }
      function f () { new Array().reduceRight(f2); }
      if (jsx.object.isNativeMethod(jsx.global,
      "Array", "prototype", "reduceRight"))
      {
      var result = jsx.tryThis(f, "e");
      result && result.name == "TypeError"
      }
```

```
4  function f2 (prev, e, i, a) { return prev + e; }
   function f () { var a = new Array();
   a.length = 10; a.reduceRight(f2); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Array", "prototype", "reduceRight"))
   {
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
5  var a = new Array();
   a[0] = 1;
   a[1] = 2;
   a[2] = 3;
   var success = false;
   function f (prev, e, i, a) { !success && prev
   == 3 && e == 2 && i == 1 && (success = true); }
   a.reduceRight(f);
   success
6  var a = new Array();
   a[0] = 1;
   a[1] = 2;
   a[2] = 3;
   function f (prev, e, i, a) { return prev - e; }
   var result = a.reduceRight(f);
   typeof result == "number" && result == 0
7  var len = Array.prototype.reduceRight.length;
   typeof len == "number" && len == 1
8  var o = new Object();
   o[0] = 1;
   o[1] = 2;
   o[2] = 3;
   o.length = 3;
   o.reduceRight = Array.prototype.reduceRight;
   function f (prev, e, i, a) { return prev - e; }
   var result = o.reduceRight(f);
   typeof result == "number" && result == 0
```

```
53 1  jsx.object.isNativeMethod(jsx.global, "Array",
      "prototype", "reverse")
   2  var a = new Array("foo", "bar", "baz");
      a.reverse();
      a[0] == "baz" && a[1] == "bar" && a[2] == "foo"
```

```
3  var a = new Array("foo", "bar", "baz");
   var len = a.length;
   var reversed = a.reverse();
   reversed.constructor == Array && reversed.length
   == len && reversed[0] == "baz" && reversed[1] ==
   "bar" && reversed[2] == "foo"
4  var o = new Object();
   o[0] = "foo";
   o[1] = "bar";
   o[2] = "baz";
   o.length = 3;
   o.reverse = Array.prototype.reverse;
   var reversed = o.reverse();
   reversed && reversed.constructor == Object && o[0]
   == "baz" && o[1] == "bar" && o[2] == "foo"
```

```
54  1  jsx.object.isNativeMethod(jsx.global, "Array",
       "prototype", "shift")
    2  var a = new Array();
       var result = a.shift();
       a.length == 0 && typeof result == "undefined"
    3  var a = new Array("foo", "bar");
       var len = a.length;
       a.shift();
       len == 2 && a.length == 1 && typeof a[1] ==
       "undefined" && a[0] == "bar"
    4  new Array("foo", "bar").shift() == "foo"
    5  var o = new Object();
       o[0] = "foo";
       o[1] = "bar";
       o.length = 2;
       o.shift = Array.prototype.shift;
       var len = o.length;
       o.shift();
       len == 2 && o.length == 1 && typeof o[1] ==
       "undefined" && o[0] == "bar"
```

```
55  1  jsx.object.isNativeMethod(jsx.global, "Array",
       "prototype", "slice")
    2  var a = new Array("foo", "bar", "baz")
       var a2 = a.slice();
       a2.length == 3 && a2[0] == "foo" && a2[1] == "bar"
       && a2[2] == "baz"
```

```
3  var a = new Array("foo", "bar", "baz")
   var a2 = a.slice(1);
   a2.length == 2 && a2[0] == "bar" && a2[1] == "baz"
4  var a = new Array("foo", "bar", "baz")
   var a2 = a.slice(1, 2);
   a2.length == 1 && a2[0] == "bar"
5  var a = new Array("foo", "bar", "baz")
   var a2 = a.slice(-2);
   a2.length == 2 && a2[0] == "bar" && a2[1] == "baz"
6  var a = new Array("foo", "bar", "baz")
   var a2 = a.slice(1, -1);
   a2.length == 1 && a2[0] == "bar"
7  var len = Array.prototype.slice.length;
   typeof len == "number" && len == 2
8  var o = new Object();
   o[0] = "foo";
   o[1] = "bar";
   o[2] = "baz";
   o.length = 3;
   o.slice = Array.prototype.slice;
   var a2 = o.slice(1, 2);
   a2.length == 1 && a2[0] == "bar"
```

```
56  1  jsx.object.isNativeMethod(jsx.global, "Array",
       "prototype", "some")
    2  var result = jsx.tryThis("new Array().some()",
       "e");
       result && result.name == "TypeError"
    3  var success = false;
       var a = new Array("foo", "bar");
       function f (e, i, a2) { !success && e == "foo"
       && i == 0 && a2 == a && (success = true); }
       a.some(f);
       success
    4  var a = new Array("foo", "bar");
       function f (e, i, a2) { return e == "baz"; }
       var result = a.some(f);
       typeof result == "boolean" && !result
    5  var a = new Array("foo", "bar");
       function f (e, i, a2) { return e == "bar"; }
       var result = a.some(f);
       typeof result == "boolean" && result
```

```
6   var a = new Array("he", "yo");
    var o = new Object();
    o[0] = "foo";
    o[1] = "bar";
    o.length = 2;
    function f (e, i, a) { return this[i] == "baz"; }
    var result = a.some(f, o);
    typeof result == "boolean" && !result
7   var a = new Array("baz", "bla");
    var o = new Object();
    o[0] = "foo";
    o[1] = "bar";
    o.length = 2;
    function f (e, i, a) { return this[i] == "bar"; }
    var result = a.some(f, o);
    typeof result == "boolean" && result
8   var a = new Array("foo", "bar");
    function f (e, i, a2) { return e == "bar"; }
    a.some(f);
    a.constructor == Array && a.length == 2 && a[0]
    == "foo" && a[1] == "bar"
9   var a = new Array("baz", "bla");
    var o = new Object();
    o[0] = "foo";
    o[1] = "bar";
    o.length = 2;
    function f (e, i, a) { return this[i] == "bar"; }
    a.some(f, o);
    o.constructor == Object && o.length == 2 && o[0]
    == "foo" && o[1] == "bar"
10  var len = Array.prototype.some.length;
    typeof len == "number" && len == 1
11  var o = new Object();
    o[0] = "foo";
    o[1] = "bar";
    o.length = 2;
    o.some = Array.prototype.some;
    function f (e, i, a2) { return e == "bar"; }
    var result = o.some(f);
    typeof result == "boolean" && result
```

```
57  1   jsx.object.isNativeMethod(jsx.global, "Array",
        "prototype", "sort")
```

```
2  var a = new Array("foo", "bar");
   a.sort();
   a[0] == "bar" && a[1] == "foo"
3  var a = new Array("bar", "foo");
   function f (a, b) { return a > b ? -1 : (a < b ? 1
   : 0); }
   a.sort(f);
   a[0] == "foo" && a[1] == "bar"
4  var a = new Array("foo", "bar");
   var result = a.sort();
   result.constructor == Array && result.length ==
   2 && result[0] == "bar" && result[1] == "foo"
5  var o = new Object();
   o[0] = "bar";
   o[1] = "foo";
   o.length = 2;
   o.sort = Array.prototype.sort;
   function f (a, b) { return a > b ? -1 : (a < b ? 1
   : 0); }
   o.sort(f);
   o[0] == "foo" && o[1] == "bar"
```

```
58 1  jsx.object.isNativeMethod(jsx.global, "Array",
      "prototype", "splice")
   2  var a = new Array("foo", "bar", "baz");
      a.splice(1, 1);
      a.length == 2 && a[0] == "foo" && a[1] == "baz"
   3  var a = new Array("foo", "bar", "baz");
      a.splice(1, 0, "bla");
      a.length == 4 && a[0] == "foo" && a[1] == "bla"
      && a[2] == "bar" && a[3] == "baz"
   4  var a = new Array("foo", "bar", "baz");
      a.splice(-2, 1);
      a.length == 2 && a[0] == "foo" && a[1] == "baz"
   5  var len = Array.prototype.splice.length;
      typeof len == "number" && len == 2
```

	6	`var o = new Object();`
		`o[0] = "foo";`
		`o[1] = "bar";`
		`o[2] = "baz";`
		`o.length = 3;`
		`o.splice = Array.prototype.splice;`
		`var result = o.splice(1, 1, "bla");`
		`result && result.constructor == Array &&`
		`result.length == 1 && result[0] == "bar" &&`
		`o.length == 3 && o[0] == "foo" && o[1] == "bla"`
		`&& o[2] == "baz"`

59	1	`jsx.object.isNativeMethod(jsx.global, "Array",` `"prototype", "toSource")`
	2	`new Array("foo").toSource() == '["foo"]'`
60	1	`jsx.object.isNativeMethod(jsx.global, "Array",` `"prototype", "toString")`
	2	`new Array("foo", "bar").toString() == "foo,bar"`

	3	`var o = new Object(), o2 = new Object();`
		`function str() { return this.foo; }`
		`o.toString = str;`
		`o.foo = "bar";`
		`o2.toString = str;`
		`o2.foo = "baz";`
		`new Array(o, o2).toString() == "bar,baz"`

	4	`var o = new Object();`
		`o[0] = new Object();`
		`o[1] = new Object();`
		`o.length = 2;`
		`o.toString = Array.prototype.toString;`
		`var result = o.toString();`
		`result && result == (new Object()).toString()`

	5	`var o = new Object();`
		`o[0] = new Object();`
		`o[1] = new Object();`
		`o.length = 2;`
		`function f () { return "42"; }`
		`o.join = f;`
		`o.toString = Array.prototype.toString;`
		`var result = o.toString();`
		`result && result == "42"`

61	1	`jsx.object.isNativeMethod(jsx.global, "Array",` `"prototype", "unshift")`

```
2  var a = new Array();
   a.unshift();
   a.length == 0
3  var a = new Array("foo");
   a.unshift("bar");
   a.length == 2 && a[0] == "bar" && a[1] == "foo"
4  var a = new Array("foo");
   var result = a.unshift("bar");
   result == 2
5  var o = new Object();
   o[0] = "bar";
   o.length = 1;
   o.unshift = Array.prototype.unshift;
   var len = o.length;
   o.unshift("foo");
   len == 1 && o.length == 2 && typeof o[0] ==
   "string" && o[0] == "foo" && typeof o[1] ==
   "string" && o[1] == "bar"
```

```
37  1  jsx.object.isNativeMethod(jsx.global, "Array",
       "some")
    2  if (jsx.object.isNativeMethod(jsx.global, "Array",
       "some"))
       {
       var result = jsx.tryThis("Array.some(new
       Array())", "e");
       result && result.name == "TypeError"
       }
    3  var success = false;
       var a = new Array("foo", "bar");
       function f (e, i, a2) { !success && e == "foo"
       && i == 0 && a2 == a && (success = true); }
       Array.some(a, f);
       success
    4  var a = new Array("foo", "bar");
       function f (e, i, a2) { return e == "baz"; }
       var result = Array.some(a, f);
       typeof result == "boolean" && !result
    5  var a = new Array("foo", "bar");
       function f (e, i, a2) { return e == "bar"; }
       var result = Array.some(a, f);
       typeof result == "boolean" && result
```

```
6  var a = new Array("he", "yo");
   var o = new Object();
   o[0] = "foo";
   o[1] = "bar";
   o.length = 2;
   function f (e, i, a) { return this[i] == "baz"; }
   var result = Array.some(o, f, o);
   typeof result == "boolean" && !result
7  var a = new Array("baz", "bla");
   var o = new Object();
   o[0] = "foo";
   o[1] = "bar";
   o.length = 2;
   function f (e, i, a) { return this[i] == "bar"; }
   var result = Array.some(a, f, o);
   typeof result == "boolean" && result
8  var a = new Array("foo", "bar");
   function f (e, i, a2) { return e == "bar"; }
   Array.some(a, f);
   a.constructor == Array && a.length == 2 && a[0]
   == "foo" && a[1] == "bar"
9  var a = new Array("baz", "bla");
   var o = new Object();
   o[0] = "foo";
   o[1] = "bar";
   o.length = 2;
   function f (e, i, a) { return this[i] == "bar"; }
   Array.some(a, f, o);
   o.constructor == Object && o.length == 2 && o[0]
   == "foo" && o[1] == "bar"
10 function f2 (e) { return true; }
   function f () { Array.some(void(0), f2); }
   if (jsx.object.isNativeMethod(jsx.global, "Array",
   "some"))
   {
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
```

62 1 `var b : boolean;`

63 1 `jsx.object.isNativeMethod(jsx.global, "Boolean")`

