



Acrobat JavaScript Scripting Guide

Technical Note #5430

バージョン：Acrobat 6.0




アドビ システムズ株式会社

Adobe Solutions Network

<http://partners.adobe.com/asn/japan/developer/benefits.jsp>

2003 年 5 月



Copyright 2003 Adobe Systems Incorporated. All rights reserved.

注意：ここに含まれるすべての記載情報は、Adobe Systems Incorporated（アドビシステムズ社）に所有権があります。この出版物（ハードコピー、電子データともに）はすべて、電子的、機械的、複写、録音、その他いかなる形式あるいは手段によっても、事前に発行者の文書による許可を受けることなく、複製または転写することはできません。

PostScript は、アドビシステムズ社の登録商標です。本文中で使用されている PostScript という名称はすべて、特に明記していない限り、アドビシステムズ社により定義された PostScript 言語を指します。PostScript という名称は、アドビシステムズ社が考案した PostScript 言語インタプリタ製品の製品商標としても使用されています。

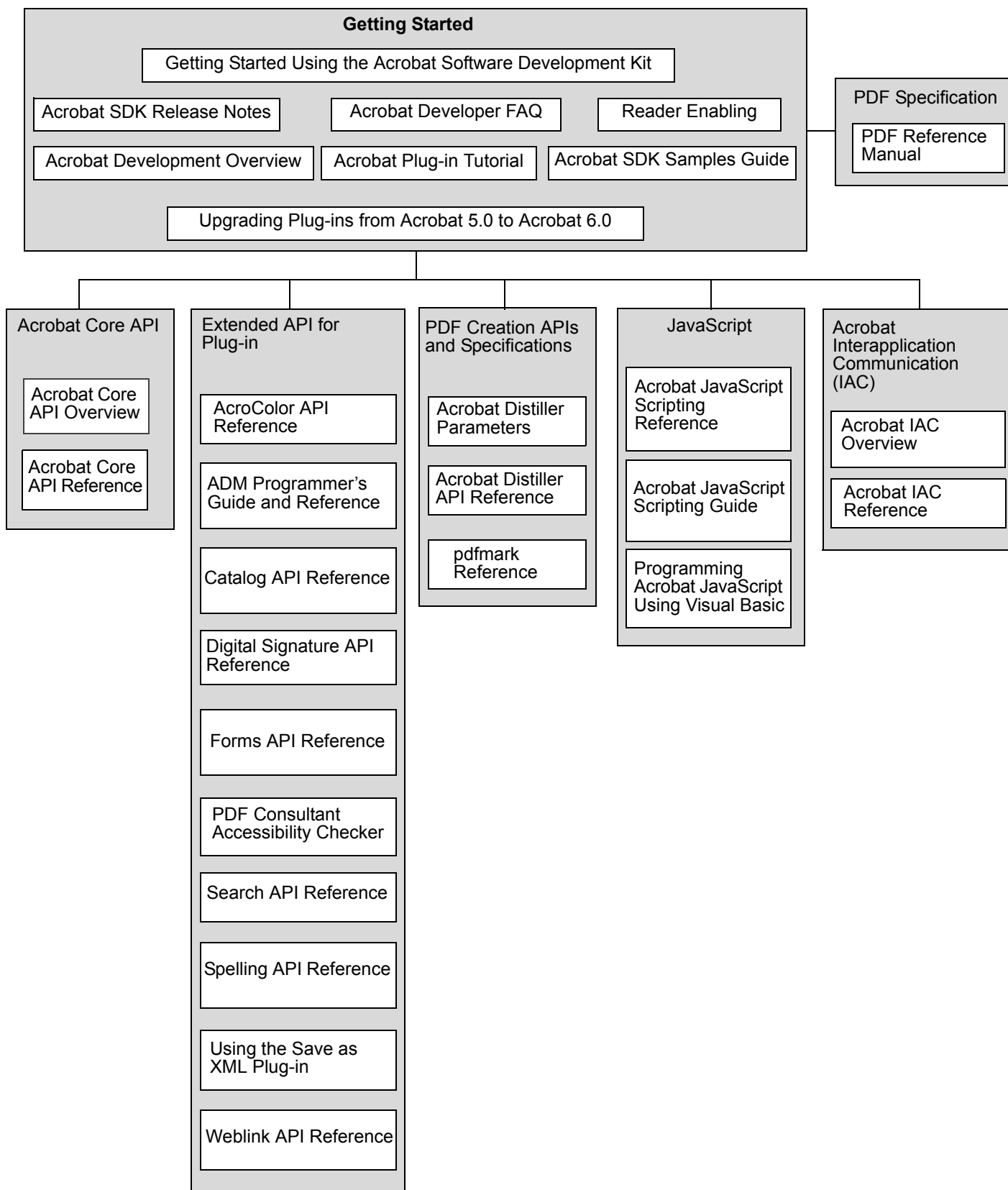
特に指定されている場合を除き、「PostScript プリンタデバイス」、「PostScript ディスプレイデバイス」または同様の記述は、アドビシステムズ社により考案またはライセンス化された PostScript 技術を含むプリンタデバイス、ディスプレイデバイス、またはその他の製品を指し、単に PostScript 言語と互換性があるだけのデバイスあるいはその他の製品を指すものではありません。

Adobe、Adobe ロゴ、Acrobat、Acrobat ロゴ、Acrobat Capture、Distiller、PostScript、PostScript ロゴおよび Reader は、アドビシステムズ社の米国およびその他の国における登録商標または商標です。

Apple、Macintosh、および Power Macintosh は、米国およびその他の国で登録された Apple Computer, Inc. の商標です。PowerPC は、IBM Corporation の米国における登録商標です。ActiveX、Microsoft、Windows、および Windows NT は、米国およびその他の国における Microsoft Corporation の登録商標または商標です。UNIX は、The Open Group の登録商標です。その他すべての商標は、それぞれの権利帰属者の所有物です。

この出版物およびその記載情報については、現時点の状況で提供されており、予告なしに変更される可能性があります。アドビシステムズ社は、この出版物に関して、間違いまたは不整合についての責任および義務を負いません。また、いかなる種類（明示的、非明示的、または法的）の保証も行いません。商業性、特別な用途への適合、および第三者の権利の非侵害に関しても、すべての保証を明示的に否認します。

Acrobat SDK Documentation Roadmap



目次

はじめに	9
はじめに	9
対象読者	9
目的と範囲	9
前提条件	10
構成	10
このマニュアルの使用法	10
このマニュアルの表記規則	11
その他のリソース	12
第 1 章 Acrobat JavaScript の概要	13
はじめに	13
この章の目的	13
内容	13
概要	13
Acrobat JavaScript とは	14
Acrobat JavaScript でできること	15
Acrobat JavaScript オブジェクトの概要	17
App オブジェクト	17
Doc オブジェクト	17
その他のよく使用される Acrobat JavaScript オブジェクト	18
データベースオブジェクト	19
JavaScript 言語に関する注意事項	19
第 2 章 Acrobat JavaScript エディタおよびデバッガコンソール	21
JavaScript エディタおよびデバッガコンソールの概要	21
この章の目的	21
内容	21
JavaScript コンソール	22
JavaScript コンソールを開く	22
JavaScript の実行	22
コードの整形	22
JavaScript エディタの使用	23
デフォルトの JavaScript エディタの指定	25
組み込みの Acrobat JavaScript エディタの使用	26
外部エディタの使用	26

エディタの付加機能	26
エディタに対する付加機能の指定	27
構文エラーの箇所でエディタを開けるかどうかの確認	28
実習：JavaScript コンソールの操作	29
JavaScript を使用可能にする	29
インタラクティブ JavaScript コンソールを使用可能にする	29
JavaScript コンソールを使用する	30
第 3 章 Acrobat JavaScript デバッガ	33
Acrobat JavaScript デバッガの概要	33
この章の目的	33
内容	34
Acrobat JavaScript デバッガを使用可能にする	34
デバッガダイアログウィンドウ	36
メインコントロールグループ	37
デバッガの表示ウィンドウ	37
デバッガのボタン	38
実行を再開	39
割り込み	39
終了	40
ステップオーバー	40
ステップイン	40
ステップアウト	40
デバッガのスクリプトウィンドウ	41
スクリプトウィンドウ内のスクリプトへのアクセス	41
PDF ファイルの内部にあるスクリプト	41
PDF ファイルの外部にあるスクリプト	42
コールスタックリスト	42
検査詳細ウィンドウ	43
詳細ウィンドウのコントロール	43
変数の検査	44
ウォッチ	45
ブレークポイント	45
デバッガの起動	47
実行の開始時点からのデバッグ	47
スクリプト内の任意の箇所からのデバッグ	47
コードのステップ実行	48
実習：電卓	48
電卓	49
はじめに	50
ランタイムエラーのデバッグ	51



もう1つのランタイムエラー	52
既知の問題	53
まとめ	54
第4章 フォームでの Acrobat JavaScript の使用	55
簡単な JavaScript の作成	55
自動日付フィールドの作成	55
算術計算の実行	56
「ページに移動」アクションの割り当て	57
電子メールで文書やフォームを送信する	58
一定の条件が満たされるまでフィールドを非表示にする	59
JavaScript アクションの操作	60
文書レベルの JavaScript アクションの操作	61
プログラムによるフォームフィールドの作成	62
ボタン	64
チェックボックス	66
コンボボックス	67
リストボックス	70
ラジオボタン	71
電子署名	71
テキスト	73
付録 A Acrobat JavaScript に関する簡単な FAQ	75
JavaScript はどこにあり、どのように使用するのですか？	75
フォルダレベルの JavaScript	75
文書レベル	75
フィールドレベル	76
フォームフィールドに名前を付けるときにはどのようにすればいいですか？	76
日付オブジェクトはどのように使用するのですか？	77
日付から文字列への変換	77
文字列から日付への変換	78
日付の演算	79
どうすれば文書を保護できますか？	80
文書に対するアクセス制限	80
権限の制限	80
電子署名	80
制約のある Acrobat JavaScript メソッドをユーザが使用できるようにするには どうすればよいですか？	81
署名フィールドに署名した後、どうすれば文書をロックできますか？	81
どうすれば文書へのアクセスを簡単にできますか？	82
文書のメタデータ	82

フィールドの説明	83
タブ順序の設定	83
読み上げ順序	83
どうすれば JavaScript でグローバル変数を定義できますか？	83
グローバル変数の永続化	83
どうすればフォームデータを電子メールアドレスに送信できますか？	84
どうすれば他のフィールドの値に応じてフィールドを非表示にできますか？	84
どうすれば別に開かれているフォームのフィールド値を現在のフォームから参照 できますか？	84
どうすればキー入力を 1 つずつインターセプトできますか？	85
どうすればネストされたポップアップメニューを作成できますか？	85
どうすれば独自の色を作成できますか？	85
どうすればユーザ入力を促すダイアログを表示できますか？	85
どうすれば JavaScript で URL を取り出せますか？	86
どうすればフィールドのホットヘルプテキストを動的に変更できますか？	86
どうすればズーム倍率をプログラムで変更できますか？	86
どうすればマウスが特定の領域に入った（あるいは出た）かを判断できますか？	86

はじめに

はじめに

Adobe Acrobat JavaScript Scripting Guide によろこそ。このスクリプティングガイドは、Adobe Acrobat 6 Pro に搭載されている JavaScript 開発環境を使用して Acrobat アプリケーションを開発したり拡張したりする方法を簡単にまとめたものです。

JavaScript 言語は、インタラクティブな Web ページをより簡単に作成できる言語として、Netscape Communications によって開発されました。Adobe は、この JavaScript を拡張して、そのインタラクティブな処理を PDF フォームに簡単に組み込めるようにしました。Acrobat フォームでは JavaScript を使用して、データのフォーマット、計算、検証を行ったり、アクションを割り当てたりすることができます。

JavaScript を組み込むレベルとしては、プラグインレベル、文書レベル、フィールドレベルの 3 種類がありますが、このマニュアルでは、文書レベルとフィールドレベルのスクリプトのみを扱います。

- プラグインレベルのスクリプトについて詳しくは、[60 ページの「JavaScript アクションの操作」](#)を参照してください。
- 文書レベルのスクリプトは、開いている文書で実行され、その文書のみ適用されます。
- フィールドレベルのスクリプトは、特定のフォームフィールドに関連付けられます。このタイプのスクリプトは、「マウスボタンを放す」などのイベントが発生したときに実行されます。

対象読者

このマニュアルは、Acrobat ソリューションのプロバイダや Acrobat のパワーユーザを対象読者としています（Adobe 製品を利用したソリューションの開発に関心をお持ちの方）。必要とする情報リソースを利用することができるのであれば、最低限の手助けで、新しい技術を効率良く短時間のうちに習得することができます。このマニュアルはそういったリソースのひとつです。

目的と範囲

このマニュアルの目的は、次のとおりです。

- Acrobat ソリューションの開発や導入に役立つ Adobe Acrobat JavaScript の機能の紹介。
- Acrobat JavaScript のスクリプティング機能や使用方法の詳細な解説。
- Acrobat JavaScript や関連技術に関する他のリソースの紹介。

このマニュアルを一通り読んで、用意されている実習を終えれば、Acrobat JavaScript を使えるようになります。開発を行う際は、このマニュアルを読み直したり、このマニュアルの内容を補う追加情報を該当する技術エリアで調べるなどしてください。

前提条件

このマニュアルは、Acrobat 6 の一般的なユーザインタフェースに読者が精通していることを前提としています。ユーザインタフェースについて詳しくは、Acrobat に付属しているオンラインヘルプを参照してください。このマニュアルでは一般的なユーザインタフェースについて説明している部分もありますが、主に JavaScript コードの作成に使用するユーザインタフェースに絞って解説しています。また、読者は標準的な JavaScript スクリプト言語の基本を習得している必要があります。このマニュアルに用意されている実習を行うためには、Acrobat 6 Pro が必要です。

構成

このマニュアルは、次の各章と付録で構成されています。

- 第 1 章 「Acrobat JavaScript の概要」
- 第 2 章 「Acrobat JavaScript エディタおよびデバッグコンソール」
- 第 3 章 「Acrobat JavaScript デバッグ」
- 第 4 章 「フォームでの Acrobat JavaScript の使用」
- 付録 A 「Acrobat JavaScript に関する簡単な FAQ」

このマニュアルの使用法

このマニュアルの第 2 章と第 3 章には実習が用意されており、Acrobat JavaScript を実際に動かしてみることができます。この実習を行う場合は、次の準備が必用です。

1. 使用する Windows または Macintosh ワークステーションに、Acrobat Pro がインストールされていることを確認します。実習は、特に注意書きがない限り、Windows 版および Macintosh 版の Acrobat で動作するように設計されています。
2. JavaScript の実習用ディレクトリを、ローカルのハードドライブに作成します。実習で使った PDF 文書などのファイルは、このディレクトリに保存します。
3. デバッグの章では、実習を始める前に、必要なファイルが入っている **.zip** ファイルを探して、その内容をローカルディレクトリに展開しておく必要があります。

注意： Macintosh で **.zip** ファイルの内容を展開するには、Stuffit Expander アプリケーションが必要です。



このマニュアルの表記規則

Acrobat マニュアルでは、次の表記規則に基づいて各書体を使用しています。

フォント	使用対象	例
固定幅	パスおよびファイル名	<code>C:¥templates¥mytmpl.fm</code>
	本文と区別するコード例	次は変数宣言です。 <code>AVMenu commandMenu,helpMenu;</code>
固定幅の太字	本文中のコードアイテム	<code>GetExtensionID</code> メソッド ...
	参照文書内のパラメータ名と文字値	<code>proc</code> が <code>false</code> を返すと、列挙が終了します。
固定幅の斜体字	擬似コード	<code>ACCB1 void ACCB2 ExeProc(void)</code> <code>{ do something }</code>
	コード例の中のプレースホルダ	<code>AFSimple_Calculate(cFunction, cFields)</code>
青字	Web ページへのライブラリンク	Acrobat Solutions Network の URL は、次のとおりです。 http://partners/adobe.com/asn/
	このマニュアル内の節へのライブラリンク	「Using the SDK」を参照してください。
	他の Acrobat SDK マニュアルへのライブラリンク	『Acrobat Core API Overview』を参照してください。
	このマニュアルおよびその他の Acrobat SDK マニュアルで説明されているコードアイテムへのライブラリンク	<code>ASAtom</code> が存在するかどうかテストします。
太字	PostScript 言語および PDF オペレータ、キーワード、ディクショナリキー名	<code>setpagedevice</code> オペレータ
	ユーザインタフェイス名	<code>ファイルメニュー</code>
斜体	PostScript 変数	<code>filename deletefile</code>

その他のリソース

このマニュアルでは、Acrobat JavaScript や関連技術に関して、次のリソースを参照しています。

- Acrobat[®] JavaScript Scripting Reference
このスクリプティングガイドと対を成すリファレンスです。このリファレンスでは、Acrobat JavaScript のすべてのオブジェクトが詳細に説明されています。
- Adobe[®] Acrobat[®] ヘルプ
このオンラインマニュアルは、Acrobat 6 に付属しています。
- [Netscape のディベロッパ Web サイト](http://devedge.netscape.com/)から入手できる JavaScript のリファレンス資料 (<http://devedge.netscape.com/>)
これは、次のマニュアルで構成されています。
 - Core JavaScript Guide
 - Core JavaScript Reference
- Acrobat eForms ソリューショントレーニング (<http://partners.adobe.com>)
- Acrobat レビューおよびマークアップトレーニング (<http://partners.adobe.com>)
- Portable Document Format (PDF) Reference、バージョン 1.4

このマニュアルでは、オンラインで表示される Acrobat SDK マニュアルは、ライブリンク（青字）になっています。ただし、リンクが正しく動作するためには、SDK と同じディレクトリ構造で、ローカルのファイルシステムにマニュアルをインストールする必要があります。これらのマニュアルは、SDK をインストールする際に自動的に配置されます。何らかの理由で SDK 全体をインストールせず、すべてのマニュアルがインストールされていない場合は、Adobe Solutions Network の Web サイトから必要なマニュアルを入手してください。そして、そのマニュアルを適切なディレクトリにインストールしてください。その際、このマニュアルの最初にある「Acrobat SDK Documentation Roadmap」が参考になります。

1

Acrobat JavaScript の概要

はじめに

この章では、Adobe Acrobat JavaScript の概要について説明します。デバッグ機能が新たに追加された Acrobat 6 の JavaScript 開発環境は、Acrobat Pro 版でのみサポートされています。

このマニュアルでは、顧客や社内のニーズを満たすソリューションを Acrobat JavaScript を使用して開発する方法を多数紹介しています。さらに詳しい情報や例については、[12 ページの「その他のリソース」](#)に記載されている、Web サイトなどのリソースを参照してください。