```
2  var b = Boolean(void(0));
   typeof b == "boolean" && !b
```

```
3  var b = Boolean(null);
   typeof b == "boolean" && !b
4  var b = Boolean(false);
   typeof b == "boolean" && !b
5  var b = Boolean(true);
   typeof b == "boolean" && b
6  var b = Boolean(0);
   typeof b == "boolean" && !b
7  var b = Boolean(0);
   typeof b == "boolean" && !b
8  var b = Boolean(NaN);
   typeof b == "boolean" && !b
9  var b = Boolean("");
   typeof b == "boolean" && !b
10 var b = Boolean("0");
   typeof b == "boolean" && b
11 var b = Boolean(new Object());
   typeof b == "boolean" && b
```

```
64  1  !!jsx.object.getFeature(jsx.global, "Boolean",
       "prototype")
```

```
65  1  jsx.object.isNativeMethod(jsx.global, "Boolean",
       "prototype", "toSource")
    2  (false).toSource() == "false"
    3  (true).toSource() == "true"
```

```
66  1  jsx.object.isNativeMethod(jsx.global, "Boolean",
       "prototype", "toString")
    2  (false).toString() == "false"
    3  (true).toString() == "true"
    4  function f () { o.toString(); }
       if (jsx.object.isNativeMethod(jsx.global,
       "Boolean", "prototype", "toString"))
       {
       var o = new Object();
       o.toString = Boolean.prototype.toString;
       var result = jsx.tryThis(f, "e");
       result && result.name == "TypeError"
       }
```

```
67  1  jsx.object.isNativeMethod(jsx.global, "Boolean",
       "prototype", "valueOf")
    2  var result = (false).valueOf();
       typeof result == "boolean" && !result
```

```
3  var result = (true).valueOf();
   typeof result == "boolean" && result
4  function f () { o.valueOf(); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Boolean", "prototype", "valueOf"))
   {
   var o = new Object();
   o.valueOf = Boolean.prototype.valueOf;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
```

68
```
1  var b : byte;
```

70
```
1  var c : char;
```

72
```
1  const answer = 42;
   answer == 42
2  const answer = 42;
   answer = 23;
   answer == 42
```

286
```
1  jsx.object.isNativeMethod(Date, "now")
2  typeof Date.now() == "number"
```

73
```
1  !!jsx.object.getFeature(jsx.global, "Date",
   "prototype")
```

74
```
1  jsx.object.isNativeMethod(jsx.global, "Date",
   "prototype", "getFullYear")
2  var result = new Date(2012, 0, 1).getFullYear();
   typeof result == "number" && result == 2012
3  var result = new Date(NaN).getFullYear();
   typeof result == "number" && isNaN(result)
```

75
```
1  jsx.object.isNativeMethod(jsx.global, "Date",
   "prototype", "getMilliseconds")
2  var result = new Date(2012, 0, 1,
   0, 0, 0, 42).getMilliseconds();
   typeof result == "number" && result == 42
3  var result = new Date(NaN).getMilliseconds();
   typeof result == "number" && isNaN(result)
```

76
```
1  jsx.object.isNativeMethod(jsx.global, "Date",
   "prototype", "getUTCDate")
2  typeof new Date().getUTCDate() == "number"
```

```
      3  var result = new Date(NaN).getUTCDate();
         typeof result == "number" && isNaN(result)
```

```
77    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "getUTCDay")
      2  typeof new Date().getUTCDay() == "number"
      3  var result = new Date(NaN).getUTCDay();
         typeof result == "number" && isNaN(result)
```

```
78    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "getUTCFullYear")
      2  var result = new Date(2012, 6,
         1).getUTCFullYear();
         typeof result == "number" && result == 2012
      3  var result = new Date(NaN).getUTCFullYear();
         typeof result == "number" && isNaN(result)
```

```
79    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "getUTCHours")
      2  typeof new Date().getUTCHours() == "number"
      3  var result = new Date(NaN).getUTCHours();
         typeof result == "number" && isNaN(result)
```

```
80    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "getUTCMilliseconds")
      2  var result = new Date(2012, 0, 1,
         0, 0, 0, 42).getUTCMilliseconds();
         typeof result == "number" && result == 42
      3  var result = new Date(NaN).getUTCMilliseconds();
         typeof result == "number" && isNaN(result)
```

```
81    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "getUTCMinutes")
      2  typeof new Date().getUTCMinutes() == "number"
      3  var result = new Date(NaN).getUTCMinutes();
         typeof result == "number" && isNaN(result)
```

```
82    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "getUTCMonth")
      2  typeof new Date(2012, 0, 1).getUTCMonth() ==
         "number"
      3  var result = new Date(NaN).getUTCMonth();
         typeof result == "number" && isNaN(result)
```

```
83    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "getUTCSeconds")
```

```
      2  var result = new Date(2012, 0,
         1, 0, 0, 42).getUTCSeconds();
         typeof result == "number" && result == 42
      3  var result = new Date(NaN).getUTCSeconds();
         typeof result == "number" && isNaN(result)
```

```
84    1  jsx.object.isMethod(jsx.global, "Date",
         "prototype", "getVarDate")
```

```
85    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "setFullYear")
      2  var d = new Date(2011, 0, 1);
         d.setFullYear(2012);
         d.getFullYear() == 2012
      3  var d = new Date(2011, 0, 1);
         d.setFullYear(2012, 5);
         d.getFullYear() == 2012 && d.getMonth() == 5
      4  var d = new Date(2011, 0, 1);
         d.setFullYear(2012, 5, 11);
         d.getFullYear() == 2012 && d.getMonth() == 5 &&
         d.getDate() == 11
      5  var result = new Date(2011, 0,
         1).setFullYear(2012);
         typeof result == "number" && !isNaN(result)
      6  var len = Date.prototype.setFullYear.length;
         typeof len == "number" && len == 3
```

```
86    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "setMilliseconds")
      2  var d = new Date(2012, 0, 1, 0, 0, 0, 0);
         d.setMilliseconds(42);
         d.getMilliseconds() == 42
      3  var result = new Date(2012, 0, 1,
         0, 0, 0, 0).setMilliseconds(42);
         typeof result == "number" && !isNaN(result);
```

```
87    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "setUTCDate")
      2  var d = new Date(2012, 5, 1);
         d.setUTCDate(11);
         d.getUTCDate() == 11
      3  var result = new Date(2012, 5, 1).setUTCDate(11);
         typeof result == "number" && !isNaN(result)
```

```
89    1  jsx.object.isNativeMethod(jsx.global, "Date",
         "prototype", "setUTCFullYear")
```

```
2  var d = new Date(2011, 0, 2);
   d.setUTCFullYear(2012);
   d.getUTCFullYear() == 2012
3  var d = new Date(2011, 0, 2);
   d.setUTCFullYear(2012, 5);
   d.getUTCFullYear() == 2012 && d.getUTCMonth() == 5
4  var d = new Date(2011, 0, 2);
   d.setUTCFullYear(2012, 5, 11);
   d.getUTCFullYear() == 2012 && d.getUTCMonth() == 5
   && d.getUTCDate() == 11
5  var result = new Date(2011, 0,
   2).setUTCFullYear(2012);
   typeof result == "number" && !isNaN(result)
6  var len = Date.prototype.setUTCFullYear.length;
   typeof len == "number" && len == 3
```

90

```
1  jsx.object.isNativeMethod(jsx.global, "Date",
   "prototype", "setUTCHours")
2  var d = new Date(2011, 0, 2, 0, 0, 0, 0);
   d.setUTCHours(11);
   d.getUTCHours() == 11
3  var d = new Date(2011, 0, 2, 0, 0, 0, 0);
   d.setUTCHours(11, 11);
   d.getUTCHours() == 11 && d.getUTCMinutes() == 11
4  var d = new Date(2011, 0, 2, 0, 0, 0, 0);
   d.setUTCHours(11, 11, 11);
   d.getUTCHours() == 11 && d.getUTCMinutes() == 11
   && d.getUTCSeconds() == 11
5  var d = new Date(2011, 0, 2, 0, 0, 0, 0);
   d.setUTCHours(11, 11, 11, 11);
   d.getUTCHours() == 11 && d.getUTCMinutes()
   == 11 && d.getUTCSeconds() == 11 &&
   d.getUTCMilliseconds() == 11
6  var result = new Date(2011, 0,
   2, 0, 0, 0, 0).setUTCHours(11);
   typeof result == "number" && !isNaN(result)
7  var len = Date.prototype.setUTCHours.length;
   typeof len == "number" && len == 4
```

91

```
1  jsx.object.isNativeMethod(jsx.global, "Date",
   "prototype", "setUTCMilliseconds")
2  var d = new Date(2012, 0, 1, 0, 0, 0, 0);
   d.setUTCMilliseconds(42);
   d.getUTCMilliseconds() == 42
```

	3	`var result = new Date(2012, 0, 1, 0, 0, 0, 0).setUTCMilliseconds(42); typeof result == "number" && !isNaN(result);`
92	1	`jsx.object.isNativeMethod(jsx.global, "Date", "prototype", "setUTCMinutes")`
	2	`var d = new Date(2011, 0, 2, 0, 0, 0, 0); d.setUTCMinutes(11); d.getUTCMinutes() == 11`
	3	`var d = new Date(2011, 0, 2, 0, 0, 0, 0); d.setUTCMinutes(11, 11); d.getUTCMinutes() == 11 && d.getUTCSeconds() == 11`
	4	`var d = new Date(2011, 0, 2, 0, 0, 0, 0); d.setUTCMinutes(11, 11, 11); d.getUTCMinutes() == 11 && d.getUTCSeconds() == 11 && d.getUTCMilliseconds() == 11`
	5	`var result = new Date(2011, 0, 2, 0, 0, 0, 0).setUTCMinutes(11); typeof result == "number" && !isNaN(result)`
	6	`var len = Date.prototype.setUTCMinutes.length; typeof len == "number" && len == 3`
93	1	`jsx.object.isNativeMethod(jsx.global, "Date", "prototype", "setUTCMonth")`
	2	`var d = new Date(2011, 0, 2); d.setUTCMonth(5); d.getUTCMonth() == 5`
	3	`var d = new Date(2011, 0, 2); d.setUTCMonth(5, 11); d.getUTCMonth() == 5 && d.getUTCDate() == 11`
	4	`var result = new Date(2011, 0, 2).setUTCMonth(5); typeof result == "number" && !isNaN(result)`
	5	`var len = Date.prototype.setUTCMonth.length; typeof len == "number" && len == 2`
94	1	`jsx.object.isNativeMethod(jsx.global, "Date", "prototype", "setUTCSeconds")`
	2	`var d = new Date(2011, 0, 2, 0, 0, 0, 0); d.setUTCSeconds(11); d.getUTCSeconds() == 11`
	3	`var d = new Date(2011, 0, 2, 0, 0, 0, 0); d.setUTCSeconds(11, 11); d.getUTCSeconds() == 11 && d.getUTCMilliseconds() == 11`

```
4  var result = new Date(2011, 0, 2,
   0, 0, 0, 0).setUTCSeconds(11);
   typeof result == "number" && !isNaN(result)
5  var len = Date.prototype.setUTCSeconds.length;
   typeof len == "number" && len == 2
```

```
95 1  jsx.object.isNativeMethod(jsx.global, "Date",
      "prototype", "toDateString")
   2  typeof new Date().toDateString() == "string"
```

```
96 1  jsx.object.isNativeMethod(jsx.global, "Date",
      "prototype", "toGMTString")
   2  typeof new Date().toGMTString() == "string"
```