この章の目的

この章を終えると、次のことができるようになります。

- Acrobat JavaScript と標準的な (HTML) JavaScript との違いを理解する。
- Acrobat JavaScript で何ができるかを理解する。
- [Doc オブジェクト](#)の包含階層を理解し、**App**、**Doc**、**Console**、**Global**、**Util**、**Connection**、**Statement** の各オブジェクトの使用目的を理解する。

内容

トピック

[概要](#)

[Acrobat JavaScript とは](#)

[Acrobat JavaScript でできること](#)

[Acrobat JavaScript オブジェクトの概要](#)

概要

Adobe Acrobat は、電子文書の作成時の環境に縛られることなく、簡単かつ確実に電子文書を交換し表示できる媒体としてよく知られていますが、Adobe Acrobat でできるのは、文書の表示だけではありません。

Adobe Portable Document Format (PDF) 文書には、ユーザがデータを入力するためのフィールドや、アクションを実行するためのボタンを配置することができます。このような機能を追加した PDF 文書は、eForm と呼ばれます。eForm は、企業向けの自動文書処理ソリューションには欠かせません。このソリューションでは、従来の紙のフォームに代わって eForm が使用されます。従業員は、PDF ファイルを使用してフォームに記入し、提出することができます。

Acrobat では、チームによるオンラインレビュー機能もサポートされています。レビューの準備ができた文書は、Adobe PDF に変換します。レビュー担当者が Acrobat で Adobe PDF 文書を表示してコメントを記入すると、そのコメント（注釈）は文書のベース部分とは別のレイヤーとして追加されます。Acrobat では、テキスト、グラフィック、音声、動画などの、標準的に使用される各種のコメントが数多くサポートされています。文書のコメントを他のユーザ（文書の作成者や他のレビュー担当者など）と共有したい場合は、コメントレイヤーのみを別途コメントレポジトリに書き出すこともできます。

このような使い方をする場合、Acrobat JavaScript を使用して、PDF 文書の動作をカスタマイズしたり、Acrobat に搭載されていない機能を追加したり、PDF 文書の表示内容を変更したりすることができます。Acrobat JavaScript コードは、特定の PDF 文書、PDF 文書の特定のページ、または PDF ファイルのフィールドやボタンに関連付けることができます。エンドユーザが Acrobat を操作したり、JavaScript が含まれている PDF ファイルを Acrobat で表示して操作したりすると、その操作に応じて適切な JavaScript コードが実行されます。

カスタマイズできるのは、Acrobat での PDF 文書の動作だけではありません。Acrobat アプリケーション自身もカスタマイズすることが可能です。Acrobat の以前のバージョン（Acrobat 5 またはそれ以前）でこのようなカスタマイズを行おうとすると、C や C++ などの高級言語で Acrobat プラグインを作成するしかありませんでしたが、現在では、同じ機能のほとんどが Acrobat JavaScript の拡張機能を利用することで実現できます。Acrobat のユーザインタフェースにメニューを追加するなどのタスクを行う場合は、プラグインを作成するよりも、JavaScript スクリプトを作成するほうがはるかに簡単です。

Acrobat JavaScript とは

Acrobat JavaScript は、ISO-16262 の JavaScript バージョン 1.5（以前 ECMAScript と呼ばれていたもの）のコアをベースにしています。JavaScript は、Netscape Communications が開発したオブジェクト指向のスクリプト言語です。この言語は、Web ページの処理負荷を、サーバからクライアント上の Web ベース アプリケーションに移すために開発されたものです。Acrobat JavaScript は、この JavaScript プログラミング言語に、新しいオブジェクトとその関連メソッドおよびプロパティを追加したものです。この Acrobat 特有のオブジェクトを使用して PDF ファイルを操作することで、PDF ファイルとデータベースとの通信や、表示内容の変更などが行えます。Acrobat 特有のオブジェクトは、JavaScript のコアに追加した形になっているので、**Math**、**String**、**Date**、**Array**、**RegExp** などの標準のクラスは従来どおりにアクセスできます。

PDF 文書は汎用性が高く、Acrobat プラグインを使用することで Web ブラウザに表示することができますが、Acrobat JavaScript と、Web ブラウザで使用される JavaScript (HTML JavaScript) との間には相違点がありますので、注意してください。

- Acrobat JavaScript から、HTML ページのオブジェクトにアクセスすることはできません。同様に、HTML JavaScript から、PDF ファイルのオブジェクトにアクセスすることはできません。
- HTML JavaScript では、**Window** などのオブジェクトを操作できます。Acrobat JavaScript では、このオブジェクトにはアクセスできませんが、**Doc** や **annot** などの PDF 特有のオブジェクトを操作できます。

Acrobat JavaScript でできること

Acrobat JavaScript を使用すると、PDF 文書の中でさまざまなことが行えます。Adobe Solutions Network (ASN) では、Acrobat JavaScript でできるさまざまな処理を習得できる Adobe Acrobat JavaScript トレーニングコースを用意しています。このコースの資料は、次のサイトから PDF ファイルで入手できます。

<http://partners.adobe.com/asn/developer/training/acrobat/javascript/index.jsp>

ASN コースでは、次のような内容を扱っています。

- 計算の実行

PDF 文書にフィールドを作成して、数値データを収集することができます。スプレッドシートと同じように計算式を設定して、いくつかのフィールドの値に基づいて計算結果を得ることができます。詳しくは、ASN JavaScript トレーニングモジュールの「JavaScript Basics」を参照してください。

- ユーザアクションに対する応答

ユーザが PDF 文書进行操作するときには、マウスのクリック、テキストの入力、フィールドにフォーカスを合す／外すなどのアクションが行われます。これらのアクションやイベントに合わせて JavaScript コード（スクリプト）を実行するように指定することが可能です。Acrobat でこれらのアクションが検出されると、関連付けられたスクリプトが呼び出されます。詳しくは、ASN JavaScript トレーニングモジュールの「JavaScript Basics」および「Location Matters」を参照してください。

- ユーザデータの検証

eForm に入力されたデータが有効かどうかを確認できます。例えば、アプリケーションが扱える形式で日付が入力されたかどうかを確認したり、ある値がフォームに入力済みの他のデータと矛盾していないかどうかを検証したりすることができます。詳しくは、ASN JavaScript トレーニングモジュールの「Performing Validations and Calculations」を参照してください。

- Acrobat アプリケーションの変更

Acrobat のメニューやツールバーは変更することができます。JavaScript を使用することで、独自のメニュー項目を追加したり、ツールバーやメニューの表示 / 非表示を切り替えたりすることができます。詳しくは、ASN JavaScript トレーニングモジュールの「Location Matters」を参照してください。

- 文書の動作の制御

文書レベルのスクリプトを使用することで、文書が最初に開かれたときの動作を制御することができます。文書レベルのスクリプトでは、PDF の表示パラメータの設定など、ユーザの目に見えるような変更を行うことができます。また、ページレベルのスクリプトやフィールドレベルのスクリプトで使用する関数や変数を初期化することもできます。スクリプトは、ページを開く／閉じるなどのアクションに関連付けることもできます。詳しくは、ASN JavaScript トレーニングモジュールの「Location Matters」を参照してください。

- 文書の表示内容や機能の動的な変更

電子フォーム文書の大きな利点として、ユーザが入力したデータに応じて表示内容を動的に変更できることがあります。例えば、出張経費報告書の場合、費用を客先に請求できる場合

とそうでない場合とで、日当の限度額を変更することができます。このような変更の例としては、次のようなものがあります。

- フィールドのプロパティの変更（非表示、読み取り専用、必須、印刷しないなど）
- リストボックスやコンボボックスに表示する選択肢の変更
- フィールドやボタンに関連付けられているアクションの変更（新規 JavaScript の追加など）
- フィールドの動的な作成
- 注釈の生成

詳しくは、ASN JavaScript トレーニングモジュールの「Creating Fields with JavaScript」を参照してください。

- バッチシーケンスによる複数の PDF ファイルの処理

バッチシーケンスを使用することで、一連の PDF ファイルに対して JavaScript を実行することができます。例えば、コメントの抽出、スペルミスの検出、PDF ファイルの自動印刷など、さまざまなタスクを実行できます。処理対象のファイルは、バッチシーケンスを定義するときに指定しても構いませんし、シーケンスを実行する直前に指定しても構いません。詳しくは、ASN JavaScript トレーニングモジュールの「Batch Processing with Sequences」を参照してください。

- Acrobat のページテンプレートを使用した新規ページの動的な生成

テンプレートをベースにした新規ページを、ユーザの入力やアクションに基づいて JavaScript で作成できます。テンプレートは、既存の PDF 文書に動的に追加可能な、ページのひな形です。詳しくは、ASN JavaScript トレーニングモジュールの「Templates and JavaScript」を参照してください。

- ページの上にテンプレートを重ねて表示内容を変更

フォームの中に、表示したり隠したりするフィールドやボタンが多数存在している場合、そのようなページ要素は別のテンプレートに配置したほうがうまく制御できます。テンプレートは、テキスト、フィールド、ボタンといった変化する情報のひな形で、PDF 文書に動的に重ねることができます。PDF 文書の表示内容を変更するには、テンプレートに含まれているページ要素を、指定したページに JavaScript を使用して重ね合わせます。詳しくは、ASN JavaScript トレーニングモジュールの「Templates and JavaScript」を参照してください。

- データベースとのやり取り

Acrobat JavaScript には、ADBC（Acrobat 固有の ODBC の実装）を介してデータベースと通信するための専用オブジェクトが用意されています。JavaScript を使用することで、データベースからの値の読み出し、データの更新、新規データの挿入、データの削除などが行えます。詳しくは、ASN JavaScript トレーニングモジュールの「Integrating with a Database」を参照してください。

- コメントレポジトリの環境設定

Adobe Acrobat のチームによるレビュー機能を使用する場合は、コメントレポジトリの種類と場所を指定する必要があります。この設定は、コメントレポジトリの環境設定と呼ばれます。JavaScript 呼び出しを使用することで、特定の文書のコメントレポジトリを設定したり、Acrobat でレビューするすべての文書のコメントレポジトリを設定したりすることができます。

す。詳しくは、ASN JavaScript トレーニングモジュールの「Setting up a Comment Repository」を参照してください。

Acrobat JavaScript オブジェクトの概要

Acrobat JavaScript には、Acrobat アプリケーション、PDF 文書、および PDF 文書のフィールドやボタンを操作するためのオブジェクトがいくつか定義されています。ここでは、比較的よく使用されるオブジェクトを紹介します。

App オブジェクト

App オブジェクトは、Acrobat アプリケーション自身を表す静的なオブジェクトです。このオブジェクトには、Acrobat 固有の関数に加えて、さまざまなユーティリティ関数や便利な関数が定義されています。**App** オブジェクトを操作することで、現在開いているすべての PDF 文書を取得したり、メニューやメニュー項目を追加して Acrobat をカスタマイズしたりすることができます。また、エンドユーザが使用している Adobe 製品のタイプ（Reader、Approval、完全な Acrobat など）やバージョンを **App** に問い合わせることもできます。

Doc オブジェクト

Acrobat は PDF 文書を操作するためのアプリケーションですので、PDF 文書を操作するためのオブジェクトが用意されています。これは、**Doc オブジェクト**と呼ばれます。**Doc オブジェクト**は、ビューアで開かれている PDF ドキュメントと JavaScript インタプリタとを結ぶ、インタフェイスの役割を果たします。**Doc オブジェクト**を操作することで、その文書に関する一般情報を取得したり、文書内を移動したり、文書内の他のオブジェクトにアクセスしたりすることができます。オブジェクトの多くは、PDF 文書内のアイテム（ブックマーク、フィールド、テンプレート、注釈、サウンドなど）を表します。

注意： HTML JavaScript のオブジェクト階層に慣れている方は、その階層が包含階層であり、継承階層でないことをご存知だと思いますが、Acrobat JavaScript の場合もまったく同じです。オブジェクトは、上位オブジェクトのプロパティやメソッドを継承しません。同様に、オブジェクト間でメッセージが自動的に渡されることは、どの方向についてもありません。

図 1.1 Doc オブジェクトの包含階層

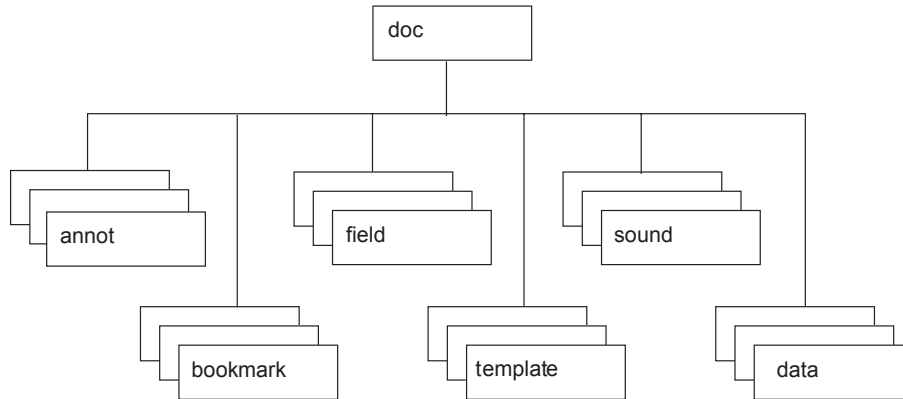


図 1.1 に、Doc オブジェクトに関連するオブジェクトの包含階層を示します。

JavaScript から Doc オブジェクトにアクセスするには、さまざまな方法がありますが、**this** オブジェクトを使用するのが最も一般的です。このオブジェクトは、通常、基盤となる現在のドキュメントの Doc オブジェクトを指しています。

その他のよく使用される Acrobat JavaScript オブジェクト

PDF 文書に直接関係しないオブジェクトもいくつかあります。**App** の他にも、**Console**、**Global**、**Util** など、プログラマの役に立つオブジェクトが用意されています。

Console

Console オブジェクトは、JavaScript コンソールにアクセスするための静的なオブジェクトで、デバッグメッセージを表示したり、JavaScript を実行したりすることができます。このオブジェクトは、Adobe Reader や Acrobat Standard では動作しません。**Console** オブジェクトは、デバッグの補助ツールとして使用したり、対話的にコードを試してみる場合に使用したりすることができます。

Dbg

Dbg オブジェクトは、Acrobat Pro でのみ使用可能なオブジェクトです。これを使用することで、コマンドラインコンソールからデバッガを制御することができます。JavaScript デバッガダイアログボックスのツールバーにボタンとして用意されている基本機能は、すべて **Dbg** オブジェクトのメソッドで実行できます。また、**Dbg** オブジェクトを使用してブレークポイントの設定、削除、検査を行うこともできます。

Global

Acrobat を次回起動したときにも利用するような永続的データの保存には、**Global** オブジェクトを使用します。また、バッチシーケンスなどで、文書のグループ全体に関する情報を保存する場合にも使用できます。例えば、バッチシーケンスのコードでは、処理する文書の合計数は **Global** のプロパティに保存することがよくあります。また、文書に関する情報を **Report** オブジェクトに保存する必要がある場合には、そのオブジェクトを **Global** のプロパティに割り

当てて、アクセスできるようにします。**Global** オブジェクトの使用については、ASN JavaScript トレーニングモジュールの「Location Matters」および「Batch Processing with Sequences」を参照してください。

Util

Util オブジェクトは静的な JavaScript オブジェクトで、数多くのユーティリティメソッド、および文字列と日付のフォーマットや解析を行う便利な関数が定義されています。

データベースオブジェクト

ADBC、**Connection**、**Statement** の各オブジェクトを使用することで、データベースとやり取りを行うことができます。これらの Acrobat 特有の JavaScript オブジェクトを使用して、Acrobat Database Connectivity (ADBC) を構成し、ODBC 呼び出しを使用してデータベースへの接続を確立し、データにアクセスします。開発者は、SQL ステートメントを使用して、データの挿入、更新、取得、削除を行えます。SQL ステートメントは、**Statement** オブジェクトの **execute()** メソッドに渡すだけで実行できます。これらのオブジェクトについては、ASN JavaScript トレーニングモジュールの「Integrating with a Database」を参照してください。

JavaScript 言語に関する注意事項

ここで、JavaScript を始めたばかりのプログラマが起こしやすい問題を 1 つ挙げておきます。この問題は、JavaScript 言語の 2 つの特徴が組み合わさることで発生します。1 つ目の特徴は、JavaScript は大文字と小文字が区別される言語だということです。例えば、「**fillColor**」と「**fillcolor**」は別のプロパティになります。もう 1 つの特徴は、新しいプロパティ名に値を割り当てるだけで、作業中のオブジェクトにローカルなプロパティが新たに作成されるということです。この 2 つの特徴が組み合わさると、なかなか発見できないバグが発生することがあります。プロパティに値を割り当てるときに、プロパティ名の大文字／小文字を間違えると、新しいプロパティが作成されるだけで、目的の値は変更されず、元の値のままになります。

2

Acrobat JavaScript エディタおよびデバッグコンソール

JavaScript エディタおよびデバッグコンソールの概要

Acrobat には、Acrobat JavaScript 機能を実装したりテストしたりするための開発環境が用意されています。Windows システムと Macintosh システムのどちらのコード開発環境にも、Acrobat の組み込みエディタが用意されています。また、サードパーティ製のエディタを使用するように選択することもできます。この章では、Acrobat JavaScript デバッグコンソールと組み込みの JavaScript エディタを使用する方法を説明し、短いスクリプトを評価するための簡単な方法を紹介します。

Acrobat デバッガの全機能については、[第 3 章「Acrobat JavaScript デバッガ」](#)で紹介しています。ここでは、ブレークポイントの設定、変数の検査、コードのステップ実行などの高度なデバッグ作業が説明されています。

この章の目的

この章を終えると、次のことができるようになります。

- コードの作成に使用するエディタの種類を指定する。
- Acrobat でサポートされている、外部エディタの付加機能を理解する。
- JavaScript コンソールを起動して、対話的にコードの実行および出力ステートメントの表示を行う。

内容

トピック	実習
JavaScript コンソール	実習：JavaScript コンソールの操作
JavaScript エディタの使用	
デフォルトの JavaScript エディタの指定	
組み込みの Acrobat JavaScript エディタの使用	
外部エディタの使用	

JavaScript コンソール

Acrobat JavaScript コンソールは、JavaScript コードのテストやデバッグが行えるインタフェースです。コンソール上での編集が可能で、インタラクティブな処理が行えます。また、オブジェクトのプロパティやメソッドを、コードの中で使用する前に、コンソールで試してみることもできます。コンソールをエディタとして使用して、コード内の行やブロックを対話的に実行することもできます。

`Console` オブジェクトを使用すると、JavaScript コンソールをコードの中で操作することができます。`Console` は静的オブジェクトなので、使用するためにインスタンスを作成する必要はありません。`Console` の `println()` メソッドを使用して、デバッグ情報を表示できます。コンソールの表示、非表示、クリアなどを行うメソッドも用意されています。

JavaScript コンソールを開く

Acrobat アプリケーションの Acrobat JavaScript コンソールは、次のようにして開くことができます。

1. 次のいずれかの方法で、デバッグウィンドウを開きます。
 - **アドバンスト / JavaScript / デバッグ**メニューコマンドを選択します (Windows および Macintosh)。
 - `Ctrl+j` を押します (Windows)。
 - `Control+j` を押します (Macintosh)。
2. デバッグの表示リストで「**コンソール**」または「**スクリプトとコンソール**」を選択します。プログラムからコンソールを開く場合は、`console.show()` を使用します。

JavaScript の実行

JavaScript コードの開発時にテストを行う方法の 1 つとして、JavaScript コンソールで評価を行う方法があります。JavaScript コードをコンソールで評価する場合、基本的に次の 2 通りの方法があります。

- 1 行のコードを評価する場合は、評価したい行にカーソルを置き、数値キーパッドの `Enter` キー (または標準キーボードの `Ctrl+Enter` キー) を押すと、評価結果が得られます。
- コードブロックを評価する場合は、入力したコードのブロック全体を選択し、数値キーパッドの `Enter` キー (または標準キーボードの `Ctrl+Enter` キー) を押します。

コードの評価結果は、コンソールに直ちに表示されます。コードブロックを評価する場合は、ブロック内の最後の式の結果しかコンソールに表示されないのに注意してください。

コードの整形

JavaScript コンソールでは、`Tab` キーを使用してコードを整形することができます。

- カーソル位置に 4 文字分の空白を挿入するには、`Tab` キーを押します。

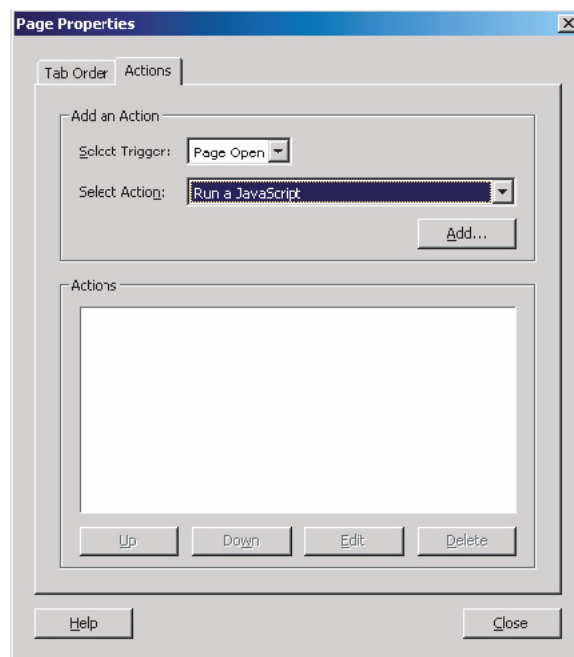
- カーソルを左に 4 文字分移動するには、Shift+Tab キーを押します。(左端までに 4 文字分のスペースがない場合は、行の先頭に移動します)。
- 行またはコードブロック全体を右に 4 文字分移動するには、コードまたは行の一部を選択して Tab キーを押します。
- 行またはコードブロック全体を左に 4 文字分移動するには、コードまたは行の一部を選択して、Shift+Tab キーを押します (左側に 4 文字のスペースがない場合は、可能なだけ左に寄ります)。

JavaScript エディタの使用

オブジェクトの特定のイベントに対して JavaScript を定義したり、定義されている JavaScript を編集するときには、JavaScript コンソールではなく専用の JavaScript エディタを使用するのがふつうです。このエディタを開く一般的な手順は、次のとおりです。

1. JavaScript コードのアクションを関連付けるオブジェクト (フィールド、リンク、ブックマーク、ページなど) を選択します。
2. そのオブジェクトのプロパティダイアログボックスを開きます。図 2.1 にその例を示します。

図 2.1 ページのプロパティ

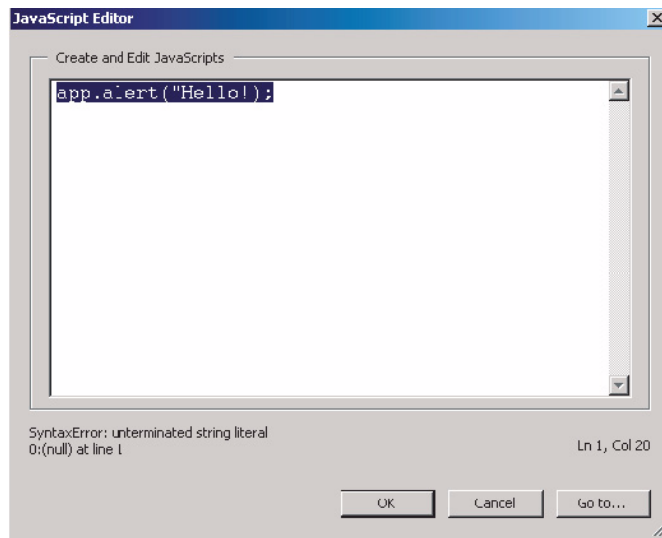


3. JavaScript アクションを実行するためのトリガ (「マウスボタンを放す」や「ページを開く」など) を選択します。
4. 「アクションを選択」ドロップダウンリストから「JavaScript を実行」を選択します。
5. 「追加」をクリックして、JavaScript エディタを開きます。

6. ユーザがページを開いたときに実行する JavaScript スクリプトを、エディタウィンドウに入力します。
7. 完了したら「OK」をクリックしてエディタを閉じます。

コードにエラーがある場合は、[図 2.2](#) に示すように、問題のコード行が強調表示され、エラーメッセージが表示されます。

図 2.2 JavaScript エディタによって検出されたエラー



[図 2.2](#) の場合は、文字列の右側に引用符が抜けています。

重要なメモ

JavaScript アクションは、JavaScript エディタを使用して、PDF 文書内のさまざまな場所（「レベル」）のオブジェクトに関連付けることができます。例えば、フォーム内の特定のフィールドや、文書レベルにスクリプトを関連付けることができます。文書レベルに関連付けると、文書内のスクリプト可能なあらゆる場所から、そのスクリプトを使用できます。エディタを開く手順は、スクリプトを実行する場所によって異なります。

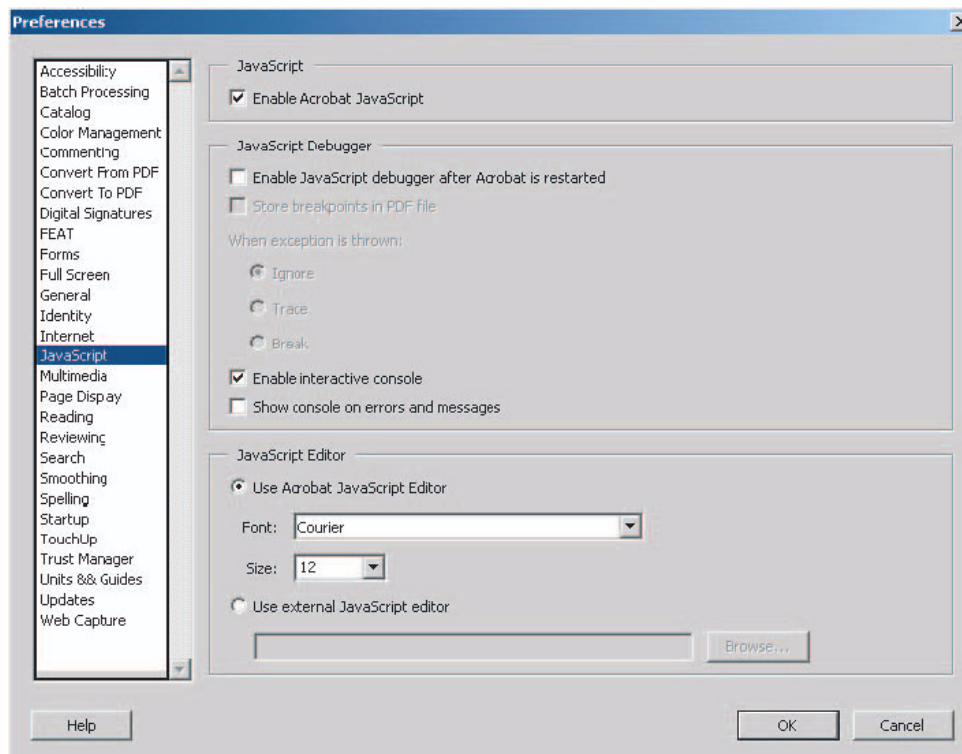
PDF 文書内のさまざまな場所、およびアプリケーションレベルで JavaScript エディタを開いて使用する方法について詳しくは、ASN JavaScript トレーニングモジュールの「Location Matters」を参照してください。

デフォルトの JavaScript エディタの指定

エディタは、Acrobat に付属している組み込みの JavaScript エディタを使用することもできますし、外部の JavaScript エディタを使用することもできます。デフォルトの JavaScript エディタを設定するには、次のようにします。

1. **編集／環境設定**を選択して、**環境設定**ダイアログボックスを開きます。
2. ダイアログボックスの左側のオプションリストから「**JavaScript**」を選択します。図 2.3 に示すような**環境設定**ダイアログが表示されます。
3. 「**JavaScript エディタ**」セクションで、使用するエディタを選択します。

図 2.3 環境設定でのエディタの選択



「**Acrobat JavaScript エディタ**」オプションを選択すると、組み込みの Acrobat JavaScript エディタがデフォルトに設定されます。

「**外部 JavaScript エディタ**」オプションを選択すると、外部エディタがデフォルトに設定されます。外部エディタを使用する場合は、エディタへのパスをテキストボックスに入力する（または貼り付ける）か、「**参照 ...**」をクリックして使用する JavaScript エディタを指定します。

注意：参照によって外部エディタを指定した場合には、エディタコマンドにコマンドオプションが自動的に追加されます。詳しくは、26 ページの「**エディタの付加機能**」を参照してください。

4. 「**OK**」をクリックして、**環境設定**ダイアログボックスを閉じます。

組み込みの Acrobat JavaScript エディタの使用

組み込みの Acrobat JavaScript エディタでも、Acrobat JavaScript コンソールと同じように、JavaScript コードを評価できます。評価する行またはコードブロックを選択し、数値キーパッドの Enter キーまたは標準キーボードの Ctrl+Enter キーを押すと、評価が行われます。

このエディタは、デバッガコンソールと同じように機能します。ただし、コードの評価結果は、エディタではなくコンソールウィンドウに表示されます。これは、エディタに入力したコードと表示される評価結果を区別するためです。

組み込みの JavaScript エディタでコードを整形する方法は、コンソールウィンドウの場合と同じです。詳しくは、[22 ページの「コードの整形」](#)を参照してください。

外部エディタの使用

JavaScript を編集するためのデフォルトのエディタとして外部エディタを指定した場合、Acrobat で JavaScript を編集する必要があるときには、常にそのエディタが使用されます。その際、Acrobat によって一時ファイルが生成され、そのファイルが外部エディタで開かれます。外部エディタでファイルを編集した場合、変更内容を Acrobat に認識させるためには、ファイルを保存する必要があります。また、外部エディタでファイルを編集している間は Acrobat にアクセスできません。外部エディタでは、Acrobat JavaScript コードを評価することはできません。外部エディタを閉じると、再び Acrobat にアクセスできるようになります。

エディタの付加機能

Acrobat では、Windows 用エディタの付加機能を利用できます（ただし、コマンドラインで指定可能な場合に限られます）。Acrobat でサポートしているのは、ファイル名 (%f) と対象の行番号 (%n) の 2 つのコマンドラインパラメータです。Macintosh 用エディタのパラメータは、サポートされていません。

エディタの機能の中で特に重要なのは、新しい編集セッションを開始するたびにエディタの新しいインスタンスを起動させる機能です。エディタのインスタンスが既に起動しているときに新しいファイルを読み込む必要がある場合、多くのエディタでは、その実行中のインスタンスでファイルを読み込むのがデフォルトの動作になっています。このような場合に新しくセッションを開くと、既存のセッションは閉じられてしまいます。ファイルに加えた変更内容は、保存しない限り Acrobat には認識されませんので、そのセッションで保存されていない変更内容は失われてしまいます。このような事態を避けるためには、新しい編集セッションを開始する前にエディタを閉じるように心がけるか、または常に新しいインスタンスを起動するようにエディタに指示する必要があります。

エディタの内部設定で、常に新しいインスタンスを起動するように設定できる場合は、そのオプションを設定してください。エディタの新しいインスタンスを起動するためにコマンドラインパラメータを使用する必要がある場合は、**編集/環境設定/JavaScript ダイアログ**のエディタコマンドラインに、そのパラメータを追加できます。これで、編集内容が失われることはなくなります。

Acrobat では、行番号をパラメータ (%n) としてコマンドラインに挿入することができます。開始行番号をコマンドラインで指定できるエディタの場合は、構文エラーが見つかった行でエディタを開始することができます。

Acrobat には、現在普及している多くの外部エディタ用の、定義済みのコマンドラインテンプレートが組み込まれています。外部エディタの設定は、**編集／環境設定／JavaScript** で定義します。定義済みのコマンドラインテンプレートが用意されている外部エディタを「参照」ボタンで指定すると、コマンドラインフィールドにそのテンプレートの内容が入力されます。入力されたコマンドラインオプションは、必要に応じて編集することができます。定義済みテンプレートが用意されていないエディタを使用する場合は、必要に応じて適切なコマンドラインパラメータをコマンドラインフィールドに指定してください。

エディタに対する付加機能の指定

Acrobat では、CodeWrite、Emacs、SlickEdit などのいくつかのエディタについて、前述の 2 つのコマンドをサポートしています。

Acrobat で現在サポートされていないエディタを使用する場合は、そのエディタの Web サイトにアクセスしてエディタのマニュアルを参照するか、そのエディタに付属しているマニュアルを参照してください。エディタの機能を確認するために、次の事項を調べてください。

- 常に新しいインスタンスを開くように指示するコマンドスイッチは何ですか？スイッチはエディタによって異なります。例えば、`/NI` や `+new` の後ろにファイル名を指定する場合があります。ファイル名は `%f` で表されます。ファイルやフォルダの名前にはスペースが含まれていることがありますので、正しく処理されるように `%f` は引用符で囲んでください。
- ファイルを開いたときに特定の行番号にジャンプするように指示する方法が用意されていますか？

行番号を指定するコマンドスイッチとしては、`-#`、`-L`、`+`、`-l` などがあります。これらの後に、行番号を表す `%n` を指定します。

ジャンプする行がない場合は、`%n` の値がヌルになることがあります。したがって、Acrobat では、コマンドのその部分をオプションにして、ジャンプ先の行番号が存在する場合にのみ挿入されるようにする仕組みが用意されています。コマンドスイッチやパラメータを角かっこ (`[...]`) で囲むと、その部分は行番号パラメータが使用される場合にのみ挿入されます。ヌルの行番号を適切に処理できないエディタの場合は、行番号スイッチと `%n` パラメータを角かっこで囲んでください。

コマンドラインで、`%`、`[`、`]` を文字として指定する必要がある場合は、それぞれ `%%`、`%[`、`%)` と入力してください。

例えば、`vs.exe` は Visual SlickEdit エディタとして認識され、コマンドラインに次のように入力されます。

```
"C:¥ProgramFiles¥vslick¥win¥vs.exe" "%f" +new [-#%n]
```

参照ダイアログで「開く」をクリックすると、**編集／環境設定／JavaScript** ダイアログにこのように表示されます。Acrobat でエディタが正しく認識され、テンプレートが入力されるためには、インストール済みのエディタを参照ダイアログで指定する必要があります。

Acrobat から JavaScript エディタを開く必要が生じた場合には、コマンドラインの中で適切な置換が行われた後に、オペレーティングシステムのシェルによってそのコマンドラインが実行されます。前述の例で、構文エラーが 43 行目に見つかった場合には、次のようなコマンドラインが生成されます。

```
"C:¥Program Files¥vslick¥win¥vs.exe" "C:¥Temp¥jsedit.js" +new -#43
```

表 2.1 JavaScript の外部エディタとしてサポートされ、コマンドラインテンプレートが用意されているエディタ

エディタ	Web サイト	テンプレートコマンドラインの引数
Boxer	http://www.boxersoftware.com	-G -2 "%f" [-L%n]
ConTEXT	http://fixedsys.com/context	"%f" [/g1:%n]
CodeWright	http://www.codewright.com	-M -N -NOSPLASH "%f" [-G%n]
Emacs	http://www.gnusoftware.com/Emacs	[+%n] "%f"
Epsilon	http://www.lugaru.com	[+%n] "%f"
Multi-Edit	http://www.multiedit.com	/NI /NS /NV [/L%n] "%f"
TextPad	http://www.textpad.com	-m -q "%f"
UltraEdit	http://www.ultraedit.com	"%f" [-l%n]
VEDIT	http://www.vedit.com	-s2 "%f" [-l %n]
Visual SlickEdit	http://www.slickedit.com	+new "%f" [-#%n]

構文エラーの箇所でエディタを開けるかどうかの確認

TextPad などの一部のエディタでは、行番号を指定してエディタを開く機能がサポートされていません。この機能がサポートされているかどうか確認するには、次のようにします。

1. 構文エラーのないスクリプトをエディタで開きます。
2. 構文エラーを付け加えます。
3. 構文エラーのない行にカーソルを移動します。
4. ファイルを閉じて保存します。
 構文エラーの修正を求めるダイアログが表示されるか確認します。
 表示された場合は、構文エラーを含む行が正しく示されているか確認します。

構文エラーがあるファイルを保存して閉じる

構文エラーが含まれているファイルを保存して閉じると、エラーを修正するかどうかを確認する次のメッセージがダイアログボックスに表示されます（ここでは 123 行目にエラーがあるとしません）。

JavaScript エラー：行 123

修正しますか。

注意：「いいえ」をクリックすると、ファイルは破棄されます。

必ず「はい」をクリックしてください。Acrobat はエディタへのパスを展開し、指定された構文に従って行番号を挿入します。次にエディタが開き、カーソルが 123 行目に置かれます。

実習：JavaScript コンソールの操作

この実習を行うためには、コンピュータに Acrobat 6 Pro がインストールされている必要があります。

この実習では、Acrobat で JavaScript が使用可能になっていることを確認した後、Acrobat JavaScript コンソールを使用してコードの編集と評価を行います。

この実習を終えると、次のことができるようになります。

- Acrobat JavaScript を使用可能／使用不能にする。
- JavaScript デバッガを使用可能／使用不能にする。
- デバッガでコンソールを開く。
- コンソールウィンドウでコードを評価する。