```
97 1  jsx.object.isNativeMethod(jsx.global, "Date",
      "prototype", "toISOString")
   2  function f () { return d.toISOString(); }
      if (jsx.object.isNativeMethod(jsx.global,
      "Date", "prototype", "toISOString"))
      {
      var d = new Date();
      d.setTime(NaN);
      var result = jsx.tryThis(f, "e");
      result && result.name == "RangeError"
      }
   3  typeof new Date().toISOString() == "string"
   4  var d = new Date();
      d.setUTCFullYear(2012, 11, 11);
      d.setUTCHours(12, 12, 12, 999);
      d.toISOString() == d.getUTCFullYear() + "-" +
      (d.getUTCMonth() + 1) + "-" + d.getUTCDate() + "T"
      + d.getUTCHours() + ":" + d.getUTCMinutes() + ":"
      + d.getUTCSeconds() + "." + d.getUTCMilliseconds()
      + "Z"
```

```
98 1  jsx.object.isNativeMethod(jsx.global, "Date",
      "prototype", "toJSON")
   2  var d = new Date(NaN);
      var result = d.toJSON();
      typeof result == "object" && result == null
   3  typeof new Date().toJSON() == "string"
```

	4	`var d = new Date();`
		`d.setUTCFullYear(2012, 11, 11);`
		`d.setUTCHours(12, 12, 12, 999);`
		`d.toJSON(42) == d.getUTCFullYear() + "-" +`
		`(d.getUTCMonth() + 1) + "-" + d.getUTCDate() + "T"`
		`+ d.getUTCHours() + ":" + d.getUTCMinutes() + ":"`
		`+ d.getUTCSeconds() + "." + d.getUTCMilliseconds()`
		`+ "Z"`

	5	`function f () { return 0; }`
		`function f2 () { return "[0]"; }`
		`var o = new Object();`
		`o.valueOf = f;`
		`o.toISOString = f2;`
		`o.toJSON = Date.prototype.toJSON;`
		`var result = o.toJSON();`
		`typeof result == "string" && result == "[0]"`

	6	`function f () { return 0; }`
		`function f2 () { return o.toJSON(); }`
		`if (jsx.object.isNativeMethod(jsx.global,`
		`"Date", "prototype", "toJSON"))`
		`{`
		`var o = new Object();`
		`o.valueOf = f;`
		`o.toJSON = Date.prototype.toJSON;`
		`var result = jsx.tryThis(f2, "e");`
		`result && result.name == "TypeError"`
		`}`

99	1	`jsx.object.isNativeMethod(jsx.global, "Date",` `"prototype", "toLocaleDateString")`
	2	`typeof new Date().toLocaleDateString() == "string"`
100	1	`jsx.object.isNativeMethod(jsx.global, "Date",` `"prototype", "toLocaleFormat")`
	2	`typeof new Date().toLocaleFormat('%A, %B %e, %Y')` `== "string"`
101	1	`jsx.object.isNativeMethod(jsx.global, "Date",` `"prototype", "toLocaleString")`
	2	`typeof new Date().toLocaleString() == "string"`
102	1	`jsx.object.isNativeMethod(jsx.global, "Date",` `"prototype", "toLocaleTimeString")`
	2	`typeof new Date().toLocaleTimeString() == "string"`

103	1	`jsx.object.isNativeMethod(jsx.global, "Date", "prototype", "toSource")`
	2	`typeof new Date().toSource() == "string"`
104	1	`jsx.object.isNativeMethod(jsx.global, "Date", "prototype", "toString")`
	2	`typeof new Date().toString() == "string"`

	3	`var d = new Date();`
		`d.setMilliseconds(0);`
		`Date.parse(d.toString()) == d.valueOf()`

105	1	`jsx.object.isNativeMethod(jsx.global, "Date", "prototype", "toTimeString")`
	2	`typeof new Date().toTimeString() == "string"`
106	1	`jsx.object.isNativeMethod(jsx.global, "Date", "prototype", "toUTCString")`
	2	`typeof new Date().toUTCString() == "string"`
109	1	`var d : decimal;`
110	1	`jsx.object.isNativeMethod(jsx.global, "decodeURI")`
	2	`decodeURI("%20;/?:@&=+$,%7E%E2%82%AC") == ";/?:@&=+$,~€"`

	3	`function f () { return decodeURI("%20%AC"); }`
		`if (jsx.object.isNativeMethod(jsx.global, "decodeURI"))`
		`{`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "URIError"`
		`}`

111	1	`jsx.object.isNativeMethod(jsx.global, "decodeURIComponent")`
	2	`decodeURIComponent( "%3B%2F%3F%3A%40%26%3D%2B%24%2C%E2%82%AC") == ";/?:@&=+$,€"`

	3	`function f () { return decodeURIComponent("%20%AC");`
		`}`
		`if (jsx.object.isNativeMethod(jsx.global, "decodeURIComponent"))`
		`{`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "URIError"`
		`}`

112
```
1 delete 1
2 var o = new Object();
  o.foo = "bar";
  var result = delete o.foo;
  typeof o.foo == "undefined" && result
3 var o = new Object();
  delete o.foo
4 !delete "".length
```

114
```
1 var d : double;
```

113
```
1 do true; while (false);
```

115
```
1 jsx.object.isNativeMethod(jsx.global, "encodeURI")
2 encodeURI(" ;/?:@&=+$,~€") ==
  "%20;/?:@&=+$,~%E2%82%AC"
```

116
```
1 jsx.object.isNativeMethod(jsx.global,
  "encodeURIComponent")
2 encodeURIComponent(";/?:@&=+$,€") ==
  "%3B%2F%3F%3A%40%26%3D%2B%24%2C%E2%82%AC"
```

118
```
1 jsx.object.isMethod(jsx.global, "Enumerator")
```

119
```
1 jsx.object.isNativeMethod(jsx.global, "Error")
2 typeof new Error().message == "string"
3 var m = new Error(42).message;
  typeof m == "string" && m == "42"
```

120
```
1 typeof new Error().description != "undefined"
```

123
```
1 typeof new Error().number == "number"
```

121
```
1 typeof Error.prototype.message == "string"
```

122
```
1 Error.prototype.name == "Error"
```

124
```
1 typeof new Error().stack == "string"
```

127
```
1 var f : float;
```

128
```
1 var o = new Object();
  o.foo = "bar";
  var i;
  for each (i in o)
  {
  i == "bar"
  }
```

```
2  var o = new Object();
   o.foo = "bar";
   for each (var i in o)
   {
   i == "bar"
   }
3  var a = new Array();
   a[0] = "foo";
   var i;
   for each (i in a)
   {
   i == "foo"
   }
4  var a = new Array();
   a[0] = "foo";
   for each (var i in a)
   {
   i == "foo"
   }
```

```
129  1  jsx.object.isNativeMethod(jsx.global, "Function")
     2  var f = new Function();
        !f()
     3  function f () { new Function("{"); }
        if (jsx.object.isNativeMethod(jsx.global,
        "Function"))
        {
        var result = jsx.tryThis(f, "e");
        result && result.name == "SyntaxError"
        }
     4  function f () { new Function("{", ""); }
        if (jsx.object.isNativeMethod(jsx.global,
        "Function"))
        {
        var result = jsx.tryThis(f, "e");
        result && result.name == "SyntaxError"
        }
     5  var f = new Function("return true;");
        f()
     6  var f = new Function("foo", "return foo;");
        f(true)
     7  var f = new Function("foo, bar", "return foo &&
        bar;");
        f(true, true)
```

	8	`var f = new Function("foo", "bar", "return foo && bar;");` `f(true, true)`

139	1	`function f () { return f.arguments; }`
		`var result = f(42);`
		`result && typeof result[0] == "number" && result[0] == 42`

140	1	`function f () { return f.arguments.callee == f; }` `f()`
141	1	`function f () { return f.arguments.length; }` `f(42)`
137	1	`function f (x, y) { return f.arity == 2; }` `f()`

143	1	`function f1 () { return f1.caller; };`
		`function f2() { return f1(); };`
		`f2() == f2`

144	1	`function f (x, y) { return f.length == 2; }` `f()`
264	1	`function f () {}` `!!jsx.object.getFeature(f, "prototype")`
136	1	`!!jsx.object.getFeature(jsx.global, "Function", "prototype")`
	2	`!Function.prototype()`
138	1	`jsx.object.isNativeMethod(jsx.global, "Function", "prototype", "apply")`

	2	`function f () { o.apply(); }`
		`if (jsx.object.isNativeMethod(jsx.global, "Function", "prototype", "apply"))`
		`{`
		`var o = new Object();`
		`o.apply = Function.prototype.apply;`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`
		`}`

	3	`function f () { return this == jsx.global; }` `f.apply(null)`
	4	`function f () { return this == jsx.global; }` `f.apply()`

```
5   var o = new Object();
    function f () { return this == o; }
    f.apply(o)
6   function f () { return arguments.length == 0; }
    f.apply(null, null)
7   function f () { return arguments.length == 0; }
    f.apply(null)
8   function f () { return true; }
    function f2 () { f.apply(null, 1); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Function", "prototype", "apply"))
    {
    var result = jsx.tryThis(f2, "e");
    result && result.name == "TypeError"
    }
9   function f (x, y) { return x && y; }
    function f2 (x, y) { return f.apply(null,
    arguments); }
    f2(true, true)
10  function f (x, y) { return x && y; }
    f.apply(null, new Array(true, true))
11  var len = Function.prototype.apply.length;
    typeof len == "number" && len == 2
```

```
31  1   function f1() { return typeof arguments !=
        "undefined" && arguments.caller; }; function f2()
        { return f1(); }; f2() == f2
```

```
287 1   jsx.object.isNativeMethod(jsx.global, "Function",
        "prototype", "bind")
    2   function f () { var o = new Object(); o.bind
        = Function.prototype.bind; o.bind(o); }
        if (jsx.object.isNativeMethod(jsx.global,
        "Function", "prototype", "bind"))
        {
        var result = jsx.tryThis(f, "e");
        result && result.name == "TypeError"
        }
    3   function f (a) { return a; }
        f.bind(null, 42)(23) == 42
    4   function f (a, b) { return b; }
        f.bind(null, 23)(42) == 42
```

```
142 1   jsx.object.isNativeMethod(jsx.global, "Function",
        "prototype", "call")
```

	2	`function f () { o.call(); }`
		`if (jsx.object.isNativeMethod(jsx.global,`
		`"Function", "prototype", "call"))`
		`{`
		`var o = new Object();`
		`o.call = Function.prototype.call;`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`
		`}`

	3	`function f () { return this == jsx.global; }` `f.call(null)`
	4	`function f () { return this == jsx.global; }` `f.call()`

	5	`var o = new Object();`
		`function f () { return this == o; }`
		`f.call(o)`

	6	`function f () { return arguments.length == 0; }` `f.apply(null)`
	7	`function f (x) { return x; }` `f.call(null, true)`
	8	`var len = Function.prototype.call.length;` `typeof len == "number" && len == 1`
146	1	`jsx.object.isNativeMethod(jsx.global, "Function",` `"prototype", "toSource")`
	2	`function f () {}` `f.toSource() == "function f() {}"`

147	1	`eval("function generator() { yield true; } var`
		`gen = generator(); jsx.object.isNativeMethod(gen,`
		`'close');")`

148	1	`eval("function generator() { yield true; } var`
		`gen = generator(); jsx.object.isNativeMethod(gen,`
		`'next');")`

149	1	`eval("function generator() { yield true; } var`
		`gen = generator(); jsx.object.isNativeMethod(gen,`
		`'send');")`

150	1	`eval("function generator() { yield true; } var`
		`gen = generator(); jsx.object.isNativeMethod(gen,`
		`'throw');")`

151	1	`jsx.object.isMethod(jsx.global, "GetObject")`

152	1	`typeof jsx.global == "object" && jsx.global != null`
17	1	`var foo : Object = new Object();` `foo`

135	1	`function f() {};`
		`if (true) { function f() { return 42; }; }`
		`f() == 42`

157	1	`typeof Infinity == "number"`
	2	`Infinity > 0`
158	1	`new Object() instanceof Object`

	2	`function f () { new Object() instanceof 42; }`
		`function f2 (e) { return e; }`
		`jsx.tryThis(f, f2).name == "TypeError"`

	3	`function f () { new Object() instanceof Math; }`
		`function f2 (e) { return e; }`
		`jsx.tryThis(f, f2).name == "TypeError"`

159	1	`var i : int;`
162	1	`jsx.object.isNativeMethod(jsx.global, "isFinite")`
	2	`var result = isFinite(NaN)` `typeof result == "boolean" && !result`
	3	`var result = isFinite(Infinity);` `typeof result == "boolean" && !result`
	4	`var result = isFinite(-Infinity);` `typeof result == "boolean" && !result`
	5	`var result = isFinite(42);` `typeof result == "boolean" && result`
163	1	`jsx.object.isNativeMethod(jsx.global, "Iterator")`

	2	`var lang = { name: 'JavaScript', birthYear: 1995 };`
		`var it = Iterator(lang);`
		`var pair = it.next();`
		`pair[0] == "name" && pair[1] == "JavaScript"`

	3	`var lang = { name: 'JavaScript', birthYear: 1995 };`
		`var it = Iterator(lang);`
		`var pair = it.next();`
		`pair = it.next();`
		`pair[0] == "birthYear" && pair[1] == 1995`