JavaScript を使用可能にする

Acrobat で JavaScript アクションを作成および使用するためには、JavaScript を使用可能にする必要があります。Acrobat で JavaScript が使用可能になっているかどうか確認するには、次のようにします。

1. Acrobat アプリケーションを起動します。
2. **編集／環境設定**を選択して、**環境設定**ダイアログボックスを開きます。
3. ダイアログボックスの左側のオプションリストから「**JavaScript**」を選択します。
4. 「**Acrobat JavaScript を使用**」が選択されていない場合は、選択します。
5. この後、JavaScript デバッガとデバッグコンソールを使用可能にしますので、**環境設定**ダイアログボックスは開いたままにしておいてください。

インタラクティブ JavaScript コンソールを使用可能にする

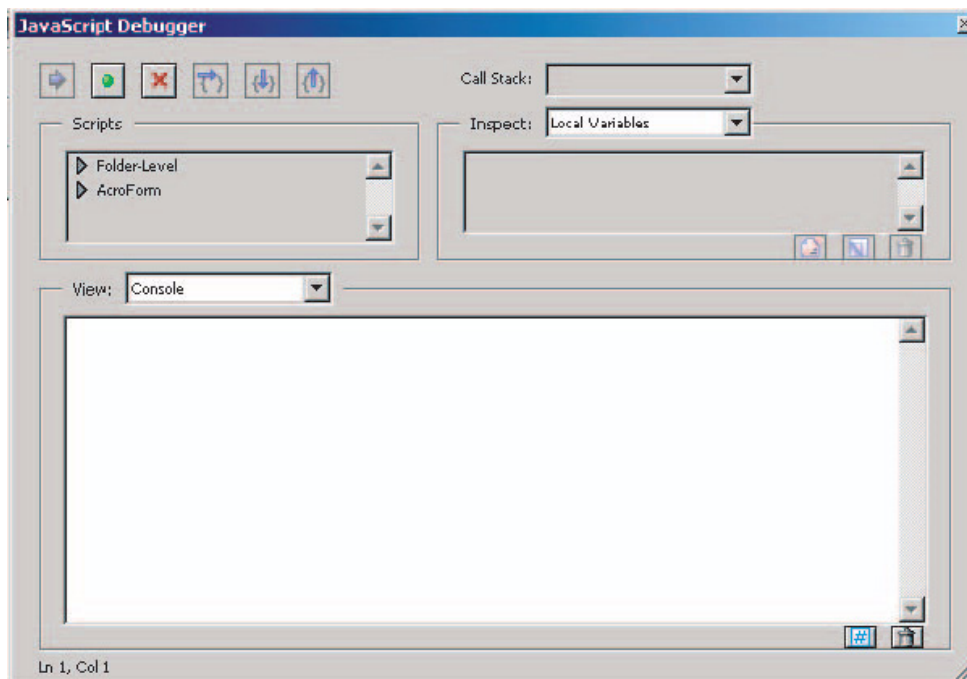
コンソールウィンドウは、JavaScript デバッガの中に含まれています。コンソールを使用するためには、デバッガを使用可能にする必要があります。

1. **環境設定**ダイアログの「JavaScript デバッガ」で、「**Acrobat を再起動した後で JavaScript デバッガを使用**」を選択します。
2. 「**インタラクティブコンソールを使用**」を選択します。このオプションを選択すると、コンソールウィンドウに入力したコードを評価できるようになります。
3. 「**エラーとメッセージをコンソールに表示**」を選択します。エラーがあった場合には、修正に役立つ情報がコンソールに表示されます。
4. 「**OK**」をクリックして、環境設定ダイアログを閉じます。
5. Acrobat をいったん閉じて、再起動します。

JavaScript コンソールを使用する

1. アドバンスト／JavaScript／デバッガ (Ctrl+j) を選択して、JavaScript デバッガを開きます。
2. デバッガの表示リストで「コンソール」を選択します。
コンソールウィンドウが、[図 2.4](#) のように表示されます。

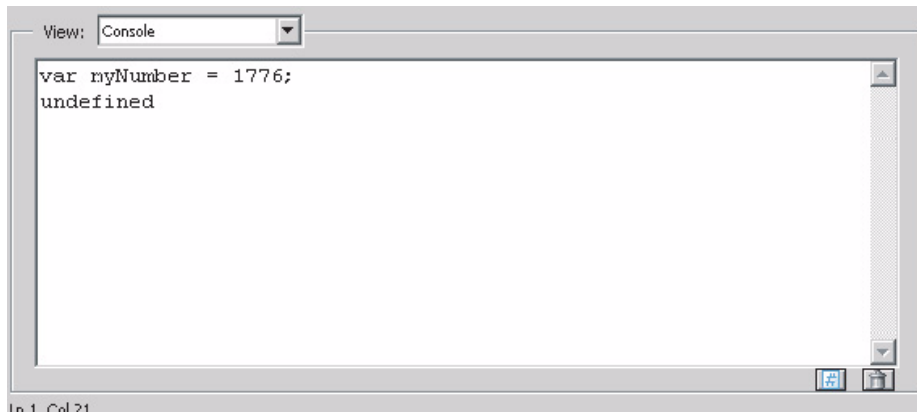
図 2.4 デバッガのコンソールウィンドウ



3. コンソールウィンドウのすぐ下にあるクリアボタン（ゴミ箱アイコン）をクリックして、ウィンドウの表示内容を削除します。
4. コンソールに、次のコードを入力します。

```
var myNumber = 1776;
```
5. このコード行のどこかにマウスカーソルを置き、数値キーパッドの Enter キーまたは標準キーボードの Ctrl+Enter キーを押してコードを評価します。結果が、[図 2.5](#) のように表示されます。

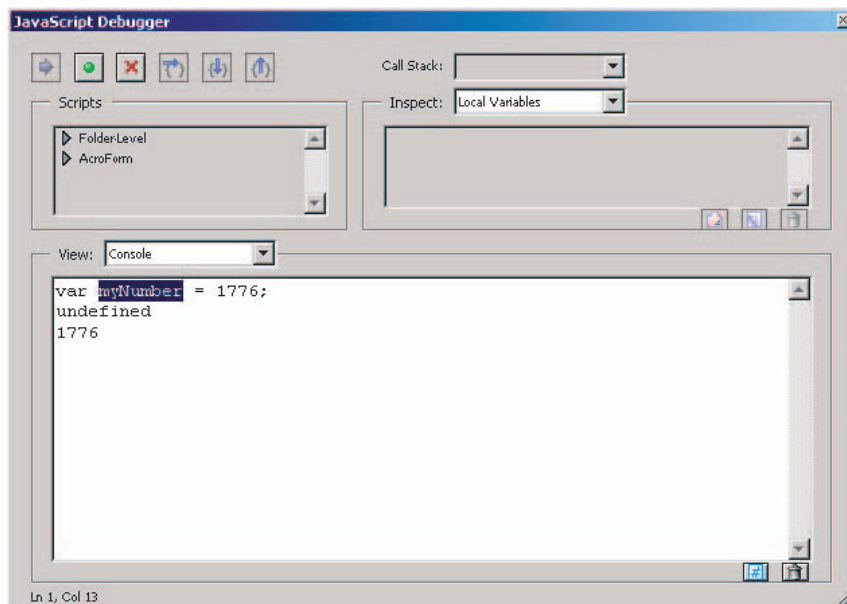
図 2.5 変数宣言の評価



コンソールウィンドウには、変数宣言の戻り値である **undefined** が表示されます。ここで、式の評価結果が、その式で設定した変数値と同じでないことに注意してください。この場合、戻り値の **undefined** は、「**myNumber** の値が定義されていない」ということを意味しているわけではありません。**undefined** を返しているのは式全体です。

6. **myNumber** 変数だけ（JavaScript ステートメントの一部）をコンソールウィンドウで評価するには、変数名を強調表示して、数値キーパッドの Enter キーまたは Ctrl+Enter キーを押します。結果が図 2.6 のように表示されます。

図 2.6 myNumber の評価

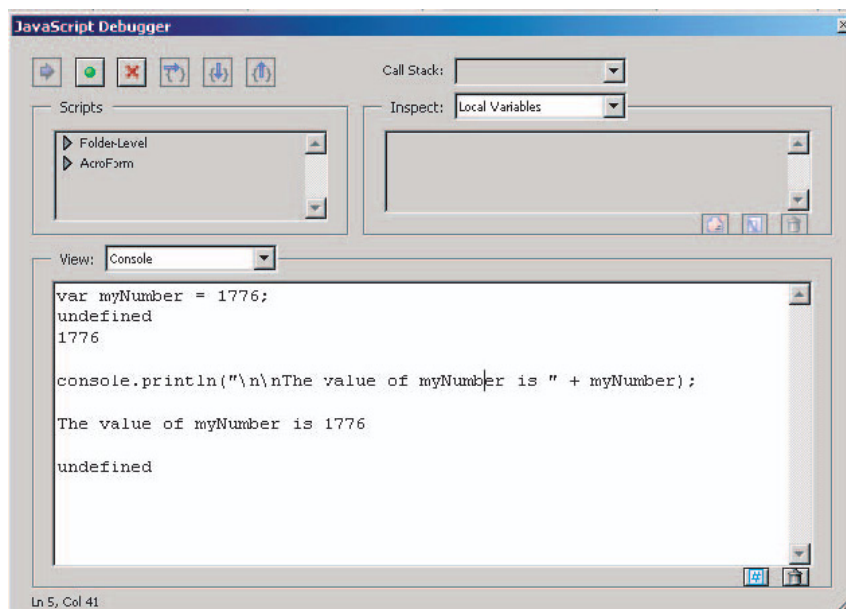


7. コンソールに現在表示されている内容の下に、次のコード行を入力します。入力したコードにカーソルを置き、数値キーパッドの Enter キーを押して評価します。

```
console.println("%n%nThe value of myNumber is " +  
myNumber);
```

C スタイルの書式文字 `\n` で強制改行を行っていることに注意してください。コンソールウィンドウに、[図 2.7](#) のような結果が表示されます。

図 2.7 コード行の評価



JavaScript コードで `console.println()` メソッドを使用すると、デバッグ情報などのメッセージをコンソールに表示できるので、大変便利です。このメソッドは、指定されたテキストを出力し、`undefined` を返します。

8. 閉じるボタンをクリックして、デバッガダイアログを閉じます。

3

Acrobat JavaScript デバッガ

Acrobat JavaScript デバッガの概要

Acrobat では、Acrobat JavaScript デバッガを使用して、JavaScript コードをデバッグすることができます。この JavaScript デバッガには、必要な機能が完備されており、ブレークポイントを設定したり、コードをステップ実行しながら変数値を検査したりすることができます。このデバッガは、Adobe Reader では利用できません。

デバッグ操作は、**アドバンスト / JavaScript / デバッガ**で表示されるダイアログで行います。実行中のスクリプトで例外が発生した場合や、設定されたブレークポイントに達した場合には、デバッグ処理を行えるように、デバッガダイアログが自動的に開きます。ユーザが手動で開く必要はありません。

注意： Web ブラウザ（Netscape や Internet Explorer など）で表示する HTML ページに含まれている JavaScript などのスクリプト言語は、このデバッガでデバッグすることはできません。

この章の目的

この章を終えると、次のことができるようになります。

- デバッガの各コントロールの使用方法を理解する。
- スクリプトの先頭または任意の箇所からデバッガを起動する。
- 対話的にコードを実行し、コンソールウィンドウに出力を表示する。
- 独自のウォッチおよびブレークポイントを設定する。
- 変数の詳細を検査する。
- 1つのデバッグセッションを終了した後、同じデバッガダイアログで次の新しいセッションを開始する。

内容

トピックおよび実習

[Acrobat JavaScript デバッグを使用可能にする](#)[デバッグダイアログウィンドウ](#)[デバッグのボタン](#)[デバッグのスクリプトウィンドウ](#)[コールスタックリスト](#)[検査詳細ウィンドウ](#)[デバッグの起動](#)[実習：電卓](#)[既知の問題](#)

Acrobat JavaScript デバッグを使用可能にする

デバッグを使用するためには、JavaScript とデバッグの両方を使用可能にする必要があります。Acrobat JavaScript 開発環境の動作は、**編集／環境設定**ダイアログで制御できます。JavaScript および JavaScript エディタを使用可能にする方法については、[第 2 章](#)を参照してください。デバッグを使用可能にするには、**環境設定**ダイアログの左側のリストから「JavaScript」を選択し、「**Acrobat を再起動した後で JavaScript デバッグを使用**」オプションにチェックマークが付いていることを確認します。新たにチェックマークを付けた場合、デバッグを使用するためには、Acrobat を再起動する必要があります。

環境設定ダイアログの「**JavaScript デバッグ**」セクションには、[図 3.1](#)に示すように、デバッグに関するその他のオプションも用意されています。

図 3.1 JavaScript 開発ツールの環境設定

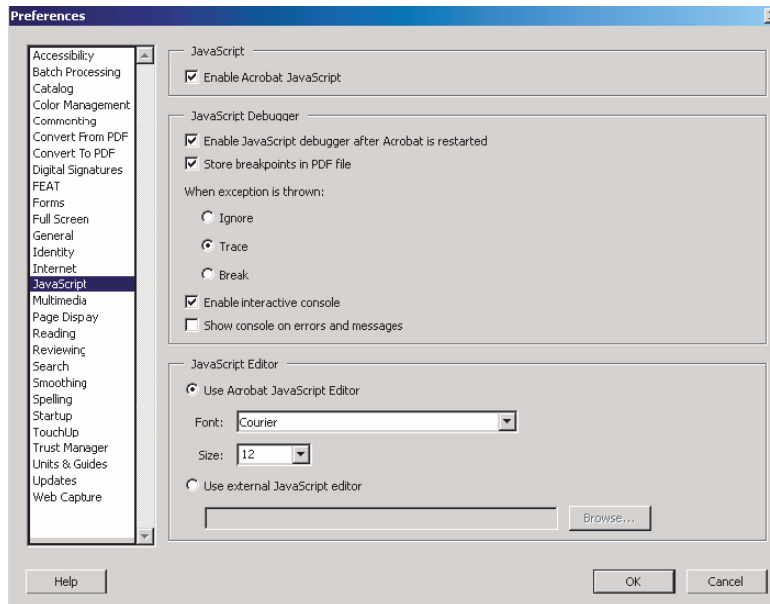


図 3.1 には、JavaScript デバッガオプションの典型的な設定例が示されています。それぞれのオプションの説明を、表 3.1 に示します。

表 3.1 JavaScript デバッガのオプション

オプション	意味
Acrobat を再起動した後で JavaScript デバッガを使用	デバッガを使用可能にするには、このオプションにチェックを付ける必要があります。Acrobat を再起動すると、ブレークポイントの設定などのデバッガ機能が使用できるようになります。
ブレークポイントを PDF ファイルに保存	このオプションを有効にすると、設定したブレークポイントが保存され、次に Acrobat を起動したときやファイルを開いたときに、再度利用できるようになります。ファイルのデバッグが完了して配布できるようになった場合に、ブレークポイントを削除するには、次のようにします。 <ul style="list-style-type: none"> このオプションをオフにします。 アドバンスド / JavaScript / 文書レベル JavaScript の編集を選択して、「ACRO_Breakpoints」スクリプトを削除します。 ファイルを保存します。

オプション	意味
例外発生時の対処	<p>例外発生時の対処方法を、次の 3 つの中から選択します。</p> <ul style="list-style-type: none"> 無視する：例外を無視します。 トレースする：スタックトレースを表示します。 中断する：実行を停止し、例外が発生した行および関数でデバッガを起動するかどうかを尋ねるメッセージウィンドウを表示します。
インタラクティブコンソールを使用	<p>このオプションを有効にすると、コンソールウィンドウでの編集が可能になります。このオプションにチェックが付いていないときに、テキストを編集しようとしてコンソールウィンドウをクリックすると、次のメッセージが表示されます。</p> <p>インタラクティブコンソールは現在使用できません。使用可能にしますか？</p> <p>「はい」をクリックすると、このオプションが有効になります。環境設定を表示すると、このオプションにチェックが付いていることが確認できます。</p>
エラーとメッセージをコンソールに表示	<p>このオプションを有効にすると、デバッガダイアログのコンソールウィンドウが開いて、メッセージが適宜表示されるようになります。また、デバッガが使用可能になっていない場合でも、エラー発生時にはデバッガダイアログが開いて、コンソールウィンドウにエラーメッセージが表示されます。ただし、コンソールウィンドウをクリックすると、次のメッセージが表示されます。</p> <p>インタラクティブコンソールは現在使用できません。使用可能にしますか？</p> <p>「はい」をクリックすると、インタラクティブコンソールが使用可能になります。</p>

デバッガダイアログウィンドウ

アドバンスト / JavaScript / デバッガを選択すると、スクリプトのデバッグを行っていない状態で、デバッガダイアログウィンドウを開くことができます。スクリプトを対話的にデバッグする前に、次の説明を読んで、ウィンドウの各部分やコントロールについて理解してください。

デバッグ可能なスクリプトの種類と、そのスクリプトがある場所については、[41 ページの「スクリプトウィンドウ内のスクリプトへのアクセス」](#)で説明しています。スクリプトで自動的にデバッガを起動して、対話的なデバッグセッションを開始する方法については、[47 ページの「デバッガの起動」](#)で説明しています。

メインコントロールグループ

デバッグダイアログウィンドウには、[図 3.2](#) に示すように、3 つのメインコントロールグループがあります。左上にあるツールバーには 6 個のボタンがあり、さまざま方法でコードをステップ実行したり、デバッグを終了したりすることができます。

ツールバーのすぐ下には**スクリプト**ウィンドウがあり、デバッグ可能なスクリプトの名前が表示されます。スクリプトは、[図 3.2](#) のように、ツリー構造で整理されています。なお、このウィンドウの**スクリプト**階層と、その下のスクリプトウィンドウは違うものですので、注意してください。**スクリプト**ウィンドウには、1 つのスクリプトの内容が表示されます。