```
4  var lang = { name: 'JavaScript', birthYear: 1995
   };
   var it = Iterator(lang);
   var pair = it.next();
   pair = it.next();
   function f () { pair = it.next(); }
   function f2 (e) { return e; }
   jsx.tryThis(f, f2) == StopIteration
```

```
164  1  jsx.object.isNativeMethod(jsx.global, "JSON",
        "parse")
     2  function f () { JSON.parse(); }
        if (jsx.object.isNativeMethod(jsx.global, "JSON",
        "parse"))
        {
        var result = jsx.tryThis(f, "e");
        result && result.name == "SyntaxError"
        }
     3  function f () { JSON.parse(""); }
        if (jsx.object.isNativeMethod(jsx.global, "JSON",
        "parse"))
        {
        var result = jsx.tryThis(f, "e");
        result && result.name == "SyntaxError"
        }
     4  var result = JSON.parse('{"answer": 42}');
        result = result && result["answer"];
        typeof result == "number" && result == 42
     5  function reviver (name, val) {
        return (name != "") ? void 0 : val; }
        var result = JSON.parse("[23]", reviver);
        result && result.length == 1 && typeof result[0]
        == "undefined"
     6  function reviver (name, val) {
        return (name != "") ? 42 : val; }
        var result = JSON.parse("[23]", reviver);
        result && result.length == 1 && typeof result[0]
        == "number" && result[0] == 42
     7  function reviver (name, val) { return 42; }
        var result = JSON.parse('[23]', reviver);
        result && typeof result == "number" && result ==
        42
```

	8	`function reviver (name, val) { return (name != "") ? void 0 : val; } var result = JSON.parse('{"answer": 42}', reviver); result && typeof result["answer"] == "undefined"`
	9	`function reviver (name, val) { return (name != "") ? 42 : val; } var result = JSON.parse('{"answer": 23}', reviver); result && typeof result["answer"] == "number" && result["answer"] == 42`
	10	`function reviver (name, val) { return 42; } var result = JSON.parse('{"answer": 23}', reviver); result && typeof result == "number" && result == 42`
165	1	`jsx.object.isNativeMethod(jsx.global, "JSON", "stringify")`
	2	`JSON.stringify(new Object()) == "{}"`
	3	`JSON.stringify(new Array()) == "[]"`
16	1	`foo: true`
166	1	`eval("var x = 5; var y = 0; let (x = x+10, y = 12) {}; x + y == 5;")`
167	1	`eval("var x = 5; var y = 0; let(x = x + 10, y = 12) x + y; x + y == 5")`
168	1	`eval("var i; if (true) { let answer = 42; i = answer * 1; } typeof i == 'number' && i == 42")`
169	1	`var lng : long;`
170	1	`jsx.object.isNativeMethod(jsx.global, "Math", "max")`
	2	`var result = Math.max(NaN, 2); typeof result == "number" && isNaN(result)`
	3	`var result = Math.max(1, NaN); typeof result == "number" && isNaN(result)`
	4	`var result = Math.max(2, 1); typeof result == "number" && result == 2`
	5	`var result = Math.max(1, 2); typeof result == "number" && result == 2`
	6	`var result = Math.max(0, 0); typeof result == "number" && result == 0`

```
      7  var result = Math.max(0, -0);
         typeof result == "number" && result == 0
      8  var result = Math.max(-0, 0);
         typeof result == "number" && result == 0
      9  var result = Math.max(-0, -0);
         typeof result == "number" && result == -0
```

```
171   1  jsx.object.isNativeMethod(jsx.global, "Math",
         "max")
      2  var result = Math.max();
         typeof result == "number" && result == -Infinity
      3  var result = Math.max(1, 2, NaN);
         typeof result == "number" && isNaN(result)
      4  var result = Math.max(1, 2, 3);
         typeof result == "number" && result == 3
      5  var result = Math.max(-0, -0, 0);
         typeof result == "number" && result == 0
      6  var result = Math.max(-0, -0, -0);
         typeof result == "number" && result == -0
      7  var len = Math.max.length;
         typeof len == "number" && len == 2
```

```
172   1  jsx.object.isNativeMethod(jsx.global, "Math",
         "min")
      2  var result = Math.min(NaN, 2);
         typeof result == "number" && isNaN(result)
      3  var result = Math.min(1, NaN);
         typeof result == "number" && isNaN(result)
      4  var result = Math.min(1, 2);
         typeof result == "number" && result == 1
      5  var result = Math.min(2, 1);
         typeof result == "number" && result == 1
      6  var result = Math.min(0, 0);
         typeof result == "number" && result == 0
      7  var result = Math.min(0, -0);
         typeof result == "number" && result == -0
      8  var result = Math.max(-0, 0);
         typeof result == "number" && result == -0
      9  var result = Math.max(-0, -0);
         typeof result == "number" && result == -0
```

```
173   1  jsx.object.isNativeMethod(jsx.global, "Math",
         "min")
```

```
      2  var result = Math.min();
         typeof result == "number" && result == Infinity
      3  var result = Math.min(1, 2, NaN);
         typeof result == "number" && isNaN(result)
      4  var result = Math.min(3, 2, 1);
         typeof result == "number" && result == 1
      5  var result = Math.min(0, 0, -0);
         typeof result == "number" && result == -0
      6  var result = Math.min(0, 0, 0);
         typeof result == "number" && result == 0
      7  var len = Math.min.length;
         typeof len == "number" && len == 2
```

```
174   1  typeof NaN == "number" && isNaN(NaN)
```

```
 28   1  jsx.object.isNativeMethod(jsx.global,
         "ActiveXObject")
```

```
175   1  jsx.object.isNativeMethod(jsx.global, "Number")
      2  var n = Number(void(0));
         typeof n == "number" && isNaN(n)
      3  var n = Number(null);
         typeof n == "number" && n == 0
      4  var n = Number(false);
         typeof n == "number" && n == 0
      5  var n = Number(true);
         typeof n == "number" && n == 1
      6  var n = Number(0);
         typeof n == "number" && n == 0
      7  var n = Number(-Infinity);
         typeof n == "number" && n == -Infinity
      8  var n = Number(NaN);
         typeof n == "number" && isNaN(n)
      9  var n = Number("");
         typeof n == "number" && n == 0
     10  var n = Number(" ");
         typeof n == "number" && n == 0
     11  var n = Number("42");
         typeof n == "number" && n == 42
     12  var n = Number(" 42");
         typeof n == "number" && n == 42
     13  var n = Number("42 ");
         typeof n == "number" && n == 42
```

```
14  var n = Number(" 42 ");
    typeof n == "number" && n == 42
15  var n = Number("0.5");
    typeof n == "number" && n == 0.5
```

```
176 1  !!jsx.object.getFeature(jsx.global, "Number",
       "MAX_VALUE")
    2  var n = Number.MAX_VALUE;
       typeof n == "number" && (n + 1 == n)
```

```
177 1  !!jsx.object.getFeature(jsx.global, "Number",
       "MIN_VALUE")
    2  var n = Number.MIN_VALUE;
       typeof n == "number" && (n - 1 == -1)
```

```
178 1  typeof Number.NaN == "number"
    2  isNaN(Number.NaN)
```

```
179 1  typeof jsx.object.getFeature(jsx.global, "Number",
       "NEGATIVE_INFINITY") == "number"
    2  Number.NEGATIVE_INFINITY == -Infinity
```

```
180 1  typeof jsx.object.getFeature(jsx.global, "Number",
       "POSITIVE_INFINITY") == "number"
    2  Number.POSITIVE_INFINITY == Infinity
```

```
181 1  !!jsx.object.getFeature(jsx.global, "Number",
       "prototype")
    2  typeof Number.prototype == "object" &&
       Number.prototype == 0
```

```
182 1  jsx.object.isNativeMethod(jsx.global, "Number",
       "prototype", "toString")
    2  (10).toString() == "10"
    3  (10).toString(10) == "10"
    4  (10).toString(2) == "1010"
    5  (10).toString(8) == "12"
    6  (10).toString(16) == "a"
    7  (31).toString(32) == "v"
    8  function f () { return (1).toString(1); }
       var result = jsx.tryThis(f, "e");
       result && result.name == "RangeError"
    9  function f () { return (1).toString(37); }
       var result = jsx.tryThis(f, "e");
       result && result.name == "RangeError"
```

```
10  var o = new Object();
    o.toString = Number.prototype.toString;
    function f () { o.toString(); }
    var error = false;
    function f2 (e) { error = true; }
    jsx.tryThis(f, f2);
    error
11  function f () { o.toString(); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Number", "prototype", "toString"))
    {
    var o = new Object();
    o.toString = Number.prototype.toString;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
```

```
183  1  jsx.object.isNativeMethod(jsx.global, "Object")
     2  var o = Object(null);
        typeof o == "object" && !o.foo
     3  var o = Object();
        typeof o == "object" && !o.foo
     4  var o = Object(false);
        typeof o == "object" && o.constructor == Boolean
        && o == false
     5  var o = Object(true);
        typeof o == "object" && o && o.constructor ==
        Boolean
     6  var o = Object(0);
        typeof o == "object" && o.constructor == Number
        && o == 0
     7  var o = Object("");
        typeof o == "object" && o.constructor == String
        && o == ""
     8  var o = new Object()
        var o2 = Object(o);
        typeof o2 == "object" && o2 == o
```

```
262  1  function f (id, args) { return id
        == "foo" && args && args[0] == 42; }
        var x = new Object();
        x.__noSuchMethod__ = f;
        x.foo(42);
```

266

```
1  jsx.object.isNativeMethod(jsx.global, "Object",
   "create")
2  function f () { return Object.create(42); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "create"))
   {
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
3  var o = Object.create(null);
   o && typeof o.toString == "undefined"
4  var a = Object.create(Array.prototype);
   a.toString == Array.prototype.toString
5  var descr = new Object();
   descr.foo = new Object();
   descr.foo.value = 42;
   var o = Object.create(null, descr);
   typeof o.foo == "number" && o.foo == 42
```

184

```
1  jsx.object.isNativeMethod(jsx.global, "Object",
   "defineProperties")
2  function f () { return Object.defineProperties(o,
   "a", new Object()); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperties"))
   {
   var o = 42;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
3  function f () { return Object.defineProperties(o,
   props); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperties"))
   {
   var o = new Object();
   var props = new Object();
   props.a = 42;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
```

```
4  var o = new Object();
   var props = new Object();
   props.a = new Object();
   props.a.value = true;
   props.a.enumerable = false;
   Object.defineProperties(o, props);
   var success = true;
   for (var p in o)
   {
   if (p == "a")
   {
   success = false;
   break;
   }
   }
   typeof o.a == "boolean" && success
5  var o = new Object();
   var props = new Object();
   props.a = new Object();
   props.a.value = true;
   props.a.configurable = false;
   Object.defineProperties(o, props);
   delete o.a;
   typeof o.a == "boolean" && o.a == true
6  function getter_ () { return o.b; }
   function f () { return Object.defineProperties(o,
   props); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperties"))
   {
   var o = new Object();
   o.b = 42;
   var props = new Object();
   props.a = new Object();
   props.a.value = true;
   props.a.configurable = false;
   Object.defineProperties(o, props);
   delete props.a.value;
   props.a['get'] = getter_;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError" && (o.a !=
   42)
   }
```