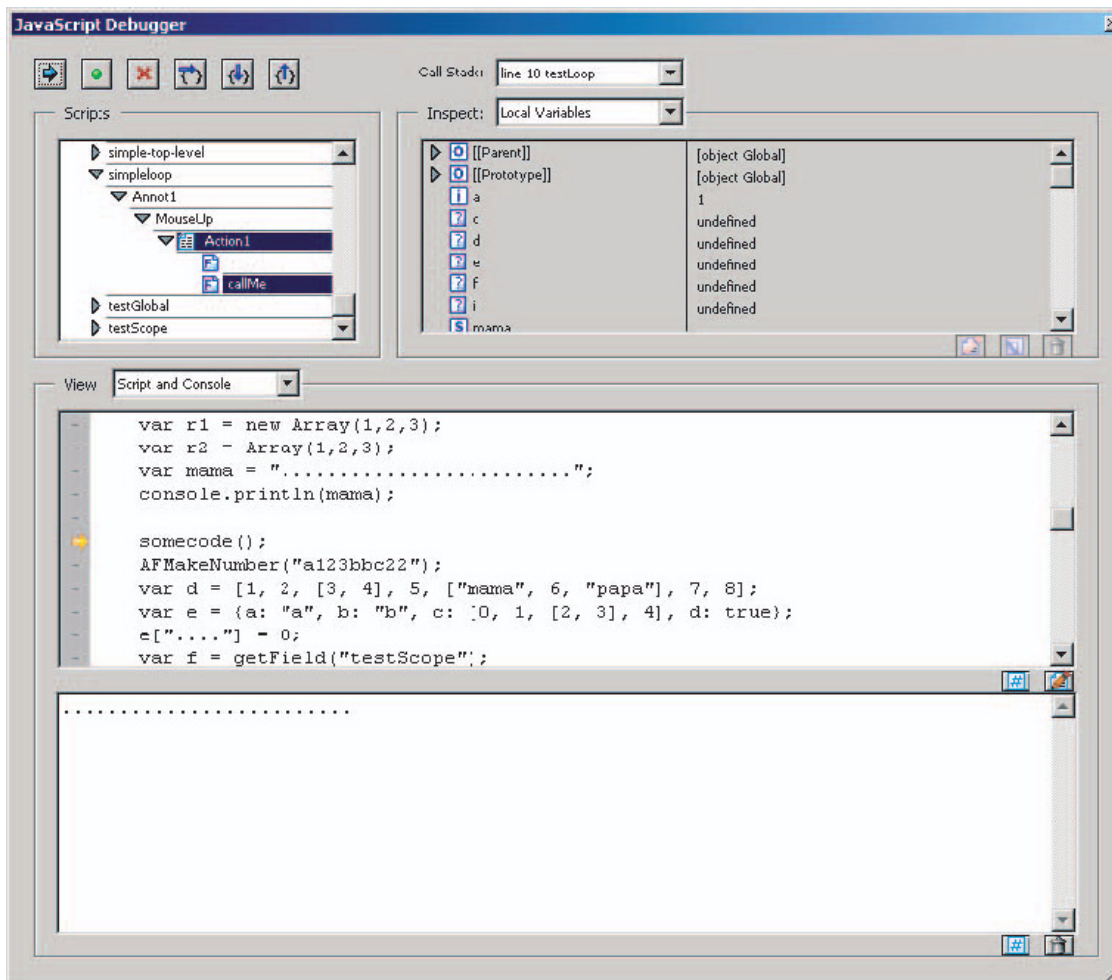
デバッグダイアログの右上には、**コールスタック**および**検査**ドロップダウンリストがあります。これらのリストでエントリを選択すると、ネストしている関数を表示したり、変数、ウォッチ、ブレークポイントなどの詳細を**検査**詳細ウィンドウに表示したりすることができます。

デバッグの表示ウィンドウ

メインコントロールグループの下には、表示ウィンドウがあります。**表示**ドロップダウンリストから、次のエントリを選択できます。

- **スクリプト**：上の「スクリプト」階層ウィンドウで選択した JavaScript スクリプトの内容が表示されます。
- **コンソール**：選択したスクリプトを実行したときの出力が、JavaScript コンソールウィンドウに表示されます。このコンソールを使用して、スクリプトを実行したり、個々のコマンドを実行したりすることもできます。その詳細については、[第 2 章 「Acrobat JavaScript エディタおよびデバッグコンソール」](#)を参照してください。
- **スクリプトとコンソール**：コンソールウィンドウとスクリプトウィンドウが同時に表示されます。[図 3.2](#) に示すように、スクリプトウィンドウはコンソールウィンドウの上に表示されます。

図 3.2 デバッガダイアログ



デバッガのボタン

Acrobat JavaScript デバッガでは、実行したスクリプトのどの部分を調べるかを、簡単に制御することができます。デバッガは [47 ページの「デバッガの起動」](#) で説明しているさまざまな方法で起動することができます。

ツールバーのデバッガボタンを [図 3.3](#) に示し、[表 3.2](#) に各ボタンの機能を簡単に示します。各ボタンの詳細については、この表の後の説明を参照してください。[48 ページの「コードのステップ実行」](#) では、これらのボタンを組み合わせる使用する方法を簡単に説明しています。

図 3.3 デバッガのボタン

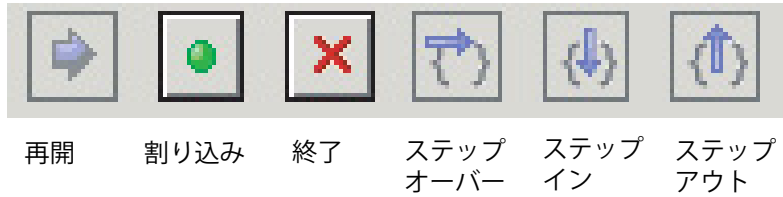


表 3.2 デバッガボタンの概要

ボタン	説明
実行を再開	デバッガで停止しているスクリプトを実行します。
割り込み	実行を停止します。
終了	スクリプトの実行を中断して、デバッガダイアログを閉じます。
ステップオーバー	命令を 1 ステップ実行します。ただし、関数呼び出しがあってもその関数には入りません。
ステップイン	命令を 1 ステップ実行します。関数呼び出しがある場合は、その関数に入ります。
ステップアウト	呼び出されている関数のコードを実行し、その関数の呼び出し元の直後の命令で停止します。

実行を再開

スクリプトが停止しているときに**実行を再開**ボタンをクリックすると、スクリプトが続きから再開され、次のいずれかに到達するまで実行されます。

- 次に実行するスクリプト
- 次のブレークポイント
- エラーの発生
- スクリプトの最後

また、**割り込み**または**終了**ボタンをクリックして実行を停止することもできます。

割り込み

割り込みボタンを使用すると、現在のスクリプトの実行が停止されます。このボタンをアクティブ状態にするには、ボタンをクリックします。アクティブ状態になると、ボタンは赤色になります。このボタンがアクティブ状態になっているときに、JavaScript を実行するアクションが Acrobat で行われると、そのスクリプトはスクリプトの先頭で実行が停止されます。**割り込み**ボタンは、一度使用されると非アクティブ状態に戻ります。別のスクリプトを停止させたい場合は、もう一度このボタンをクリックする必要があります。

終了

終了ボタンをクリックすると、デバッグセッションが中断されて、デバッグダイアログウィンドウが閉じます。

ステップオーバー

ステップオーバーボタンをクリックすると、命令が 1 ステップ実行されます。関数呼び出しの箇所でステップオーバーをクリックした場合は、その関数全体が 1 ステップで実行されます（ボタンをクリックするたびにその関数内に定義された命令が 1 つずつ実行されるわけではありません）。例えば、デバッグの位置インジケータ（黄色の矢印）が、[図 3.4](#) に示すように、関数呼び出しの左側に表示されているとします。

図 3.4 関数呼び出しの行に表示されている位置インジケータ



`callMe();`

この位置は、`callMe()` を呼び出す直前を示しています。ここでステップオーバーをクリックすると、`callMe()` 関数全体が実行され、位置インジケータが関数呼び出しの次のスクリプト命令に進みます（`callMe()` にエラーやブレークポイントがない場合）。

位置インジケータが示しているステートメントに関数呼び出しが含まれていない場合、ステップオーバーをクリックすると単純にそのステートメントが実行されます。

ステップイン

ステップインボタンをクリックすると、関数内の個々のステートメントが実行されます。例えば、デバッグの位置インジケータが[図 3.4](#) の位置にあるとします。ここでステップインをクリックすると、`callMe()` 関数内の最初のステートメントに進みます。

注意： JavaScript により実装されているのではないネイティブ関数にはステップインできませんので、注意してください。この制限は、JavaScript のコア関数だけでなく、Acrobat のネイティブ関数にも適用されます。

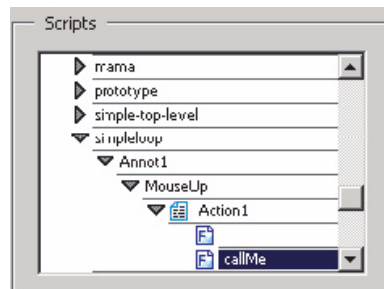
ステップアウト

ステップアウトボタンをクリックすると、呼び出されている関数のコードが実行され、その関数の呼び出し元の直後の命令で停止します。呼び出されている関数にバグがないことが確認された場合には、このボタンを使用すると、その関数の実行をすばやく完了することができます。関数呼び出しの内部にいないときにステップアウトボタンをクリックすると、スクリプトの最後かブレークポイントに到達するまでコードが実行されます（スクリプトにエラーがない場合）。

デバッグのスクリプトウィンドウ

デバッグダイアログを開くと、Acrobat で開いている PDF ファイルから自動的にスクリプトが集められます。集められたスクリプトは、[図 3.5](#) のように、スクリプトウィンドウに表示されます。

図 3.5 スクリプトウィンドウ



スクリプトウィンドウ内のスクリプトへのアクセス

目的のスクリプトにアクセスするには、スクリプトウィンドウのエントリの隣にある三角形をクリックして、1つ下の階層を開きます。続けて、同じように三角形をクリックして、下の階層を開いていきます。階層を開いていくときは、スクロールバーを使用して、適宜内容を表示させてください。表示されるスクリプトアイコンは、アクションや関数にスクリプトが定義されていることを示します。[図 3.5](#) の例では、**Button1** という名前のボタンタイプのフォームに、マウスボタンを放したときのアクションが関数として定義されています。スクリプトアイコンをクリックすると、スクリプトの先頭のステートメントがデバッグで開かれます。

Acrobat JavaScript は、PDF ファイルの内部または外部の、いくつかの場所に保存できます。PDF ファイルの内部にあるスクリプトは、JavaScript タイプのアクションに関連付けられています。以下の節では、PDF ファイルの内部および外部について、スクリプトを保存できる場所を一覧にまとめています。これらのスクリプトは、Acrobat JavaScript デバッグで直接表示して、デバッグすることができます。

PDF ファイルの内部にあるスクリプト

スクリプトウィンドウに表示される PDF ファイル内のスクリプトの種類を、[表 3.3](#) に示します。これらのスクリプトは、デバッグダイアログでいつでも表示したりデバッグしたりすることができます。これらのスクリプトは、デバッグで編集することも可能です。また、ブレークポイントを設定することもできます ([45 ページの「ブレークポイント」](#)を参照)。スクリプトに変更を加えた場合、それが反映されるのは次回の実行時からになります。実行中のスクリプトに変更を加えることはできません。

表 3.3 PDF ファイルの内部にあるスクリプト

場所	アクセス
文書レベル	アドバンスト / JavaScript / 文書レベル JavaScript の編集

場所	アクセス
文書のアクション	アドバンスト / JavaScript / 文書のアクションを設定
ページのアクション	「ページ」タブでページをクリックし、オプション / ページのプロパティを選択
フォーム	フォーム編集モード（下記参照）でフォームオブジェクトをダブルクリックして、フォームのプロパティダイアログを表示
しおり	「しおり」タブでしおりをクリックし、オプション / プロパティを選択
リンク	フォーム編集モードでリンクオブジェクトをダブルクリックして、リンクのプロパティダイアログを表示

フォーム編集モード — フォーム編集モードに切り替えるには、高度な編集ツールバーを表示する必要があります。高度な編集ツールバーが表示されていない場合は、**ツール / 高度な編集 / 高度な編集ツールバーを表示**を選択します。このツールバーは、スクリプトの作成時によく使用するので、ツールバー領域に収めておくとう便利です。

PDF ファイルの外部にあるスクリプト

Acrobat の外部にあるスクリプトも、スクリプトウィンドウに表示され、Acrobat でデバッグすることができます。それらのスクリプトの種類とそのアクセス方法を、[表 3.4](#) に示します。

表 3.4 PDF ファイルの外部にあるスクリプト

場所	アクセス
フォルダレベル	Acrobat またはユーザの JavaScripts フォルダ（グローバル変数の保存ファイル glob.js を JavaScript で生成すると、ユーザの JavaScripts フォルダが作成されます）に、JavaScript（.js）ファイルとして保存
コンソール	コンソールウィンドウで入力し評価したスクリプト
バッチ	アドバンスト / バッチ処理を選択

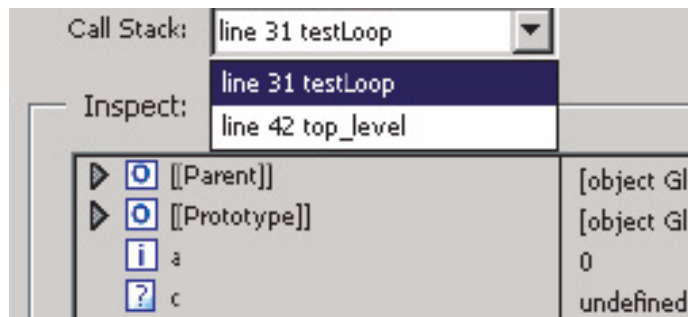
フォルダレベルのスクリプトは、通常、Acrobat での表示およびデバッグはできますが、編集はできません。コンソールスクリプトとバッチ処理スクリプトは、実行するまでデバッグには表示されません。したがって、これらのスクリプトは、その実行前にブレークポイントを設定することはできません。これらのスクリプトにアクセスするには、「先頭からデバッグ」オプションを使用するか、**debugger** キーワードを使用します。詳しくは、[47 ページの「デバッグの起動」](#)を参照してください。

コールスタックリスト

デバッグの制御ボタンの右には、**コールスタックリスト**があります。**コールスタック**ドロップダウンリストには、スクリプト内での現在の実行スコープが表示されます。[図 3.6](#) を参照してくだ

さい。コールスタックにテキストが表示されるのは、デバッガで実行が停止しているときです。それぞれのテキストエントリは、スタックに積まれている関数呼び出し（フレーム）を表しています。各エントリのテキストには、関数の名前と、スクリプト内でのその関数の行番号が表示されます。最も新しいスタックフレームは、スタックの一番上に積み、**コールスタック**ドロップダウンリストでも一番上に表示されます。スタック内の特定のフレームでのローカル変数を検査するには、そのフレームをクリックします。コールスタックリストのすぐ下にある**検査詳細**ウィンドウに、変数が表示されます。

図 3.6 コールスタック



コールスタックの別のエントリを選択すると、そのスタックフレームがアクティブになり、そのフレームに対応するスクリプト内の場所が、**表示**ウィンドウに表示されます。**表示**ウィンドウでスクリプトの左側に表示される緑色の三角形は、フレームの位置を示しています。**検査**ドロップダウンリストで「**ローカル変数**」が選択されている場合は、アクティブにしたフレームに固有な変数が**検査**詳細ウィンドウに表示されます。

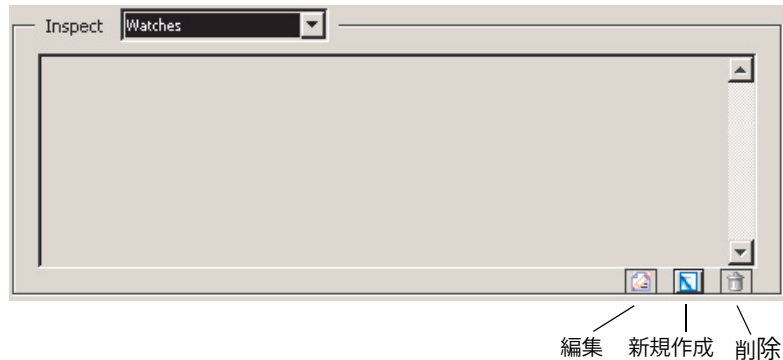
検査詳細ウィンドウ

スクリプトウィンドウの右の、**コールスタック**リストの下には、**検査**詳細ウィンドウがあります。このウィンドウを使用すると、変数の値を検査したり、あらかじめ選択した変数の値を特別に検査したりする（ウォッチを設定する）ことができます。また、ブレークポイントに関する詳細情報を得ることもできます。

詳細ウィンドウのコントロール

検査詳細ウィンドウの下には、アイテムの編集、作成、削除を行うための3つのボタンがあります。これらのボタンは、作成、編集、削除が可能なアイテムを**検査**詳細ウィンドウで選択するまで、淡色表示されます。図 3.7 に、**詳細**ウィンドウの**編集**、**新規作成**および**削除**ボタンを示します。

図 3.7 検査詳細ウィンドウのボタン



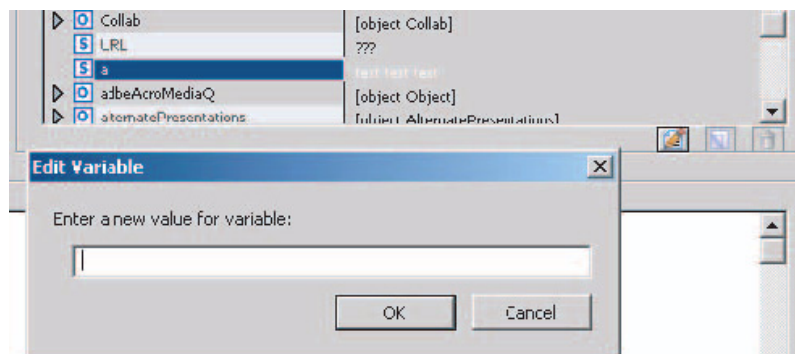
変数の検査

変数の検査機能は、JavaScript のオブジェクトおよび変数の状態を確認できる強力なツールです。このツールを使用すると、デバッグ中のスタックフレーム（ローカルスコープ）での変数の値を確認できます。プロパティがオブジェクトである場合は、そのオブジェクトを構成している一連のオブジェクトおよびプロパティのツリー構造を順次検査していくことができます。

変数の検査を行うには、**検査**ドロップダウンリストから「**ローカル変数**」を選択します。スクリプトの途中でデバッグ処理が停止すると、検査ドロップダウンリストの下の**検査**詳細ウィンドウに、変数とその値の一覧が表示されます。変数を編集する場合は、詳細ウィンドウでその変数を選択します。すると、ウィンドウの下の編集ボタンが使用可能になります。**編集**ボタンをクリックすると、[図 3.8](#) のようなポップアップウィンドウが表示されますので、変数の新しい値を入力します。

プロパティ名の隣にある三角形は、そのプロパティがオブジェクトであることを示しています。オブジェクトのプロパティを表示するには、その三角形をクリックして、オブジェクトのプロパティを展開します。リストを折りたたむには、三角形をもう一度クリックします。

図 3.8 ローカル変数の詳細



ウォッチ

ウォッチリストを使用すると、独自の変数検査を行うことができます。ウォッチは JavaScript の式で、ブレークポイントやステップ実行でデバッガが停止するたびに評価されます。ウォッチは、必要なだけウォッチリストに追加することができます。検査詳細ウィンドウの下にある 3 つのボタンを使用して、ウォッチの編集、追加、削除を行うことができます。評価された結果は、ウォッチリストへの登録順に、検査詳細ウィンドウに表示されます。

ウォッチに関して作業を行うには、検査ドロップダウンリストから「ウォッチ」を選択します。詳細ウィンドウの下の新規作成ボタンをクリックすると、ポップアップウィンドウが表示され、評価したい JavaScript の変数や式が入力できるようになります。

ウォッチの値を変更するには、目的のウォッチをリストから選択します。編集ボタンをクリックするとポップアップが表示されるので、新しい式を指定します。ウォッチを削除するには、検査詳細ウィンドウから目的のウォッチを選択して、削除ボタンをクリックします。これで、そのウォッチが詳細ウィンドウから削除されます。

ブレークポイント

検査ドロップダウンリストの「ブレークポイント」オプションを選択すると、プログラム内のブレークポイントを操作することができます。ブレークポイントとは、ローカル変数の値を確認するためにスクリプトの実行を停止させる位置のことです。ブレークポイントは、特定のコード行に対して定義します。ブレークポイントには、条件を指定することもできます ([46 ページの「条件付きブレークポイントの使用」](#)を参照してください)。

実行中のスクリプトがブレークポイントに到達すると、実行が停止され、そのコード行がデバッガに表示されます。

ブレークポイントを追加するには、スクリプトウィンドウで、スクリプトの左側にある灰色の帯の部分をクリックします。ブレークポイントを設定できる行には、短い横線が表示されます。ブレークポイントが設定されている場所には、赤い点が表示されます。赤い点をクリックすると、ブレークポイントが削除されます。

コーディングスタイルとブレークポイント

関数を定義する際に、左側の中かっこ ({) をどこに記述するかは、スタイルによります。

スタイル 1: 左側の中かっこを、関数名と同じ行に置く場合の例

```
function callMe() { // 中かっこを関数名と同じ行に置く
    var a = 0;
}
```

スタイル 2: 左側の中かっこを別の行に置く場合の例

```
function callMe()
{ // 中かっこを別の行に置く
    var a = 0;
}
```

スタイル 2 の方法を採用している場合、Acrobat JavaScript デバッガでは、関数名の行にブレークポイントを設定することはできません。この行にブレークポイントを設定しようとすると、その場所にブレークポイントが設定できないことを知らせる警告が表示されます。関数名の行には、左側の中かっこが含まれている場合にのみ、ブレークポイントを設定できます。

図 3.9 の例では、関数名 `callMe()` の次の行（左側の中かっこが置かれている行）にブレークポイントを設定できます。また、関数名 `testLoop()` と左側の中かっこが含まれている行にもブレークポイントを設定できます。どちらの場合も、効果に違いはありません。ブレークポイントから実行を開始すると、その関数が評価されます。

図 3.9 関数定義の前にブレークポイントを設定する

```
function callMe()
- {
-   var a = 0;
-   b = 0;
- }

function testLoop(max) {
-   var a = 1;
-   b = 1;
-   var c;
```

関数を呼び出す直前の位置にブレークポイントを設定してスクリプトを実行すると、呼び出された関数内の最初のステートメントで実行が停止します。

ブレークポイントの一覧表示

検査 ドロップダウンリストで「ブレークポイント」オプションを選択すると、デバッグセッションで設定したすべてのブレークポイントが一覧表示されます。**検査** 詳細ウィンドウの下にあるボタン（図 3.7 を参照）を使用すると、ブレークポイントの編集や削除が行えます。編集または削除するブレークポイントを**検査** 詳細ウィンドウで選択すると、ブレークポイントを操作するボタンが使用可能になります。

条件付きブレークポイントの使用

デバッグを行うときには、条件付きブレークポイントが必要になる場合があります。例えば、ループの制御変数が特定の値に達した場合にのみ問題が発生する繰り返しループがあるとします。このループをデバッグする場合、その特定の値に達するまでループをステップ実行していくのは、非常に手間です。実行を停止するタイミングが重要になる場合には、条件付きブレークポイントを使用すると便利です。

条件付きブレークポイントを使用すれば、指定した条件が満たされた場合にのみプログラムを停止して、デバッグを起動させることができます。この条件は、JavaScript の式で指定します。この式の評価結果が `true` になった場合は、そのブレークポイントでプログラムが停止します。それ以外の場合は停止しません。条件が付いていないブレークポイントの場合は、そのブレークポイントが設定されているコード行に達すると必ずプログラムが停止し、デバッグが起動されます。

Acrobat でブレークポイントを作成すると、条件はデフォルトで `true` に設定されます。ブレークポイントの条件を変更するには、**検査** 詳細ウィンドウから目的のブレークポイントを選択して、編集ボタンをクリックします。すると、ポップアップウィンドウが表示され、ブレークポイントの条件が変更できるようになります。

デバッグの起動

デバッグは、4通りの方法で起動できます。このうちの2つは、実行の開始時点からデバッグを行うための方法です。残りの2つは、スクリプトの任意の場所からデバッグを開始するための方法です。

実行の開始時点からのデバッグ

スクリプトの最初の行からデバッグを起動するための方法を、2つ示します。デバッグが開始したら、[ステップイン](#)ボタンを使用してデバッグを続けます。

先頭からデバッグ

アドバンスド／JavaScript を選択し、「先頭からデバッグ」オプションにチェックマークがついていない場合はこのオプションを選択します。チェックマークが付いているオプションをもう一度クリックすると、オプションがオフになります。

このオプションにチェックマークが付いている場合、Acrobat でスクリプトを実行しようとする時、最初の行で実行が停止し、デバッグが開きます。これで、スクリプトの先頭のステートメントからデバッグを開始できるようになります。

注意：「先頭からデバッグ」のチェックマークは、自動的に解除されることはありません。このオプションを使用し終えたら、忘れずにチェックを解除してください。チェックを外し忘れると、Acrobat でスクリプトが実行されるたびにスクリプトが停止します。

割り込みボタンをクリックする

デバッグウィンドウを開いて、[割り込み](#)ボタンをクリックします（ボタンが赤色になります）。割り込みボタンがアクティブ状態になっているときに、JavaScript を実行するアクションが Acrobat で行われると、そのスクリプトは先頭で実行が停止されます。

割り込みボタンは、「先頭からデバッグ」オプションとは異なり、一度スクリプトを停止させると非アクティブ状態に戻ります。別のスクリプトで実行を停止させたい場合は、もう一度ボタンをクリックしてアクティブ状態にする必要があります。

スクリプト内の任意の箇所からのデバッグ

ブレークポイントを定義する

スクリプトの特定の箇所からデバッグを開始するには、ブレークポイントを設定します。ブレークポイントの設定方法について詳しくは、[45 ページの「ブレークポイント」](#)を参照してください。

debugger キーワードを使用する

debugger キーワードを任意のコード行に挿入して、その行に達したときに実行を停止してデバッグを起動させることができます。

注意： debugger キーワードによるブレークポイントは、[検査](#)ドロップダウンリストで「ブレークポイント」を選択しても、[検査](#)詳細ウィンドウには表示されません。

コードのステップ実行

Acrobat JavaScript デバッガでは、スクリプト内のどの部分で実行時の様子を調べるかを、簡単に制御することができます。デバッガでスクリプトが停止しているときには、ツールバーのうちの4つのボタンを使用して、コードをステップ実行できます。この4つのボタンは、**実行を再開**、**ステップオーバー**、**ステップイン**、および**ステップアウト**です。それぞれのボタンについて詳しくは、[38 ページの「デバッガのボタン」](#)を参照してください。ここでは、これらのボタンを組み合わせることでデバッガを制御する方法について説明します。

現在停止している行に JavaScript 関数への呼び出しが含まれていて、その関数内のステートメントを1つずつ実行したいときは、ツールバーの**ステップイン**ボタンをクリックします。これを、関数へのステップインといいます。これによって、関数が呼び出されたときのスクリプトの動作を調べることができます。JavaScript により実装されているのではないネイティブ関数にはステップインできませんので、注意してください。

現在停止している行に関数呼び出しが含まれていて、その関数内のステートメントを1つずつ実行するのを避けたいときは、ツールバーの**ステップオーバー**ボタンをクリックします。これを、関数のステップオーバーといいます。現在停止しているステートメントに関数呼び出しが含まれていない場合、ステップオーバーを行うと単純にそのステートメントが実行されます。

呼び出された関数の内部で停止している場合、その関数の残りのステートメントのステップ実行を飛ばしたいときは、ツールバーの**ステップアウト**ボタンをクリックします。これを、関数からのステップアウトといいます。これにより、その関数を呼び出したステートメントに戻ることができます。関数の内部にいないときにステップアウトボタンをクリックすると、スクリプトの最後かブレークポイントに到達するまでコードが実行されます。

デバッガで停止している場合に、現在のステートメントから続けて実行を開始したいときは、ツールバーの**実行を再開**ボタンをクリックします。プログラムは最後まで実行されるか、次のブレークポイントに到達した場合はそこで再び停止します。

実習：電卓

以下の実習を行うためには、**TestDebugger.zip** ファイルから **Calc.pdf** ファイルを解凍 (unzip) しておく必要があります。

この実習では、スクリプトにブレークポイントを設定し、スクリプトの実行に従って変数が変化していく様子をデバッガで確認できるようにウォッチを作成します。この実習を終えると、次のことができるようになります。

- スクリプト内の任意の場所からデバッガを起動する。
- **検査詳細**ウィンドウの表示内容を理解する。
- ウォッチを作成、編集、および削除する。
- ブレークポイントの設定／解除を行う。
- 次のデバッガボタンを使用して、スクリプトでのコードの実行を制御する。
 - **ステップイン**
 - **ステップアウト**
 - **終了**

電卓

Calc.pdf では、文書レベルのスクリプトとフォームフィールドレベルのスクリプトを使用して、簡単な電卓を実装しています。キーパッドには、数値キー、機能キー、小数点 (.)、等号 (=)、キャンセルキー、入力のキャンセルキーが用意されています。

出力の表示方法

Calc.pdf では、Acrobat の **Doc オブジェクト** のメソッドである `getField()` を使用して、ユーザが入力したキーの値や式の評価結果を、電卓の表示ウィンドウに表示しています。電卓の表示ウィンドウは、**Display** テキストフィールドです。このフィールドは、`getField()` を使用した次のステートメントによって、変数 `display` にバインドされています。

```
var display = this.getField("Display");
```

数値キーは、「マウスボタンを放す」アクションとして次のスクリプトが定義されたボタンフィールドで、戻り値を電卓の表示ウィンドウに割り当てています。

```
display.value = digit_button(n);
```

例えば、ユーザが「7」のボタンを押すと、`digit_button(7)` の戻り値が電卓ウィンドウに割り当てられます。

また、四則演算を表す文字列も、同じ要領で **Func** テキストフィールドに表示されます。このフィールドは、`getField()` を使用した次のステートメントによって、変数 `func` にバインドされています。

```
var func = this.getField("Func");
```

除算 (/) キーの「マウスボタンを放す」アクションには、次のスクリプトが定義されています。

```
func_button("DIV");
```

関数 `func_button()` に文字列 "DIV" を渡しています。この関数のパラメータは `req_func` という名前なので、`func_button()` が実行されると、次のようになります。

```
req_func = "DIV"
```

この関数の中には、次のステートメントがあります。

```
func.value = req_func
```

`func` 変数は `getField()` によって「Func」フィールドにバインドされているので、このステートメントが実行されると、文字列 "DIV" がフィールドに表示されます。

2桁の値を扱うための調整

10以上の値が入力されたときには、現在の値が10倍されて調整が行われます。小数点以下の値が入力されたときには、除数が10倍されて調整が行われます。例えば、ユーザが2と4を連続して入力したとします。この場合は、2が10倍されて、その結果に4が加算されます。小数点ボタンの「マウスボタンを放す」アクションは、「現在の除数が1であれば10に変更する」というものです。したがって、小数点ボタンを押すと、その後に入力した値（小数点の右側に来る数字）は10で除算され、小数点第1位の値に調整されます。

電卓のテスト

電卓の基本的な操作に慣れるために、Acrobat で Calc.pdf を開き、式を入力して結果を確認してみてください。

電卓の操作に慣れたら、ファイルを閉じてください。

はじめに

この実習では、ランタイムエラーが発生する電卓のサンプルを扱います。デバッグを使用してエラーを修正する方法を順を追って説明していき、先ほどテストした電卓と同じように動作するようにします。

注意： この実習を始める前に、Acrobat JavaScript デバッグが使用可能になっていることを確認してください。詳しくは、[34 ページの「Acrobat JavaScript デバッグを使用可能にする」](#)を参照してください。

1. Acrobat で `CalcWithRTErrors.pdf` ファイルを開きます。
2. `Calc.pdf` のときと同じように、式を入力してみます。次のことを確認してください。
 - 入力したボタンの値と、表示された値が一致しているか。
 - 入力した機能キーと、表示された文字列が対応しているか。
3. キャンセルをクリックします。表示がクリアされ、0 が表示されます。
4. 次のテストを行って、自分の結果と「結果：」に記載されている結果を比較します。

テスト：

次の式を入力して、評価します。

$9 - 6 = ???$

キャンセルをクリックして、次の式を評価します。

$9 * 3 = ???$

その結果に 3 を足して、評価します。

$((9 * 3) + 3) =$

もう一度 3 を足して、評価します。

$(((9 * 3) + 3) + 3) =$

キャンセルをクリックします。9 に 3 を足して、評価します。

$9 + 3 = ???$

結果：

$9 - 6 = -6$

$9 * 3 = 0$

$((9 * 3) + 3) = 0$

$(((9 * 3) + 3) + 3) = 0 + 3 = 3$

$9 + 3 = 3$

5. 判明したことをまとめてみます。

以上のテストから、次のことがわかります。

- ボタンの値は、正しく表示されます。9 をクリックすると 9 が、3 をクリックすると 3 が（その他のボタンも同様）、**Display** フィールドに表示されます。

- 機能を表す文字列は、**Func** フィールドに正しく表示されます。*、-、+などをクリックすると、その機能を表す文字列が正しく表示されます。
- 等号 (=) をクリックしても、ほとんどの場合正しい結果が得られません。
- 仮説：コードの中に、少なくとも1つのランタイムエラーがあります。**entry** 変数の値から **accum** 変数の値を得る過程で、何らかの不正な処理が行われているようです。

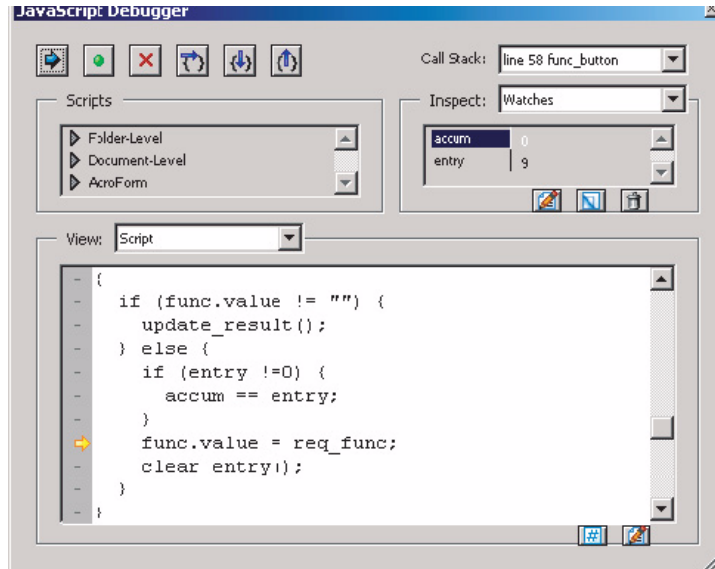
ランタイムエラーのデバッグ

1. 必要に応じて「C」をクリックし、キャンセルを行って表示をゼロにクリアします。
2. **アドバンスド / JavaScript / デバッグ**を選択して、デバッグダイアログを表示します。
3. スクリプトウィンドウに文書レベルスクリプトの **Mult** を表示し、**func_button(MULT)** ; という呼び出しにブレークポイントを設定します。
4. **検査**詳細ウィンドウで、**entry** と **accum** のウォッチを作成します。
どちらの変数も、この時点での値は 0.??? です。
5. **終了**をクリックして、デバッグを閉じます。
6. 電卓のパッドで **9 * 3** という式を入力します。
デバッグによって、ブレークポイントの箇所で実行が停止されます。
7. **ステップイン**を (1 ステップ) クリックして、**func_button()** の最初のステートメントを実行します。
8. さらに 2 ステップ進めます。
このステートメントにエラーがあります。

```
accum == entry; // これは正しくない。
```

このステートメントでは、**entry** の値を **accum** に代入する必要があります。このステートメントを実行しても、**accum** は更新されません。図 3.10 のウォッチを参照してください。
9. ここで**編集**をクリックしてエラーを修正するか、または**ステップアウト**をクリックしてセッションを終了することができます。エラーを修正するために**編集**をクリックすると、次のメッセージが表示されます。
ファイルを編集すると、現在のデバッグセッションが中断されます。ファイルを編集しますか？
「はい」をクリックするとエディタが表示され、エラーを修正できるようになります。== を = に変更します。
注意： ブレークポイントを解除するのを忘れないでください。デバッグを終了し、電卓をもう一度操作してブレークポイントまで進み、そのブレークポイントを解除してください。
または、**ステップアウト**をクリックしてデバッグセッションを終了させ、エディタでエラーを修正して**終了**をクリックしても構いません。
10. 電卓の表示をクリアします。

図 3.10 accum の値が更新されていない



もう一つのランタイムエラー

1. $9 + 3 =$
と入力します。結果は 3 になります。まだ別のランタイムエラーがあるようです。
2. 前回のセッションで使用したウォッチを削除していない場合は、そのまま使用できます。2 つのウォッチについて、次の手順を行います。まず、**検査詳細**ウィンドウで 1 つのウォッチを選択します。次に**編集**をクリックし、「OK」をクリックします。
これで変数が未定義になります。
3. **スクリプト**ウィンドウで、Equals スクリプトを表示します。
4. 関数呼び出し `update_result()`; にブレークポイントを設定します。
5. デバッガを終了します。
6. 必要に応じて表示をクリアします。
7. $9 + 3 =$
と入力します。デバッガが起動し、ブレークポイントの箇所で実行が停止します。
8. 関数 `update_result()` の内部をステップ実行します。