```
7  var o = new Object();
   var props = new Object();
   props.a = new Object();
   props.a.value = true;
   props.a.configurable = false;
   props.a.writable = true;
   Object.defineProperties(o, props);
   props.a.configurable = false;
   props.a.writable = true;
   props.a.enumerable = false;
   props.a.value = 42;
   Object.defineProperties(o, props);
   o.a == 42
8  function f () { return Object.defineProperties(o,
   props); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperties"))
   {
   var o = new Object();
   var props = new Object();
   props.a = new Object();
   props.a.value = true;
   props.a.writeable = false;
   Object.defineProperties(o, props);
   props.a.writable = true;
   props.a.value = 42;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError" && (o.a !=
   42)
   }
9  function f () { return Object.defineProperties(o,
   props); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperties"))
   {
   var props = new Object();
   props.a = new Object();
   props.a["get"] = 42;
   var o = new Object();
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
```

```
10  var props = new Object();
    props.a = new Object();
    function getter_ () { return this.b; }
    props.a["get"] = getter_;
    var o = new Object();
    o.b = true;
    function f () { Object.defineProperties(o, props);
    }
    jsx.tryThis(f);
    typeof o.a == typeof o.b && o.a == o.b
11  function f () { return Object.defineProperties(o,
    props); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperties"))
    {
    var props = new Object();
    props.a = new Object();
    props.a["set"] = 42;
    var o = new Object;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
12  var props = new Object();
    props.a = new Object();
    function setter_ (x) { this.b = x; }
    props.a["set"] = setter_;
    var o = new Object();
    o.b = true;
    Object.defineProperties(o, props);
    o.a = 42;
    o.b == 42
```

```
13  function getter_ () { return this.b; }
    function f () { return Object.defineProperties(o,
    props); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperties"))
    {
    var o = new Object();
    o.b = true;
    var props = new Object();
    props.a = new Object();
    props.a["get"] = getter_;
    props.a.value = 42;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
14  function getter_ () { return this.b; }
    function f () { return Object.defineProperties(o,
    props); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperties"))
    {
    var o = new Object();
    o.b = true;
    var props = new Object();
    props.a = new Object();
    props.a["get"] = getter_;
    props.a.writable = false;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
```

```
15  function setter_ (x) { this.b = x; }
    function f () { return Object.defineProperties(o,
    props); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperties"))
    {
    var o = new Object();
    o.b = true;
    var props = new Object();
    props.a = new Object();
    props.a["set"] = setter_;
    props.a.value = 42;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
16  function setter_ (x) { this.b = x; }
    function f () { return Object.defineProperty(o,
    props); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperties"))
    {
    var o = new Object();
    o.b = true;
    var props = new Object();
    props.a = new Object();
    props.a["set"] = setter_;
    props.a.writable = false;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
17  var o = new Object();
    var props = new Object();
    props.a = new Object();
    props.a.value = true;
    var o2 = Object.defineProperties(o, props);
    o2 == o
```

185	1	`jsx.object.isNativeMethod(jsx.global, "Object", "defineProperty")`

```
2  function f () { return Object.defineProperty(o,
   "a", new Object()); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperty"))
   {
   var o = 42;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
3  function f () { return Object.defineProperty(o,
   "a"); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperty"))
   {
   var o = new Object();
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
4  var o = new Object();
   var desc = new Object();
   desc.value = true;
   desc.enumerable = false;
   Object.defineProperty(o, "a", desc);
   var success = true;
   for (var p in o) { if (p == "a") { success =
   false; break; } }
   typeof o.a == "boolean" && success
5  var o = new Object();
   var desc = new Object();
   desc.value = true;
   desc.configurable = false;
   Object.defineProperty(o, "a", desc);
   delete o.a;
   typeof o.a == "boolean" && o.a == true
```

```
6  function getter_ () { return o.b; }
   function f () { return Object.defineProperty(o,
   "a", desc); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperty"))
   {
   var o = new Object();
   o.b = 42;
   var desc = new Object();
   desc.value = true;
   desc.configurable = false;
   Object.defineProperty(o, "a", desc);
   delete desc.value;
   desc['get'] = getter_;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError" && (o.a !=
   42)
   }
7  var o = new Object();
   var desc = new Object();
   desc.value = true;
   desc.configurable = false;
   desc.writable = true;
   Object.defineProperty(o, "a", desc);
   desc.configurable = false;
   desc.writable = true;
   desc.enumerable = false;
   desc.value = 42;
   Object.defineProperty(o, "a", desc);
   o.a == 42
```

```
8  function f () { return Object.defineProperty(o,
   "a", desc); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperty"))
   {
   var o = new Object();
   var desc = new Object();
   desc.value = true;
   desc.writeable = false;
   Object.defineProperty(o, "a", desc);
   desc.writable = true;
   desc.value = 42;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError" && (o.a !=
   42)
   }
9  function f () { return Object.defineProperty(o,
   "a", desc); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "defineProperty"))
   {
   var desc = new Object();
   desc["get"] = 42;
   var o = new Object;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
10 var desc = new Object();
   function getter_ () { return this.b; }
   desc["get"] = getter_;
   var o = new Object();
   o.b = true;
   function f () { Object.defineProperty(o, "a",
   desc); }
   jsx.tryThis(f);
   typeof o.a == typeof o.b && o.a == o.b
```

```
11  function f () { return Object.defineProperty(o,
    "a", desc); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperty"))
    {
    var desc = new Object();
    desc["set"] = 42;
    var o = new Object;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
12  var desc = new Object();
    function setter_ (x) { this.b = x; }
    desc["set"] = setter_;
    var o = new Object();
    o.b = true;
    Object.defineProperty(o, "a", desc);
    o.a = 42;
    o.b == 42
13  function getter_ () { return this.b; }
    function f () { return Object.defineProperty(o,
    "a", desc); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperty"))
    {
    var o = new Object();
    o.b = true;
    var desc = new Object();
    desc["get"] = getter_;
    desc.value = 42;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
```

```
14  function getter_ () { return this.b; }
    function f () { return Object.defineProperty(o,
    "a", desc); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperty"))
    {
    var o = new Object();
    o.b = true;
    var desc = new Object();
    desc["get"] = getter_;
    desc.writable = false;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
15  function setter_ (x) { this.b = x; }
    function f () { return Object.defineProperty(o,
    "a", desc); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperty"))
    {
    var o = new Object();
    o.b = true;
    var desc = new Object();
    desc["set"] = setter_;
    desc.value = 42;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
16  function setter_ (x) { this.b = x; }
    function f () { return Object.defineProperty(o,
    "a", desc); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "defineProperty"))
    {
    var o = new Object();
    o.b = true;
    var desc = new Object();
    desc["set"] = setter_;
    desc.writable = false;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
```

```
17  var o = new Object();
    var desc = new Object();
    desc.value = true;
    var o2 = Object.defineProperty(o, "a", desc);
    o2 == o
```

273

```
1   jsx.object.isNativeMethod(jsx.global, "Object",
    "freeze")
2   function f () { return Object.freeze(42); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "freeze"))
    {
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
3   var o = new Object();
    o.a = true;
    Object.freeze(o);
    o.a = 42;
    typeof o.a == "boolean"
4   var o = new Object();
    o.a = true;
    Object.freeze(o);
    delete o.a;
    typeof o.a == "boolean"
5   var o = new Object();
    Object.freeze(o);
    o.a = true;
    !o.a
6   function f () { return Object.defineProperty(o,
    "a", desc); }
    if (jsx.object.isNativeMethod(jsx.global,
    "Object", "freeze"))
    {
    var o = new Object();
    Object.freeze(o);
    var desc = new Object();
    desc.value = true;
    var result = jsx.tryThis(f, "e");
    result && result.name == "TypeError"
    }
7   var o = new Object();
    var o2 = Object.freeze(o);
    o2 == o
```

285 1
```
jsx.object.isNativeMethod(jsx.global, "Object",
"getOwnPropertyDescriptor")
```
2
```
function f () { Object.getOwnPropertyDescriptor(42);
}
if (jsx.object.isNativeMethod(jsx.global,
"Object", "getOwnPropertyDescriptor"))
{
var result = jsx.tryThis(f, "e");
result && result.name == "TypeError"
}
```
3
```
var o = new Object();
o.a = true;
delete o.b;
typeof Object.getOwnPropertyDescriptor(o, "b")
== "undefined"
```
4
```
if (jsx.object.isNativeMethod(jsx.global,
"Object", "getOwnPropertyDescriptor"))
{
var o = new Object();
o.a = true;
var result = Object.getOwnPropertyDescriptor(o,
"a");
typeof result.value == "boolean" && result.value
&& typeof result.writable == "boolean" &&
result.writable
&& typeof result["get"] == "undefined"
&& typeof result["set"] == "undefined"
&& typeof result.enumerable ==
"boolean" && result.enumerable
&& typeof result.configurable ==
"boolean" && result.configurable
}
```

```
5  if (jsx.object.isNativeMethod(jsx.global,
   "Object", "getOwnPropertyDescriptor"))
   {
   var o = new Object();
   var desc = new Object();
   desc.value = true;
   desc.writable = false;
   desc.enumerable = true;
   desc.configurable = true;
   Object.defineProperty(o, "a", desc);
   var result = Object.getOwnPropertyDescriptor(o,
   "a");
   typeof result.value == "boolean" && result.value
   && typeof result.writable == "boolean" &&
   !result.writable
   && typeof result["get"] == "undefined"
   && typeof result["set"] == "undefined"
   && typeof result.enumerable ==
   "boolean" && result.enumerable
   && typeof result.configurable ==
   "boolean" && result.configurable
   }
6  function f () { return this.b; }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "getOwnPropertyDescriptor"))
   {
   var o = new Object();
   o.b = true;
   var desc = new Object();
   desc["get"] = f;
   desc.enumerable = true;
   desc.configurable = true;
   Object.defineProperty(o, "a", desc);
   var result = Object.getOwnPropertyDescriptor(o,
   "a");
   typeof result.value == "undefined"
   && typeof result.writable == "undefined"
   && typeof result["get"] == "function"
   && typeof result["set"] == "undefined"
   && typeof result.enumerable ==
   "boolean" && result.enumerable
   && typeof result.configurable ==
   "boolean" && result.configurable
   }
```

```
7  function f (x) { this.b = x; }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "getOwnPropertyDescriptor"))
   {
   var o = new Object();
   o.b = true;
   var desc = new Object();
   desc["set"] = f;
   desc.enumerable = true;
   desc.configurable = true;
   Object.defineProperty(o, "a", desc);
   var result = Object.getOwnPropertyDescriptor(o,
   "a");
   typeof result.value == "undefined"
   && typeof result.writable == "undefined"
   && typeof result["get"] == "undefined"
   && typeof result["set"] == "function"
   && typeof result.enumerable ==
   "boolean" && result.enumerable
   && typeof result.configurable ==
   "boolean" && result.configurable
   }
8  if (jsx.object.isNativeMethod(jsx.global,
   "Object", "getOwnPropertyDescriptor"))
   {
   var o = new Object();
   var desc = new Object();
   desc.value = true;
   desc.writable = true;
   desc.enumerable = false;
   desc.configurable = true;
   Object.defineProperty(o, "a", desc);
   var result = Object.getOwnPropertyDescriptor(o,
   "a");
   typeof result.value == "boolean" && result.value
   && typeof result.writable == "boolean" &&
   result.writable
   && typeof result["get"] == "undefined"
   && typeof result["set"] == "undefined"
   && typeof result.enumerable ==
   "boolean" && !result.enumerable
   && typeof result.configurable ==
   "boolean" && result.configurable
   }
```

```
9  if (jsx.object.isNativeMethod(jsx.global,
   "Object", "getOwnPropertyDescriptor"))
   {
   var o = new Object();
   var desc = new Object();
   desc.value = true;
   desc.writable = true;
   desc.enumerable = true;
   desc.configurable = false;
   Object.defineProperty(o, "a", desc);
   var result = Object.getOwnPropertyDescriptor(o,
   "a");
   typeof result.value == "boolean" && result.value
   && typeof result.writable == "boolean" &&
   result.writable
   && typeof result["get"] == "undefined"
   && typeof result["set"] == "undefined"
   && typeof result.enumerable ==
   "boolean" && result.enumerable
   && typeof result.configurable ==
   "boolean" && !result.configurable
   }
```

```
186  1  jsx.object.isNativeMethod(jsx.global, "Object",
        "getOwnPropertyNames")
     2  function f () { return Object.getOwnPropertyNames(42);
        }
        if (jsx.object.isNativeMethod(jsx.global,
        "Object", "getOwnPropertyNames"))
        {
        var result = jsx.tryThis(f, "e");
        result && result.name == "TypeError"
        }
     3  var result = Object.getOwnPropertyNames(new
        Object());
        result && result.constructor == Array &&
        result.length == 0
     4  var a = new Array();
        var result = Object.getOwnPropertyNames(a);
        result.length == 1 && result[0] == "length"
```

```
187  1  jsx.object.isNativeMethod(jsx.global, "Object",
        "getPrototypeOf")
```

	2	`function f () { Object.getPrototypeOf(42); }`
		`if (jsx.object.isNativeMethod(jsx.global,`
		`"Object", "getPrototypeOf"))`
		`{`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`
		`}`

	3	`Object.getPrototypeOf(new Object()) ==` `Object.prototype`
284	1	`jsx.object.isNativeMethod(jsx.global, "Object",` `"isExtensible")`

	2	`function f () { Object.isExtensible(42); }`
		`if (jsx.object.isNativeMethod(jsx.global,`
		`"Object", "isExtensible"))`
		`{`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`
		`}`

	3	`var o = new Object();`
		`o.a = true;`
		`Object.isExtensible(o)`

	4	`var o = new Object();`
		`Object.preventExtensions(o);`
		`!Object.isExtensible(o)`

279	1	`jsx.object.isNativeMethod(jsx.global, "Object",` `"isFrozen")`

	2	`function f () { Object.isFrozen(42); }`
		`if (jsx.object.isNativeMethod(jsx.global,`
		`"Object", "isFrozen"))`
		`{`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`
		`}`

	3	`var o = new Object();`
		`o.a = true;`
		`!Object.isFrozen(o);`

	4	`var o = new Object();`
		`o.a = true;`
		`Object.freeze(o);`
		`Object.isFrozen(o)`

277	1	`jsx.object.isNativeMethod(jsx.global, "Object",` `"isSealed")`

```
2  function f () { Object.isSealed(42); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "isSealed"))
   {
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
3  var o = new Object();
   o.a = true;
   !Object.isSealed(o);
4  function C() {}
   C.prototype.b = true;
   var o = new C();
   o.a = true;
   Object.seal(o);
   Object.isSealed(o)
5  var o = new Object();
   Object.seal(o);
   Object.isSealed(o)
```

```
268  1  jsx.object.isNativeMethod(jsx.global, "Object",
        "keys")
     2  function f () { Object.keys(42); }
        if (jsx.object.isNativeMethod(jsx.global,
        "Object", "keys"))
        {
        var result = jsx.tryThis(f, "e");
        result && result.name == "TypeError"
        }
     3  var result = Object.keys(new Object());
        result && typeof result[0] == "undefined" &&
        result.length == 0
     4  function C () {}
        C.prototype.b = 42;
        var o = new C();
        o.a = true;
        var result = Object.keys(o);
        result && result[0] == "a" && typeof result[1]
        == "undefined" && result.length == 1
```

```
276  1  jsx.object.isNativeMethod(jsx.global, "Object",
        "preventExtensions")
```

```
2  function f () { Object.preventExtensions(42); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "preventExtensions"))
   {
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
3  var o = new Object();
   Object.preventExtensions(o);
   o.a = true;
   !o.a
4  function f () { return Object.defineProperty(o,
   "a", desc); }
   if (jsx.object.isNativeMethod(jsx.global,
   "Object", "preventExtensions"))
   {
   var o = new Object();
   Object.preventExtensions(o);
   var desc = new Object();
   desc.value = true;
   var result = jsx.tryThis(f, "e");
   result && result.name == "TypeError"
   }
5  var o = new Object();
   var o2 = Object.preventExtensions(o);
   o2 == o
```

```
188  1  !!jsx.object.getFeature(jsx.global, "Object",
        "prototype")
```

```
189  1  jsx.object.isNativeMethod(new Object(),
        "__defineGetter__")
   2  var o = new Object();
      o.bar = 42;
      function f () { return this.bar; }
      o.__defineGetter__("foo", f);
      o.foo == 42
```

```
190  1  jsx.object.isNativeMethod(new Object(),
        "__defineSetter__")
```

	2	`var o = new Object();`
		`o.bar = 23;`
		`function f (v) { this.bar = v; }`
		`o.__defineSetter__("foo", f);`
		`o.foo = 42;`
		`o.bar == 42`

192	1	`typeof Object.prototype.__proto__ == "object"`
	2	`new Object().__proto__ == Object.prototype`
193	1	`jsx.object.isNativeMethod(jsx.global, "Object", "prototype", "constructor")`
	2	`new Object().constructor == Object && (new (new Object()).constructor()).constructor == Object`
194	1	`jsx.object.isNativeMethod(jsx.global, "Object", "prototype", "hasOwnProperty")`

	2	`function C () {}`
		`C.prototype.foo = "bar";`
		`var o = new C();`
		`!o.hasOwnProperty("foo")`

	3	`function C () {}`
		`C.prototype.foo = "bar";`
		`var o = new C();`
		`o.foo = "baz"`
		`o.hasOwnProperty("foo")`

195	1	`jsx.object.isNativeMethod(jsx.global, "Object", "prototype", "isPrototypeOf")`
	2	`!new Object().isPrototypeOf()`
	3	`var o = Object.create(null);` `!(new Object()).isPrototypeOf(o)`

	4	`function C () {}`
		`function C2 () {}`
		`!C.prototype.isPrototypeOf(new C2())`

	5	`function C () {}`
		`function C2 () {}`
		`C.prototype.isPrototypeOf(new C())`

	6	`function C () {}`
		`function C2 () {}`
		`C2.extend(C);`
		`C.prototype.isPrototypeOf(new C2())`

196	1	`jsx.object.isNativeMethod(jsx.global, "Object", "prototype", "propertyIsEnumerable")`

	2	`!new Array().propertyIsEnumerable("0")`
	3	`!new Array().propertyIsEnumerable("length")`

	4	`var o = new Object();`
		`o.foo = "bar";`
		`o.propertyIsEnumerable("foo")`

197	1	`jsx.object.isNativeMethod(jsx.global, "Object",` `"prototype", "toSource")`
	2	`new Object().toSource() == "({})"`
269	1	`jsx.object.isNativeMethod(jsx.global, "Object",` `"seal")`

	2	`function f () { return Object.seal(42); }`
		`if (jsx.object.isNativeMethod(jsx.global,`
		`"Object", "seal"))`
		`{`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`
		`}`

	3	`var o = new Object();`
		`o.a = true;`
		`Object.seal(o);`
		`delete o.a;`
		`o.a`

	4	`var o = new Object();`
		`Object.seal(o);`
		`o.a = true;`
		`!o.a`

	5	`function f () { return Object.defineProperty(o,`
		`"a", desc); }`
		`if (jsx.object.isNativeMethod(jsx.global,`
		`"Object", "seal"))`
		`{`
		`var o = new Object();`
		`Object.seal(o);`
		`var desc = new Object();`
		`desc.value = true;`
		`var result = jsx.tryThis(f, "e");`
		`result && result.name == "TypeError"`
		`}`

	6	`var o = new Object();`
		`var o2 = Object.seal(o);`
		`o2 == o`

201	1	`jsx.object.isNativeMethod(jsx.global, "print")`

213	1	`"ab".match("b").index == 1`
211	1	`"ab".match(".").input == "ab"`
205	1	`jsx.object.isNativeMethod(jsx.global, "RegExp")`
	2	`"aA".match(new RegExp("a", "gi")).length == 2`
206	1	`jsx.object.isNativeMethod(jsx.global, "RegExp")`
	2	`"a\nA".match(new RegExp("^a$", "gim")).length == 2`
207	1	`"(a)".match(new RegExp("a"));` `RegExp.$1 == "a"`
216	1	`var v = new RegExp(".").global;` `typeof v == "boolean" && !v`
	2	`var v = new RegExp(".", "g").global;` `typeof v == "boolean" && v`
217	1	`var v = new RegExp("a").ignoreCase;` `typeof v == "boolean" && !v`
	2	`var v = new RegExp("a", "i").ignoreCase;` `typeof v == "boolean" && v`
218	1	`var v = new RegExp(".").multiline;` `typeof v == "boolean" && !v`
	2	`var v = new RegExp(".", "m").multiline;` `typeof v == "boolean" && v`
214	1	`jsx.object.isNativeMethod(jsx.global, "RegExp",` `"prototype", "compile")`
215	1	`jsx.object.isNativeMethod(jsx.global, "RegExp",` `"prototype", "exec")`
	2	`var result = new RegExp("foo").exec("foobar");` `result.length == 1 && result[0] == "foo"`

	3	`var rx = new RegExp("o", "g");`
		`var s = "foobar";`
		`var result = rx.exec(s);`
		`result.length == 1 && result[0] == "foo"`
		`result = rx.exec(s);`
		`result.length == 1 && result[0] == "foo"`
		`result = rx.exec(s);`
		`typeof result == "object" && !result`

219	1	`new RegExp("a", "gi").source == "a"`
208	1	`"a".match(new RegExp("a"));` `RegExp["$&"] == "a"`

	2	`"a".match(new RegExp("a"));` `RegExp.lastMatch == "a"`
209	1	`"ab".match(new RegExp("a"));` `RegExp["$'"] == "b"`
	2	`"ab".match(new RegExp("a"));` `RegExp.rightContext == "b"`
210	1	`"a".match(new RegExp("(.)"));` `RegExp["$+"] == "a"`
	2	`"a".match(new RegExp("(.)"));` `RegExp.lastParen == "a"`
212	1	`"ab".match(new RegExp("b"));` `RegExp["$`"] == "a"`
	2	`"ab".match(new RegExp("b"));` `RegExp.leftContext == "a"`
220	1	`var sb : sbyte;`
221	1	`jsx.object.isNativeMethod(jsx.global,` `"ScriptEngine")`
	2	`ScriptEngine()`
222	1	`jsx.object.isNativeMethod(jsx.global,` `"ScriptEngineBuildVersion")`
	2	`ScriptEngineBuildVersion()`
223	1	`jsx.object.isNativeMethod(jsx.global,` `"ScriptEngineMajorVersion")`
	2	`ScriptEngineMajorVersion()`
224	1	`jsx.object.isNativeMethod(jsx.global,` `"ScriptEngineMinorVersion")`
	2	`ScriptEngineMinorVersion()`
226	1	`var s : short;`
227	1	`jsx.object.isNativeMethod(jsx.global, "String",` `"fromCharCode")`
	2	`String.fromCharCode() == ""`
	3	`String.fromCharCode(0x20) == " "`
	4	`String.fromCharCode(0x20AC) == "€"`
	5	`var len = String.fromCharCode.length;` `typeof len == "number" && len == 1`
228	1	`!!jsx.object.getFeature(jsx.global, "String",` `"prototype")`

```
     2  var p = String.prototype;
        typeof p == "object" && p == ""
```

```
229  1  jsx.object.isNativeMethod(jsx.global, "String",
        "prototype", "charCodeAt")
     2  var result = "x".charCodeAt(-1);
        typeof result == "number" && isNaN(result)
     3  var result = "x".charCodeAt(1);
        typeof result == "number" && isNaN(result)
     4  var result = "x".charCodeAt(0);
        typeof result == "number" && result == 120
     5  var result = "€".charCodeAt(0);
        typeof result == "number" && result == 0x20AC
     6  function f () { return this.value; }
        var o = new Object();
        o.toString = f;
        o.value = "€";
        o.charCodeAt = String.prototype.charCodeAt;
        o.charCodeAt(0) == 0x20AC
```