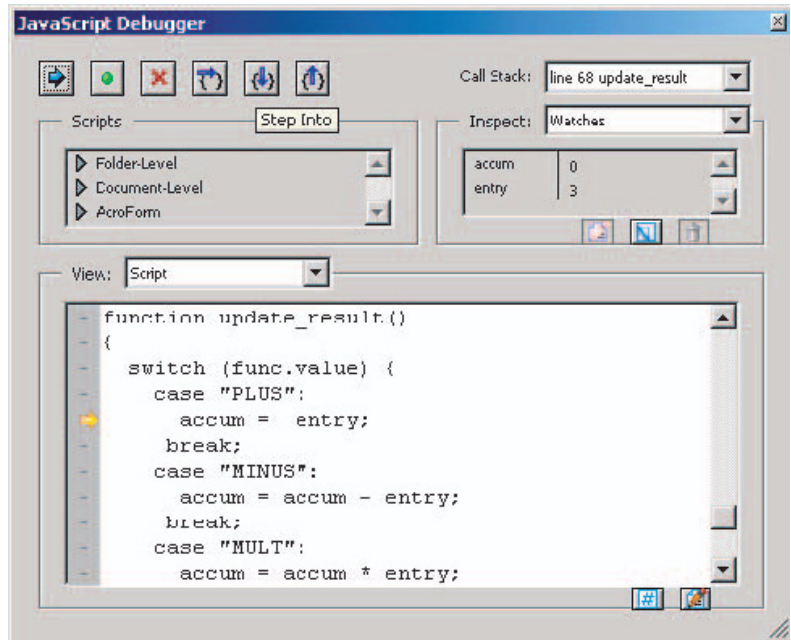
実行しているステートメントと、ウォッチの値に注意してください。最初の case ステートメントを実行します。

```
accum = entry; // これは正しくない。
```

これは正しくありません。accum に entry の値を加える必要があります。次のようにエラーを修正します。

```
accum = accum + entry;
```

図 3.11 2つ目のランタイムエラー



既知の問題

JavaScript デバッガの制約によって、一部のクラスのスクリプトは、Acrobat でデバッグが行えません。しかし、スクリプトのデバッグが可能な環境は他にも数多くあります。Acrobat でデバッグが行えない場合は、デバッグ可能な環境でスクリプトをテストした後で、正式な場所に配置してください。以下で説明するのは、Acrobat 6.0 の JavaScript デバッガを開発しテストする過程で発見された問題点の一部です。

メニュー項目スクリプトは、「先頭からデバッグ」オプションを使用してもデバッグできません。「先頭からデバッグ」メニュー項目オプションにチェックマークが付いている場合、JavaScript メソッドの `app.addItem()` で作成されたメニュー項目は、使用不能になります。このスクリプトをデバッグするには、それぞれの `addItem()` 呼び出しにおいて「cExec」パラメータで指定される関数に、ブレークポイントを設定します。この関数が `Config.js` ファイルや文書レベルスクリプトの中にある場合でも、デバッガダイアログで表示してブレークポイントを設定できます。

「先頭からデバッグ」メニュー項目オプションをオンにして、PDF ファイルを開いたときにスクリプト（文書レベルのスクリプトなど）をデバッグしようとする、デバッガで停止している最中に他のスクリプト（先頭のページを開いた時に実行されるスクリプトなど）が実行されることがあります。その後、それらのスクリプトを実行しても、正しく実行されないことがあります（特に、デバッグのために停止した文書レベルスクリプト内の定義に依存している場合）。

`DocWillPrint` イベントと `DocDidPrint` イベントはデバッグできません。これは、印刷の進捗状況バーがデバッガと競合するためです。この状況に陥った場合は、Escape キーを押してください。スクリプトのデバッグには別のイベントトリガを使用し、その後で目的のイベントにトリガを変更してください。

ブラウザ (IE、NS) で PDF ファイルを表示している場合、そのスクリプトを Acrobat でデバッグするときには、一定の制約があります。この問題は、多くの場合、PDF ファイルの一部分しかダウンロードされておらず、そのファイル内のいくつかのスクリプトが使用できないために起こります。

モーダルダイアログが開くとデバッグが行えなくなります。これは、バッチシーケンスをデバッグするときなどに問題になります。スクリプトでダイアログを作成し定義している場合、それらのダイアログはモーダルの特性を持つので、こうしたスクリプトはデバッグを行えません。このようなスクリプトをデバッグすると、Acrobat アプリケーションがハングアップすることがあります。ハングアップした場合は、Esc キーを押すと正常な状態に戻ります。

`app.setInterval` または `app.setTimeout` メソッドによるイベントが実行されているときにスクリプトをデバッグすると、警告ダイアログボックスが何度も表示される場合があります。この問題を解決するには、モーダルダイアログが消えた後で Esc キーを押してください。

Esc キーを押すのは、スクリプトのデバッグ中に Acrobat アプリケーションがハングアップしたときの最後の手段です。

まとめ

Acrobat JavaScript デバッグは、Acrobat JavaScript スクリプトのトラブルシューティングを行うための機能を完備したツールです。Acrobat JavaScript 開発環境の動作は、**編集/環境設定**ダイアログを使用して詳細に指定できます。このデバッグには、スクリプトの選択や実行に使用する、3つのメインコントロールグループがあります。6つのボタンを使用して、デバッグの開始や終了、コードのステップ実行、関数のステップオーバー、任意の場所での関数からのステップアウトなどを行えます。**スクリプト**ウィンドウでは、デバッグするスクリプトを選択できます。**コールスタック**リスト、**検査**リスト、および**検査**詳細ウィンドウを使用することで、ネストしている関数を表示したり、変数、ウォッチ、ブレークポイントなどの詳細を表示したりすることができます。詳細ウィンドウのコントロールを使用すると、ブレークポイントの設定/解除を行ったり、JavaScript コードをステップ実行しながら変数値を検査したり、スクリプトを実行しながら必要な内容だけを検査できるウォッチの作成/削除を行ったりすることができます。また、スタックをトレースしたり、スタックフレームに対応するスクリプト内の場所を表示したりすることもできます。

4

フォームでの Acrobat JavaScript の使用

簡単な JavaScript の作成

フォームに簡単な JavaScript を組み込むことで、そのインタラクティブな機能を強化することができます。ここでは、Acrobat のフォームでよく使用されるスクリプトについて説明します。ご自身が作成したフォームでそれらのスクリプトを試してみれば、JavaScript で実現できることが大まかに理解できるようになります。

フォームを作成する際にフィールド名の選択を慎重に行うことは、データの収集を行う上で重要な鍵となります。2つのフィールドに同じ名前を付けた場合、それらのフィールドは値も同じになります。この特性を利用すると、外観（表示されるページや背景色など）は異なるが同じ値を持つフィールドを、複数作成することができます。この場合、1つのフィールドを変更すれば、他のフィールドも自動的に更新されます。

注意： Acrobat で JavaScript を作成および使用するためには、JavaScript を使用可能にする必要があります。JavaScript が使用可能になっているかどうかを確認するには、**編集/環境設定**を選択して、左側のリストから「JavaScript」を選びます。「**Acrobat JavaScript を使用**」が選択されていない場合は選択して、「**OK**」をクリックします。

自動日付フィールドの作成

フォームでは、追跡調査を行えるように、日付を使用することがよくあります。ここでは、文書を開いたときに現在の日付が自動的に表示されるテキストフィールドの作成手順を説明します。

文書が開いたときに現在の日付を表示するスクリプトは、文書レベルのスクリプトとして作成します。

自動日付フィールドを作成するには：

1. テキストフィールドツールを選択して、テキストフィールドを作成します（詳細は、Acrobat オンラインヘルプの「フォームフィールドの作成」の項を参照してください）。「**一般**」タブを選択して、フィールドに「Today」という名前を付けます。
2. 「**フォーマット**」タブを選択し、フォーマット分類で「**日付**」を選択し、年月日の書式オプション（「**mmm d, yyyy**」など）を選択します。このフィールドは計算フィールドですので、「**一般**」タブでフィールドを読み取り専用にし、「**閉じる**」をクリックします。
3. 文書が開かれるたびに実行される文書レベルスクリプトを作成します。**アドバンスド/JavaScript/文書レベル JavaScript の編集**を選択します。スクリプト名に「**Today**」と入力して、「**追加**」をクリックします。
4. スクリプトウィンドウに、**Today** 関数を定義するテキストが自動的に生成されますが、これをすべて削除します。次のテキストを、この体裁のとおりに入力して（行が折り返されるかもしれませんが、それは問題ありません）、「**OK**」をクリックします。

```
var f = this.getField("Today");  
f.value = util.printd("mm/dd/yyyy", new Date());
```

このスクリプトでは、**Today** フィールドを変数 **f** にバインドした後、その値を計算しています。**new Date()** は、初期値として現在の日時を持つ日付オブジェクトを新規作成します。**util** オブジェクトは、日付を月/日/年の書式にするために使用しています。

5. JavaScript エディタダイアログボックスで「OK」をクリックし、JavaScript 関数ダイアログボックスで「閉じる」をクリックします。

フィールドの日付は、**util.printd** で生成した書式ではなく、テキストフィールドのプロパティで選択した書式で表示されることに注意してください。

算術計算の実行

JavaScript を使用すると、簡略化したフィールド表記を使用して複数のフィールドの値に基づいて算術計算を実行し、別のフィールドに計算結果を表示することができます。次の例では、3つのフォームフィールドを作成します。最初のフィールドの値から2番目のフィールドの値を引いた結果が、3番目のフィールドに自動的に表示されます。

算術計算フィールドを作成するには：

1. テキストフィールドツールを選択して、テキストフィールドを作成します（詳細は、Acrobat オンラインヘルプの「フォームフィールドの作成」の項を参照してください）。フィールドに「**ValueA**」という名前を付けます（スペースは入れないでください）。
2. 「**フォーマット**」タブをクリックし、「**数値**」を選択して、小数点以下の桁数、通貨記号（必要な場合）、および桁区切りのスタイルを選択します。
3. 「**オプション**」タブをクリックして、デフォルト値（例えば 1）を指定し、「**閉じる**」をクリックします。
4. 2番目のテキストフィールドを作成し、「**ValueB**」という名前を付けます（スペースは入れないでください）。
5. 「**フォーマット**」タブをクリックし、前に作成したフィールドと同じフォーマットを指定します。
6. 「**オプション**」タブをクリックして、デフォルト値（例えば 1）を指定し、「**閉じる**」をクリックします。
7. 3番目のテキストフィールドを作成し、「**ResultsC**」という名前を付けます（スペースは入れないでください）。
8. 「**フォーマット**」タブをクリックし、前に作成したフィールドと同じフォーマットを指定します。
9. 「**計算**」タブをクリックし、「**簡略化したフィールド表記**」ボタンをクリックして「**編集**」をクリックします。
10. JavaScript エディタウィンドウに、次のとおりに入力します。

```
ValueA -ValueB
```
11. 「**OK**」をクリックします。
12. テキストフィールドのプロパティダイアログで「**閉じる**」をクリックします。

Acrobat 5 では、次の例のように、それぞれのフィールドを **this** オブジェクトから明示的に取得し、フィールドオブジェクトの **value** プロパティを使用する必要がありました（これは、上の手順 10 の簡略化された構文に相当します）。

```
var f = this.getField("ValueA");
var g = this.getField("ValueB");
event.value = f.value - g.value;
```

簡略化したフィールド表記を利用すると、比較的複雑な計算でも簡単に定義することができます。例えば、次のような式は、

```
event.value = ( getField("income.interest").value
                + getField("income.rental").value ) * 0.45
              - getField("deductible").value;
```

新しい構文では次のように表すことができます。

```
(income¥.interest + income¥.rental) * 0.45 - deductible
```

ピリオド (.) の前に円記号 (¥) があることに注意してください。すべてのオペレータ（この例のピリオドも含む）、数字、および空白文字は、個々のフィールド名の区切りを表します（これは、解釈があいまいにならないようにするためです）。したがって、それらの文字が含まれているフィールド名を「簡略化したフィールド表記」で記述する場合は、上の例のように、それらの文字を円記号でエスケープする必要があります。「簡略化したフィールド表記」のスクリプトでは、引用符はフィールド名の一部と見なされます。

さらに複雑な計算やロジックを使用する場合は、これまでのように「**カスタムの演算スクリプト**」を使用することもできます。このウィンドウにスクリプトを入力する場合は、完全な構文でオブジェクトとプロパティを記述する必要があります。

「ページに移動」アクションの割り当て

フォームが複数のページにわたっている場合は、次のページに移動するボタンを追加しておくとう便利です。この種のアクションは、ほとんどの場合、「マウスボタンを放す」アクションに関連付けます。

ボタンをクリックすると次のページに移動する JavaScript は、少し変更するだけで、前のページ、最初のページ、最後のページに移動するスクリプトになります。次に示す手順では、これらのスクリプトをすべて紹介します。

ボタンに「ページに移動」アクションを指定するには：

1. ボタンツールを選択して、ボタンを作成します（詳細は、Acrobat オンラインヘルプの「フォームフィールドの作成」の項を参照してください）。このボタンに「GoNext」という名前を付けます。

このフォームに複数の「ページに移動」ボタンを作成する場合は、それぞれのフィールドに「GoNext」、「GoPrev」、「GoFirst」、「GoLast」という名前を付けます。

2. 「表示方法」タブをクリックし、境界線、背景色、テキスト、フィールドの表示を指定します。「オプション」タブをクリックして、必要に応じてオプションを指定します（詳細は、Acrobat オンラインヘルプの「インタラクティブなボタンの作成」の項を参照してください）。
3. 「アクション」タブをクリックし、「トリガを選択」で「マウスボタンを放す」を選択します。
4. 「アクションを選択」で「JavaScript を実行」を選択して、「追加」をクリックします。

- ボタンをクリックすると次のページに移動するには、スクリプトウィンドウに次のとおりに入力して、「OK」をクリックします。

```
this.pageNum++;
```

その他の「ページに移動」ボタンについても、同様に次のスクリプトを入力します。

前のページに移動：

```
this.pageNum--;
```

最初のページに移動：

```
this.pageNum = 0;
```

最後のページに移動：

```
this.pageNum = this.numPages - 1;
```

- 「OK」をクリックし、ボタンのプロパティダイアログボックスで「閉じる」をクリックします。

電子メールで文書やフォームを送信する

指定した電子メールアドレスに PDF 文書を自動的に送信するボタンを、フォーム上に作成できます。また、フォームのデータのみを FDF ファイルとして送信することもできます。

以下の例の *name@address.com* 変数は、フォームの送信先の電子メールアドレスを表します。ほとんどの電子メールには、メッセージの内容を簡単に説明する件名がありますが、この例では *Message Subject Description* 変数が電子メールメッセージの説明に相当します。また、2つの引用符が連続して入力されている箇所が2箇所ありますが、ここには必要に応じて cc: および (ブラインドの) bcc: の電子メールアドレスを入力できます。

文書やフォームを電子メールで送信するアクションを割り当てるには：

- ボタンツールを選択して、ボタンを作成します（詳細は、Acrobat オンラインヘルプの「フォームフィールドの作成」の項を参照してください）。このボタンに「MailPDF」という名前を付けます。

フォームのデータのみ (FDF ファイル) を送信するボタンも作成したい場合は、そのボタンを作成して、「MailFDF」という名前を付けます。FDF ファイルには入力されたデータのみが格納され、フォーム自体は格納されないため、ファイルサイズが小さくなります。

- 「表示方法」タブをクリックし、境界線、背景色、テキスト、フィールドの表示を指定します。「オプション」タブをクリックして、必要に応じてオプションを指定します（詳細は、Acrobat オンラインヘルプの「インタラクティブなボタンの作成」の項を参照してください）。
- 「アクション」タブをクリックし、「トリガを選択」で「マウスボタンを放す」を選択します。
- 「アクションを選択」で「JavaScript を実行」を選択して、「追加」をクリックします。
- ボタンをクリックすると指定した電子メールアドレスに PDF 文書が送信されるようにするには、スクリプトウィンドウに次のとおりに入力して、「OK」をクリックします。

```
this.mailDoc(true, "name@address.com", "", "", "Message Subject Description");
```

フォームのデータのみを FDF ファイルとして送信する場合は、次のスクリプトを使用します。

```
this.mailForm(true, "name@address.com", "", "", "Message Subject Description");
```

6. 「OK」をクリックし、ボタンのプロパティダイアログボックスで「閉じる」をクリックします。

一定の条件が満たされるまでフィールドを非表示にする

より複雑なフォームでは、一定の条件が満たされるまで、フィールドを非表示（非アクティブ）にしておきたい場合があります。例えば、あるフィールドは、別のフィールドに一定以上の金額が入力されるまで、非表示、淡色表示、または読み取り専用にしておく場合があります。

以下の例では、ActiveValue フィールドに 100 ドル以上の金額が入力されない限り、GreaterThan フィールドはアクティブになりません。この例のアクティブなフィールドの名前は ActiveValue で、非アクティブなフィールドの名前は GreaterThan です。

あるフィールドで一定の条件が満たされたときに、別のフィールドをアクティブにするには：

1. テキストフィールドツールを選択して、テキストフィールドを作成します。（詳細は、Acrobat オンラインヘルプの「フォームフィールドの作成」の項を参照してください）。このフィールドに「ActiveValue」という名前を付けます。
2. 「フォーマット」タブをクリックして、分類リストから「数値」を選択します。小数点以下の桁数は 2、通貨記号はドル、桁区切りのスタイルは一般的なもの（デフォルト）を選択します。「閉じる」をクリックします。
3. テキストフィールドをもう 1 つ作成して、「GreaterThan」という名前を付けます。
4. 「フォーマット」タブをクリックして、分類リストから「数値」を選択します。小数点以下の桁数は 2、通貨記号はドル、桁区切りのスタイルは一般的なもの（デフォルト）を選択します。「閉じる」をクリックします。
5. ActiveValue フィールドをダブルクリックします。「検証」タブをクリックし、「カスタム検証スクリプトを実行」を選択して「編集」をクリックします。
6. GreaterThan フィールドの動作別に、それぞれのコードを示します。
 - 100 ドル以上の金額が ActiveValue フィールドに入力されるまで、GreaterThan フィールドを非表示しておくには、スクリプトウィンドウに次のとおりに入力して、「OK」をクリックします。

```
var f = this.getField("GreaterThan");  
f.hidden = (event.value < 100);
```

- 100 ドル以上の金額が ActiveValue フィールドに入力されるまで、GreaterThan フィールドを読み取り専用にするには、スクリプトウィンドウに次のとおりに入力して、「OK」をクリックします。

```
var f = this.getField("GreaterThan");  
f.readonly = (event.value < 100);
```

- 100 ドル以上の金額が ActiveValue フィールドに入力されるまで、GreaterThan フィールドを灰色表示し、読み取り専用には、スクリプトウィンドウに次のとおりに入力して、「OK」をクリックします。

```
var f = this.getField("GreaterThan");
f.readonly = (event.value < 100) ;
f.textColor = (event.value < 100) ? color.gray : color.black;
```