```
230  1  jsx.object.isNativeMethod(jsx.global, "String",
        "prototype", "concat")
     2  "".concat() == ""
     3  "foo".concat("bar") == "foobar"
     4  "foo".concat("bar", "baz") == "foobarbaz"
     5  var len = String.prototype.concat.length;
        typeof len == "number" && len == 1
     6  function MyString(str) { this.value = str; }
        function myString_toString () { return this.value;
        }
        MyString.prototype.toString = myString_toString;
        MyString.prototype.concat = String.prototype.concat;
        (new MyString("foo")).concat(new
        MyString("bar")).concat("baz") == "foobarbaz"
```

```
231  1  jsx.object.isNativeMethod(jsx.global, "String",
        "prototype", "localeCompare")
     2  "a".localeCompare("ä")
```

	3	`function MyString (str) { this.value = str; }`
		`function myString_toString () { return this.value;`
		`}`
		`MyString.prototype.toString = myString_toString;`
		`MyString.prototype.localeCompare`
		`= String.prototype.localeCompare;`
		`(new MyString("a")).localeCompare(new`
		`MyString("ä"))`

232	1	`jsx.object.isNativeMethod(jsx.global, "String",` `"prototype", "match")`

	2	`var result1 = "a".match("a");`
		`var result2 = "a".match(new RegExp("a"));`
		`result1[0] == result2[0]`

	3	`var result1 = "aaa".match(new RegExp("a"));`
		`var result2 = new RegExp("a").exec("aaa");`
		`result1[0] == result2[0]`

	4	`var s = "abc";`
		`var rx = new RegExp(".", "g");`
		`var a = s.match(rx);`
		`var rx2 = new RegExp(".", "g");`
		`var a2 = rx2.exec(s);`
		`a2[a2.length] = rx2.exec(s)[0];`
		`a2[a2.length] = rx2.exec(s)[0];`
		`(a.length == a2.length) && (a[0] == a2[0]) &&`
		`(a[1] == a2[1]) && (a[2] == a2[2])`

	5	`function MyString (str) { this.value = str; }`
		`function myString_toString () { return this.value;`
		`}`
		`MyString.prototype.toString = myString_toString;`
		`MyString.prototype.match = String.prototype.match;`
		`var result1 = (new MyString("aaa")).match(new`
		`RegExp("a"));`
		`var result2 = new RegExp("a").exec((new`
		`MyString("aaa")));`
		`result1[0] == result2[0]`

234	1	`jsx.object.isNativeMethod(jsx.global, "String",` `"prototype", "replace")`

```
2  var success = false;
   function f (match, p1, p2, offset, s)
   {
   !success && (match == "ab") && (p1 ==
   "a") && (p2 == "b") && (offset == 0)
   && (s == "abc") && (success = true);
   return "xx";
   }
   var result = "abc".replace(new RegExp("(.)(.)"),
   f);
   success && result == "xxc";
3  function MyString (str) { this.value = str; }
   function myString_toString () { return this.value;
   }
   MyString.prototype.toString = myString_toString;
   MyString.prototype.replace = String.prototype.replace;
   var success = false;
   function f (match, p1, p2, offset, s)
   {
   !success && (match == "ab") && (p1 ==
   "a") && (p2 == "b") && (offset == 0)
   && (s == "abc") && (success = true);
   return "xx";
   }
   var result = new MyString("abc").replace(new
   RegExp("(.)(.)"), f);
   success && result == "xxc";
```

233	1	`jsx.object.isNativeMethod(jsx.global, "String", "prototype", "replace")`
	2	`var rx = new RegExp("."); "abc".replace(rx, "x") == "xbc"`
	3	`var rx = new RegExp(".", "g"); "abc".replace(rx, "x") == "xxx"`
	4	`"aba".replace("a", "x") == "xba"`
	5	`"abc".replace(new RegExp( "(b)(k?(j?(i?(h?(g?(f?(e?(d?(c?))))))))))"), "$$ $& $` $' $1 $10") == "a$ bc a b c"`

```
6  function MyString (str) { this.value = str; }
   function myString_toString () { return this.value;
   }
   MyString.prototype.toString = myString_toString;
   MyString.prototype.replace = String.prototype.replace;
   (new MyString("abc")).replace(new RegExp(
   "(b)(k?(j?(i?(h?(g?(f?(e?(d?(c?))))))))))"),
   "$$ $& $` $' $1 $10") == "a$ bc a b c"
```

```
235  1  jsx.object.isNativeMethod(jsx.global, "String",
        "prototype", "search")
     2  var result = "a".search(".");
        var result2 = "a".search(new RegExp("."));
        result == result2
     3  var result = "ab".search("b");
        typeof result == "number" && result == 1
     4  var result = "a".search("b");
        typeof result == "number" && result == -1
     5  function MyString (str) { this.value = str; }
        function myString_toString () { return this.value;
        }
        MyString.prototype.toString = myString_toString;
        MyString.prototype.search = String.prototype.search;
        var result = (new MyString("ab")).search(new
        MyString("b"));
        typeof result == "number" && result == 1
```

```
236  1  jsx.object.isNativeMethod(jsx.global, "String",
        "prototype", "slice")
     2  var result = "abc".slice();
        typeof result == "string" && result == "abc"
     3  var result = "abc".slice(1);
        typeof result == "string" && result == "bc"
     4  var result = "abc".slice(-1);
        typeof result == "string" && result == "c"
     5  var result = "abc".slice(1, 2);
        typeof result == "string" && result == "b"
     6  var result = "abc".slice(-2, 2);
        typeof result == "string" && result == "b"
     7  var result = "abc".slice(1, -1);
        typeof result == "string" && result == "b"
     8  var result = "abc".slice(-1, -1);
        typeof result == "string" && result == ""
```

	9	`var len = String.prototype.slice.length;` `typeof len == "number" && len == 2`

	10	`function MyString (str) { this.value = str; }`
		`function myString_toString () { return this.value;`
		`}`
		`MyString.prototype.toString = myString_toString;`
		`MyString.prototype.slice = String.prototype.slice;`
		`var result = (new MyString("abc")).slice(1, 2);`
		`typeof result == "string" && result == "b"`

265	1	`var result = "ab".split(new RegExp(""));`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

	2	`var result = "ab".split(new RegExp("(?:)"));`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

	3	`var result = "a,b;c".split(new RegExp("[,;]"));`
		`result.length == 3 && result[0] == "a" &&`
		`result[1] == "b" && result[2] == "c"`

	4	`var result = "abc".split("", 2);`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

	5	`var result = "a,b;c".split(new RegExp("[,;]"), 2);`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

	6	`var len = String.prototype.split.length;` `typeof len == "number" && len == 2`

	7	`function MyString (str) { this.value = str; }`
		`function myString_toString () { return this.value;`
		`}`
		`MyString.prototype.toString = myString_toString;`
		`MyString.prototype.split = String.prototype.split;`
		`var result = (new MyString("ab")).split(new`
		`RegExp(""));`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

237	1	`jsx.object.isNativeMethod(jsx.global, "String",` `"prototype", "split")`
	2	`var result = "a,b".split();` `result.length == 1 && result[0] == "a,b"`

	3	`var result = "ab".split("");`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

	4	`var result = "a,b".split(",");`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

	5	`function MyString (str) { this.value = str; }`
		`function myString_toString () { return this.value;`
		`}`
		`MyString.prototype.toString = myString_toString;`
		`MyString.prototype.split = String.prototype.split;`
		`var result = (new MyString("a,b")).split(new`
		`MyString(","));`
		`result.length == 2 && result[0] == "a" &&`
		`result[1] == "b"`

238	1	`jsx.object.isNativeMethod(jsx.global, "String",` `"prototype", "substr")`
	2	`"abc".substr() == "abc"`
	3	`"abc".substr(1) == "bc"`
	4	`"abc".substr(1, 1) == "b"`
	5	`"abc".substr(-2) == "bc"`
	6	`"abc".substr(-2, 1) == "b"`
	7	`var len = String.prototype.substr.length;` `typeof len == "number" && len == 2`

	8	`function MyString (str) { this.value = str; }`
		`function myString_toString () { return this.value;`
		`}`
		`MyString.prototype.toString = myString_toString;`
		`MyString.prototype.substr = String.prototype.substr;`
		`(new MyString("abc")).substr(1, 1) == "b"`

239	1	`jsx.object.isNativeMethod(jsx.global, "String",` `"prototype", "trim")`

	2	`var result = "\u0009\u000B\u000C\u0020\u00A0\uFEFFx\u000A`
		`\u000D\u2028\u2029".trim();`
		`typeof result == "string" && result.length == 1`
		`&& result == "x"`

	3	`function MyString (str) { this.value = str; }`
		`function myString_toString () { return this.value;`
		`}`
		`MyString.prototype.toString = myString_toString;`
		`MyString.prototype.trim = String.prototype.trim;`
		`var result = (new MyString("\u0009\u000B\u000C\u0020`
		`\u00A0\uFEFFx\u000A\u000D\u2028\u2029")).trim();`
		`typeof result == "string" && result.length == 1 &&`
		`result == "x"`

240	1	`"abc"[1] == "b"`

241	1	`var b = true;`
		`switch (b)`
		`{`
		`case true:`
		`true;`
		`break;`
		
		`default:`
		`false;`
		`}`

242	1	`jsx.tryThis("throw 42", "e") == 42`

243	1	`var result = jsx.tryThis("try`
		`{ true; } catch (e) {}", "e");`
		`typeof result == "boolean" && result`

	2	`var result = jsx.tryThis("try { new`
		`Object()(); } catch (e) { true; }", "e");`
		`typeof result == "boolean" && result`

245	1	`var result = jsx.tryThis("try {} catch`
		`(e) { e; } finally { true; }", "e");`
		`typeof result == "boolean" && result`

	2	`var result = jsx.tryThis("try { new Object()();`
		`} catch (e2) { e2; } finally { true; }", "e");`
		`typeof result == "boolean" && result`

246	1	`var result = jsx.tryThis("try {`
		`new Object()(); } catch (e if e`
		`instanceof TypeError) { true; }", "e");`
		`typeof result == "boolean" && result`

244	1	`var result = jsx.tryThis("try {} finally { true;`
		`}", "e");`
		`typeof result == "boolean" && result`

247	1	`typeof foo == "undefined"`
	2	`var foo;` `typeof foo == "undefined"`
	3	`var foo = null;` `typeof foo == "object"`
	4	`typeof false == "boolean"`
	5	`typeof true == "boolean"`
	6	`typeof 42 == "number"`
	7	`typeof "" == "string"`

	8	`typeof new Object() == "object"`
	9	`function f () {}` `typeof f == "function"`
248	1	`undefined; true`
	2	`typeof undefined == "undefined"`
24	1	`var \u0041 = true; \u0041`
267	1	`var π = 3.141592653589793; typeof π == "number"`
249	1	`jsx.object.isNativeMethod(jsx.global, "VBArray", "prototype", "dimensions")`
250	1	`jsx.object.isNativeMethod(jsx.global, "VBArray", "prototype", "getItem")`
251	1	`jsx.object.isNativeMethod(jsx.global, "VBArray", "prototype", "lbound")`
252	1	`typeof window != "undefined" && window`
257	1	`eval("function gen() { yield true; } var g = gen(); true")`
25	1	`({a: "b"}).a == "b"`
26	1	`({foo: "bar",})`

Tabelle A.3.: Testfälle

A.3. Resultate

A.3.1. Marktanteile Februar 2012

Browser Version	Total Market Share
Microsoft Internet Explorer 8.0	27.85%
Microsoft Internet Explorer 9.0	12.60%
Firefox 10	8.44%
Chrome 16.0	8.09%
Chrome 17.0	8.06%
Microsoft Internet Explorer 6.0	6.87%
Microsoft Internet Explorer 7.0	4.69%
Firefox 9.0	4.45%
Safari 5.1	3.61%
Firefox 3.6	3.10%
Opera 11.x	1.48%
Firefox 8.0	1.38%
Safari 5.0	1.09%
Firefox 4.0	0.62%
Firefox 6.0	0.57%
Firefox 7.0	0.51%
Firefox 5.0	0.50%
Chrome 14.0	0.45%
Firefox 3.5	0.44%
Firefox 3.0	0.44%
Chrome 15.0	0.39%
Chrome 12.0	0.34%
Firefox 11	0.27%
Chrome 18.0	0.25%
Proprietary or Undetectable	0.24%
Chrome 13.0	0.21%
Safari 5.0 - Maxthon Edition	0.20%
Microsoft Internet Explorer 8.0 - Maxthon Edition	0.15%
Chrome 11.0	0.14%
Safari 4.1	0.14%
Safari 4.0	0.14%
Chrome 10.0	0.14%
Microsoft Internet Explorer 6.0 - Tencent Traveler Edition	0.12%
Microsoft Internet Explorer 8.0 - TheWorld Edition	0.12%

Microsoft Internet Explorer 8.0 - Tencent Traveler Edition	0.11%
Opera 10.x	0.11%
Firefox 2.0	0.11%
Chrome 16.0 - Maxthon Edition	0.10%
Opera 9.x	0.10%
Chrome 18.0 - Maxthon Edition	0.10%

Tabelle A.4.: Browser-Marktanteile Februar 2012 (Quelle: NET APPLICATIONS.COM 2011)

A.3.2. Nicht sichere Features

ID	Code	Beschreibung
288	`"use strict";`	Strict Mode
6	`/[^]/ : RegExp`	RegExp literal with empty negated character range
9	`/.../[g][i][m][y] : RegExp`	RegExp literal with optional sticky modifier
134	`= function (...) expression : Function`	
22	`[expression for each (propertyValue in Object) [if (condition)]] : Array`	Array comprehension
21	`[value, ] : Array`	Array initializer with trailing comma
23	`[var] [var1, [var2], var3, ...] = Array`	Destructuring assignment
263	`arguments : Arguments`	
34	`Array.every(iterable, callback : Function) : boolean`	
41	`Array.prototype.every(callback : Function[, thisValue]) : boolean`	
42	`Array.prototype.filter( callback : Function[, thisArg : Object]) : number|int`	
43	`Array.prototype.forEach( callback : Function[, thisArg : Object]) : number|int`	
44	`Array.prototype.indexOf( searchElement[, fromIndex : Number|int]) : number|int`	
46	`Array.prototype.lastIndexOf( searchElement[, fromIndex : Number|int]) : number|int`	
48	`Array.prototype.map(callback : Function[, thisArg : Object]) : number|int`	

```
 49  Array.prototype.pop()
 51  Array.prototype.reduce(
     callback : Function[,
     initialValue]) : any
 52  Array.prototype.reduceRight(
     callback : Function[,
     initialValue]) : any
 56  Array.prototype.some(callback
     : Function[, thisValue]) :
     boolean
 59  Array.prototype.toSource()
     : string
 60  Array.prototype.toString()
     : string
 61  Array.prototype.unshift()
     : number|int
 37  Array.some(iterable,
     callback : Function) :
     boolean
 62  boolean
 65  Boolean.prototype.toSource()
     : string
 68  byte
 70  char
286  Date.now()
 84  Date.prototype.getVarDate()
 97  Date.prototype.toISOString()
     : string
 98  Date.prototype.toJSON([key])
     : string
100  Date.prototype.toLocaleFormat(
     format : String) : string
103  Date.prototype.toSource()
     : string
109  decimal
112  delete
114  double
118  Enumerator(...)
119  Error([message : String])      Error constructor/factory
120  error.description : string
123  error.number : number|int
124  error.stack : string
```

128	`for each ([var] identifier in Object)`	
137	`function.arity : number|int`	
138	`Function.prototype.apply(...)`	
31	`Function.prototype.arguments.caller : Function|null`	
287	`Function.prototype.bind( thisArg[, arg1[, arg2, ...]]) : Function`	
146	`Function.prototype.toSource() : string`	
147	`generator.close()`	
148	`generator.next()`	
149	`generator.send(expression)`	
150	`generator.throw(expression)`	
151	`GetObject(...)`	
17	`identifier : type`	Type declaration
158	`instanceof`	
159	`int`	
163	`Iterator(Object) : Iterator`	
164	`JSON.parse(text : String[,reviver : Function]) : Object`	
165	`JSON.stringify(value[, replacer[, space]]) : string`	
166	`let (assignment[, ...]) { [statements] }`	Block scoping: let statement
167	`let (assignment[, ...]) expression`	Block scoping: let expression
168	`let assignment[, ...]`	Block scoping: let definition
169	`long`	
28	`new ActiveXObject( "serverName.typeName"[, location : String])`	
182	`Number.prototype.toString() : string`	
262	`object.__noSuchMethod__ : Function`	__noSuchMethod__ method

```
266 Object.create(obj :
    Object[, properties])
184 Object.defineProperties(o
    : Object, properties :
    Object) : Object
185 Object.defineProperty(o :
    Object, property : String,
    attr : Object) : Object
273 Object.freeze(o : Object)
    : Object
285 Object
    .getOwnPropertyDescriptor(obj
    : Object, property :
    String) : Object
186 Object.getOwnPropertyNames(o
    : Object) : Array
187 Object.getPrototypeOf(o :
    Object) : Object
284 Object.isExtensible(o :
    Object) : boolean
279 Object.isFrozen(o :
    Object) : boolean
277 Object.isSealed(o :
    Object) : boolean
268 Object.keys(o : Object) :
    Array
276 Object.preventExtensions(o
    : Object) : Object
189 Object.prototype
    .__defineGetter__(propertyName
    : String, getter :
    Function)
190 Object.prototype
    .__defineSetter__(propertyName
    : String, setter :
    Function)
192 Object.prototype.__proto__
195 Object.prototype
    .isPrototypeOf(o : Object)
    : boolean
197 Object.prototype.toSource()
    : string
```

269	`Object.seal(obj : Object) : Object`	
201	`print(string)`	
207	`RegExp.$integer`	
220	`sbyte`	
221	`ScriptEngine()`	
222	`ScriptEngineBuildVersion()`	
223	`ScriptEngineMajorVersion()`	
224	`ScriptEngineMinorVersion()`	
226	`short`	
227	`String.fromCharCode(Number|uint)`	
236	`String.prototype.slice(...)`	
238	`String.prototype.substr(start[, length])`	
239	`String.prototype.trim() : string`	
240	`string[Number|uint]`	String subscripting
246	`try { [statements] } catch (identifier if expression) { [statements] } [catch (identifier) { [statements] }] [finally { [statements] }]`	
249	`VBArray.prototype.dimensions()`	
250	`VBArray.prototype.getItem(...)`	
251	`VBArray.prototype.lbound(...)`	
257	`yield expression`	Generator expression
26	`{propertyName: propertyValue, } : Object`	Object initializer with trailing comma

Tabelle A.5.: Nicht sichere Features

Selbständigkeitserklärung

«Ich erkläre hiermit, dass ich diese Thesis selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche kenntlich gemacht. Ich versichere zudem, dass ich bisher noch keine wissenschaftliche Arbeit mit gleichem oder ähnlichem Inhalt an der Fernfachhochschule Schweiz oder an einer anderen Hochschule eingereicht habe. Mir ist bekannt, dass andernfalls die Fernfachhochschule Schweiz zum Entzug des aufgrund dieser Thesis verliehenen Titels berechtigt ist.»

Zollikofen, 12. März 2012

B. Unterschiede zur Originalfassung

Diese überarbeitete Fassung vom 20. Juni 2013 (Revision 27) unterscheidet sich – ausser in den in Anhang A.1.2.1 und A.1.2.2 angegebenen Punkten – von der bei der Fernfachhochschule Schweiz am 12. März 2012 eingereichten, von dieser akzeptierten und am 26. April 2012 positiv geprüften Originalfassung (Revision 18a).

B.1. Allgemein

- «d. h. » und «z. B. » wurden mit `\ie` und `\eg` durchgängig typographisch richtig gesetzt (schmales geschütztes Leerzeichen, Abkürzung gefolgt von Leerzeichen).
- Das erste Zeichen »'« in der Danksagung wurde durch das Zeichen »‘« ersetzt.
- Namen von Autoren wurden in Kapitälchen gesetzt.
- Das Kapitel «Unterschiede zur Originalfassung» wurde hinzugefügt.

B.2. Abstract

- Der Seitenkopf und die Seitenzahl wurden entfernt. (`\thispagestyle{empty}` ergänzt)
- Zusätzliche Silbentrennung bei «ECMAScript» für korrektes Schriftbild
- 2. Absatz: «Features können ~~werden~~ so … werden, … » (Wort entfernt)

B.3. Einführung

- Motivation
  - Webressourcen
    - «~~Die~~ Ecma-TC39 stellt auf ~~ihrer~~ Website» wurde in «Das Ecma-TC39 stellt auf seiner Website» geändert. (Technical Committee)
    - «Zaytsev untersucht auf seiner Website die Kompatibilität von Features, die mit ECMAScript Edition 5 eingeführt wurden; von Features, die mit Edition 6 eingeführt werden sollen, und ~~von auch~~auch von proprietären Features.» (Worte vertauscht)

B.4. ECMAScript und ECMAScript-Implementierungen

- ECMAScript
  - ECMAScript-Editionen
    - Klassifizierung
      - Syntax
        - Seite 16 unten: schliessende Klammer ergänzt
- ECMAScript-Implementierungen
  - KDE JavaScript (seit 2000)
    - »Debian GNU/Linux Unstable („Sid“)« wurde in »Debian GNU/Linux «sid» (instabil)« geändert. (Anpassung an offizielle Schreibweise)
- Kompatibilitätsmassnahmen
  - Syntaxelemente
    - Der Satz «Nicht alle Punktuatoren sind auch Operatoren (z. B. ist `(...)` der Gruppierungsoperator [*grouping operator*], die Eigenschaftszugriffsyntax `[...]` [*bracket property accessor*] hingegen nicht).» wurde durch «Nicht alle Punktuatoren sind auch Operatoren; z. B. ist `(...)` der Gruppierungsoperator (*grouping operator*), die Eigenschaftszugriffsyntax `[...]` (*bracket property accessor*) hingegen nicht.» ersetzt. (Vermeidung geschachtelter Klammern)

        * «aufgefangen» wurde durch «behandelt» ersetzt. (einheitliche Formulierung)

B.5. Material und Methoden

- Material
  - Backend
    * «Backends» wurde in «Backend» geändert (es waren ursprünglich zwei Backends vorgesehen)
    * Web-Applikation
      · Die Fussnote für «DBMS» wurde zum ersten Vorkommen der Abkürzung verschoben.
  - Frontend
    * Testfälle
      · Der Satz «Ist keine Feature-ID angegeben – es gilt also `$this->feature->id` === `null` –, so wird der Dialog für das Hinzufügen konfiguriert, andernfalls für das Bearbeiten.» wurde in «Ist keine Feature-ID angegeben (es gilt also `$this->feature->id === null`), so wird der Dialog für das Hinzufügen konfiguriert, andernfalls für das Bearbeiten.» geändert. (Vermeidung mehrdeutiger Darstellung)
- Methoden
  - Untersuchungsgegenstand
    * Syntaxelemente
      · «aufgefangen» wurde durch «behandelt» ersetzt. (einheitliche Formulierung)

B.6. Resultate

- Sichere Versionen

  - «Basierend auf statistischen Erhebungen von Net Applications (Anhang A.3.1) wurden die folgenden getesteten Versionen von ECMAScript-Implementierungen als sicher eingestuft; d. h. die Bedeutung der entsprechenden Host-Umgebungen ist nach Ansicht dieses Autors zu gering, als dass die Testergebnisse für die korrespondierenden Versionen von Implementierungen in Erwägung gezogen werden müssten:» (Komma durch Semikolon ersetzt; Verbesserung des Leseflusses)

B.7. Diskussion

- Auswertung
  - «Rückschlüsseł» wurde in «Rückschlüsse» geändert (Schreibfehler)

B.8. Literaturverzeichnis

- Folgeseiten haben den korrekten Seitenkopf (`\markboth{...}` ergänzt)
- Die Namen der Autoren wurden in Kapitälchen gesetzt (`biblatex`-Befehle ergänzt)
- Lange URLs werden sauber umbrochen (`\sloppy` ergänzt)
- Der für “New in JavaScript 1.8.1” angegebene URL wurde durch den korrekten ersetzt (Copypaste-Fehler in BibTEX-Eintrag `mdn:js181`)

B.9. Anhang

- Abschnitte beginnen auf einer neuen Seite