7. JavaScript エディタダイアログボックスで「OK」をクリックし、フィールドのプロパティダイアログボックスで「閉じる」をクリックします。

JavaScript アクションの操作

JavaScript アクションを使用すると、フォームフィールド、リンク、しおり、文書、ページアクションなどから JavaScript を起動することができます。このためには、JavaScript に精通していることが必要になります。よく使用する関数をフィールドレベルのスクリプトとして保存すると、別のスクリプトからその関数が呼び出せるようになります。文書レベルのスクリプトとして関数を保存すると、現在の文書内のすべてのスクリプトからその関数が利用できるようになります。フォルダレベルのスクリプトとして関数を保存すると、アプリケーション内のすべてのスクリプトからその関数が利用できるようになります。フォルダレベルのスクリプトは、拡張子が .js のファイルに格納されます。このスクリプトは、JavaScripts サブフォルダ内に配置しておく必要があります。

JavaScript アクションを選択するには：

1. 目的のフォームフィールド、リンク、しおりまたはページアクションを作成するか、選択します。
2. マウスの右ボタンを押す (Windows) か、Control キーを押しながらクリック (Mac OS) して、「プロパティ」を選択します。
3. アクションとして JavaScript を選択します。フォームフィールド、リンク、しおりまたはページアクションでのアクションの選択については、Acrobat オンラインヘルプの「Adobe PDF 文書へのナビゲーションの追加」／「特殊効果を追加する動作の使用」／「動作の種類について」の項を参照してください。
4. 「追加」をクリックします。
5. 表示されたテキストボックスに、事前に用意したカスタムスクリプトをコピーするか、または直接スクリプトを入力して、「OK」をクリックします。
6. 「閉じる」をクリックします。

文書レベルの JavaScript を作成するには：

1. アドバンスト／JavaScript／文書レベル JavaScript の編集を選択します。
2. スクリプトの名前をテキストボックスに入力します。
3. 「追加」をクリックします。
4. 表示されたテキストボックスに、事前に用意したカスタムスクリプトをコピーするか、または直接スクリプトを入力して、「OK」をクリックします。下のテキストボックスにスクリプト名が表示されます。

5. 「閉じる」をクリックします。
6. **アドバンスト / JavaScript / デバッガ**を選択して、コンソールウィンドウを開きます。JavaScript の実行時にエラーが発生すると、コンソールウィンドウに警告メッセージが表示されます。結果をクリアするには、クリアボタンをクリックします。ウィンドウを閉じるには、閉じるボタンをクリックします。

文書レベルの既存の JavaScript を編集または削除するには：

1. **アドバンスト / JavaScript / 文書レベル JavaScript の編集**を選択します。
2. 文書レベルの JavaScript を編集するには、リストから目的の JavaScript を選択して「編集」をクリックします。表示されたテキストボックスで、テキストを変更するか、事前に用意したカスタムスクリプトをコピーします。「OK」をクリックして、編集を終了します。
3. 文書レベルの JavaScript を削除するには、下のテキストボックスから目的の JavaScript を選択して、「削除」をクリックします。
4. 「閉じる」をクリックします。

フォルダレベルの JavaScript を作成するには：

1. JavaScript 関数を記述したテキストファイルを作成します。拡張子を .js にしてそのファイルを保存します。
2. 作成したテキストファイルを、Acrobat フォルダ内の JavaScripts ディレクトリか、ユーザの JavaScript フォルダにコピーします。

文書レベルの JavaScript アクションの操作

現在の文書内のあらゆる JavaScript からアクセス可能な JavaScript 関数に加え、文書全体に適用される文書レベルの JavaScript アクションを作成することもできます。文書全体に関わる操作（印刷など）には文書レベルのアクションを関連付けることができ、その操作が行われると、関連付けられたアクションが実行されます。JavaScript コードは、任意のエディタで作成および編集できます。文書レベルの JavaScript アクションは、一度にまとめて編集することもできます。

文書レベルの JavaScript アクションを設定するには：

1. **アドバンスト / JavaScript / 文書のアクションを設定**を選択します。
2. アクションの設定ダイアログボックスで、該当する文書操作を選択します。
 - 「文書を閉じる」を選択すると、文書を閉じるときに JavaScript が実行されます。
 - 「文書を保存する」を選択すると、文書を保存するときに JavaScript が実行されます。
 - 「文書を保存した」を選択すると、文書を保存した後で JavaScript が実行されます。
 - 「文書を印刷する」を選択すると、文書を印刷するときに JavaScript が実行されます。
 - 「文書を印刷した」を選択すると、文書を印刷した後で JavaScript が実行されます。
3. 「編集」をクリックします。JavaScript エディタが表示されます（デフォルトの JavaScript エディタは、環境設定ダイアログボックスで設定できます）。

4. エディタにコードを入力して、「OK」をクリックします。外部エディタを使用している場合は、そのエディタでの手順に従ってください。Acrobat に戻る前に、作成したコードを保存し、エディタウィンドウを閉じる必要があります。ダイアログボックスの「この JavaScript を実行」セクションにコードが表示されます。また、JavaScript アクションが設定されていることを示す緑色の丸印が、操作の隣に表示されます。

5. 「OK」をクリックします。

文書レベルの JavaScript アクションを削除するには：

1. アドバンスト／JavaScript／文書のアクションを設定を選択します。
2. アクションの設定ダイアログボックスで、削除したい JavaScript が設定されている文書操作を選択します。
3. 「編集」をクリックします。
4. JavaScript エディタでコードを削除して、「OK」をクリックします。
5. 「OK」をクリックして、アクションの設定ダイアログボックスを閉じます。

文書レベルの JavaScript アクションをすべてまとめて編集するには：

1. アドバンスト／JavaScript／文書のアクションを設定を選択します。
2. 「すべてを編集」をクリックします。そのファイルのすべてのスクリプトが表示されます。文書レベルのスクリプトを表示します。
3. エディタでコードを編集して、「OK」をクリックします。
4. ダイアログボックスの「OK」をクリックします。

プログラムによるフォームフィールドの作成

Acrobat には、フォームフィールドを作成するための JavaScript プロパティおよびメソッドが数多く用意されており、フォームフィールドの外観および関連するアクションを設定することもできます。ここでは、それらのプロパティとメソッドを、Acrobat のユーザインタフェース (UI) ごとに紹介します。Acrobat UI のフォームツールバーとプロパティダイアログを使用してフォームフィールドのプロトタイプを作成した場合も、この節を参照して、対応する JavaScript のプロパティやメソッドを調べることができます。

フォームフィールドには次の 7 種類があり、以下の節で詳しく説明しています。

- ボタン
- リストボックス
- 電子署名
- チェックボックス
- ラジオボタン
- テキスト
- コンボボックス

フォームフィールドの作成には、Acrobat UI のフォームツールバーまたは **Doc Object** の **addField** メソッドを使用できます。フォームフィールドをプログラムで作成するには、次のようにします。

```
var f = this.addField("myField", "fieldtype", 0, [100, 472, 172, 400]);
```

fieldtype には、**button**、**combobox**、**listbox**、**checkbox**、**radiobutton**、**signature**、**text** のいずれかを指定できます。**ラジオボタン**を除くすべてのフィールドは、この方法で作成できます。**ラジオボタン**を作成するには、複数のボタンを関連付ける必要があります。その方法の詳細については、71 ページの「ラジオボタン」を参照してください。

ページ番号は 0 から始まります。座標は、左上の x 座標、左上の y 座標、右下の x 座標、右下の y 座標の順です。したがって、上記の行の場合は、文書の 1 ページ目の [100, 472, 172, 400] の位置にフィールドが作成されます。つまり、幅と高さが 1 インチ (72 ポイント) のフィールドが、ページの左端から 100 ポイント、下端から 400 ポイント離れた位置に作成されます。デザインにもよりますが、チェックボックスはこの例よりも小さく、テキストボックスはこの例よりも大きくするのがふつうです。

フィールドには、デフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトの **f** で、この節全体を通して使用されます。

すべてのフィールドプロパティダイアログの左下隅には、「**ロック**」という名前のチェックボックスがあります。このチェックボックスにチェックを付けると、フォーム編集ツールでカーソルを合わせてもフィールドが選択できなくなります。これによって、UI の操作中に誤ってフィールドを変更してしまうことが防げます。このチェックボックスにチェックを付けている場合でも、ダブルクリックすればフィールドプロパティを編集できます。このチェックボックスは、各ダイアログの右下にある「**閉じる**」ボタンと同じように、UI からのみ操作できます。プログラムからは操作できません。

Acrobat 5 のフォームツール UI の操作は上記の方法に類似しており、フォームフィールドを作成してからそのタイプを**ボタン**などに設定していました。Acrobat 6 の UI では、フォームフィールドツールは 1 つではなく、フォームフィールドのタイプごとにそれぞれのツールが用意されるようになりました。プログラムによる作成方法は、従来の方法から変わっていません。

既存のフィールドの Field オブジェクトは、**Doc Object** の **getField** メソッドを使用して取得することができます。

```
var f = this.getField("myField");
```

どのフィールドオブジェクトでも、**type** プロパティを使用してそのタイプを確認できます。例えば、Field オブジェクト **f** のタイプを表示するには、次のコマンドを使用します。

```
console.println(f.type);
```

type は、読み取り専用のプロパティであることに注意してください。Acrobat 5 の UI とは異なり、フィールドのタイプを変更してその外観を変更することはできません。フィールドのタイプが異なれば、オブジェクトのタイプもまた別のものになります。

どのフィールドタイプでも、プロパティダイアログの「一般」タブに用意されているプロパティは同じです。Acrobat のボタンフィールドの UI には、フィールドの表示方法、オプション、およびアクションを設定するタブがあり、これらのプロパティにはすべてフィールドレベルの JavaScript プロパティおよびメソッドからアクセスすることができます。

ボタン

ボタンのプロパティダイアログボックスには、「一般」、「表示方法」、「オプション」、「アクション」の 4 つのタブがあります。

「一般」タブ

「一般」タブでは、次のプロパティを設定できます。

名前	参照	例
名前	<code>name</code>	<code>console.println(f.name);</code>
ツールヒント	<code>userName</code>	<code>f.userName = "Submit Button"</code>
一般プロパティ		
読み取り専用	<code>readonly</code>	<code>f.readonly = true;</code>
表示と印刷	<code>display</code>	<code>f.display = display.visible</code>
向き	<code>rotation</code>	<code>f.rotation = 90;</code>

表の注意事項

- `name` は、読み取り専用のプロパティです。UI でフィールドを作成するときに設定するか、`addField` を使用してプログラムで作成するときに設定します。下の例を参照してください。
- Acrobat 6.0 では、`Field.rotation` プロパティを使用して、ボタンなどのフィールドを 90 度単位で回転できるようになりました。例えば、下のコードを実行すると、縦向きに回転したボタンが作成されます。

```
var f = this.addField("actionField", "button", 0,
    [200, 250, 250, 400]);
f.strokeColor = color.black;
f.fillColor = color.ltGray;
f.borderStyle = border.b;
f.buttonSetCaption("Push Me");
f.rotation = 90;
```

Acrobat 5 では、次の例のように、ページを回転してからボタンを作成し、ページの回転を元に戻すことで、回転したボタンを作成していました。

```
this.setPageRotations(this.pageNum, this.pageNum, 90);
var f = this.addField("actionField", "button", 0, [200, 250, 300, 200]);
f.delay = true;
f.strokeColor = color.black;
f.fillColor = color.ltGray;
f.borderStyle = border.b;
f.delay=false;
this.setPageRotations(this.pageNum, this.pageNum);
```

「表示方法」タブ

「表示方法」タブでは、フィールドの基本的な外観を設定できます。下の表に、JavaScript を使用して外観をプログラムから設定する方法をまとめます。

領域／名前	参照	例
境界線と色		
境界線の色	<code>strokeColor</code>	<code>f.strokeColor = color.black;</code>
塗りつぶしの色	<code>fillColor</code>	<code>f.fillColor = color.ltGray;</code>
幅	<code>lineWidth</code>	<code>f.lineWidth = 1;</code>
スタイル	<code>borderStyle</code>	<code>f.borderStyle = border.b</code>
テキスト		
フォントサイズ	<code>textSize</code>	<code>f.textSize = 16;</code>
文字の色	<code>textColor</code>	<code>f.textColor = color.blue;</code>
フォント	<code>textFont</code>	<code>f.textFont = font.Times;</code>

「オプション」タブ

「オプション」タブでは、強調表示、レイアウト、ボタン表示の外観を設定できます。

領域／名前	参照	例
レイアウト	<code>buttonPosition</code>	<code>f.buttonPosition = position.iconOnly;</code>
動作	<code>highlight</code>	<code>f.highlight = highlight.p</code>
詳細設定 ...		
サイズ変更	<code>buttonScaleWhen</code>	<code>f.buttonScaleWhen = scaleHow.always</code>
倍率	<code>buttonScaleHow</code>	<code>f.buttonScaleHow = scaleHow.proportional</code>
境界線に合わせる	<code>buttonFitBounds</code>	<code>buttonFitBounds = true;</code>
ボタン内のアイコンの位置	<code>buttonAlignX</code> および <code>buttonAlignY</code>	<code>f.buttonAlignX = 50;</code> <code>f.buttonAlignY = 50;</code>
アイコンとラベル		
状態	<code>buttonSetIcon</code>	<code>f.buttonSetIcon(i);</code>
ラベル	<code>buttonSetCaption</code> および <code>buttonGetCaption</code>	<code>f.buttonSetCaption("Push Me");</code>
アイコン	<code>buttonSetIcon</code>	表の注意事項を参照

フォームでの Acrobat JavaScript の使用

プログラムによるフォームフィールドの作成

表の注意事項

- 「アイコンとラベル」の「アイコンの選択」に対応する基本メソッドは `buttonSetIcon` です。このメソッドはアイコンをボタンの表面に関連付けますが、名前付きアイコンがあらかじめ PDF ファイルに含まれている必要があります。ボタンの表面にアイコンに関連付ける完全な例については、`Doc` オブジェクトの `addIcon` を参照してください。
- 名前付きアイコンをドキュメントに含めるには、`buttonImportIcon` を使用します。

「アクション」タブ

ボタンフィールドのアクションを設定するには、フィールドレベルの `setAction` メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseUp", "app.beep(0);")
```

チェックボックス

チェックボックスのプロパティダイアログボックスには、「一般」、「表示方法」、「オプション」、「アクション」の4つのタブがあります。

「一般」タブ

「一般」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[一般](#)」タブを参照してください。

「表示方法」タブ

「表示方法」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[表示方法](#)」タブを参照してください。ただし、`textFont` を選択できない点が異なります。チェックボックスの場合、フォントは常に Adobe Pi になります。

「オプション」タブ

「オプション」タブでは、フィールドで使用するチェックマークのスタイルや、書き出し値を設定することができます。

領域/名前	参照	例
チェックボックススタイル	<code>style</code>	<code>f.style = style.ci;</code>
書き出し値	<code>setFocus</code>	<code>f.setExportValues(["buy"]);</code>
チェックボックスをデフォルトでチェックする	<code>defaultIsChecked</code>	<code>f.defaultIsChecked(0);</code> 表の注意事項を参照

表の注意事項

- チェックボックスをデフォルトでチェックする：`defaultIsChecked` を使用しただけでは、フィールドは必ずしもチェックされません。フィールドをチェックするには、`this.resetForm([f.name])` を使用してフィールドをリセットするか、`checkThisBox` を使用して `f.checkThisBox(0)`; とします。

- チェックボックスをデフォルトでチェックする：「オプション」タブの「チェックボックスをデフォルトでチェックする」チェックボックスにチェックが付いているかどうかを判断するには、`isDefaultChecked` を使用します。
- チェックボックスフィールドがチェックされているかどうかを判断するには、`isBoxChecked` を使用します。

「アクション」タブ

「アクション」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「アクション」タブを参照してください。

コンボボックス

コンボボックスのプロパティダイアログボックスには、「一般」、「表示方法」、「オプション」、「アクション」、「フォーマット」、「検証」、「計算」の7つのタブがあります。

「一般」タブ

「一般」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「一般」タブを参照してください。

「表示方法」タブ

「表示方法」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「表示方法」タブを参照してください。

「オプション」タブ

「オプション」タブでは、コンボボックスの項目リストを編集できます。項目を追加して、その名前および対応する書き出し値を設定することができます。リスト内の項目の順序は、変更したりリソートしたりでき、項目は削除することもできます。これらの操作に直接対応する命令は限られていますが、メソッドとプロパティ（`setItems`、`numItems`、`getItemAt`、`insertItemAt`、`deleteItemAt`、`clearItems` および `currentValueIndices`）を使用して同様の操作を実行できます。

また、UI に用意されている次の設定についても、プログラムで直接設定することができます。

領域／名前	参照	例
カスタムテキストの入力を許可	<code>editable</code>	<code>f.editable = true;</code>
スペルチェック	<code>doNotSpellCheck</code>	<code>f.doNotSpellCheck = true;</code>
選択した値をすぐに確定	<code>commitOnSelChange</code>	<code>f.commitOnSelChange = true;</code>

表の注意事項

- 項目および書き出し値：`addField` でコンボボックスを作成後、`setItems` を使用して、項目名とその書き出し値を簡単に設定することができます。次に例を示します。

```
f.setItems([ ["California", "CA"], ["Ohio", "OH"],
["Arizona", "AZ"] ]);
```

- 項目の並べ替え: 「オプション」タブのこのチェックボックスに直接フックすることはできません。このチェックボックスは、UI を使用して項目が追加された際にリストを並べ替えるよう Acrobat に指示するものです。上の `setItems` の例では項目はアルファベット順に入力されませんが、配列オブジェクトのソートメソッドを使用すれば、リストをプログラムで並べ替えることができます。次に例を示します。

```
function compare (a,b) { // define a compare function
    if (a[0] < b[0] ) return -1;
    if (a[0] > b[0] ) return 1;
    return 0;
}
var tmp = new Array();
var f = this.getField("myCombo");
// load [item, exportvalue]
for (var i = 0; i < f.numItems; i++)
    tmp[i] = [f.getItemAt(i,false), f.getItemAt(i)];
tmp.sort(compare); // sort of first component
f.clearItems(); // out with the old
f.setItems(tmp); // in with the new
```

「アクション」タブ

「アクション」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「アクション」タブを参照してください。

「フォーマット」タブ

コンボボックスのアクションは、フィールドレベルの `setAction` に「Format」というトリガー名を指定して設定することができます。UI にはいくつかのフォーマットのカテゴリーがあります。次の表に JavaScript との対応を示します。カスタムを除くすべてのフォーマットは、`Javascripts¥aform.js` に含まれている関数で実現することができます。

領域/名前	参照	例
数値	<code>Javascripts¥aform.js</code> の <code>AFNumber_Format()</code>	<code>f.setAction("Format", 'AFNumber_Format(2, 0, 0, 0, "¥u20ac", true)');</code> <code>f.setAction("Keystroke", 'AFNumber_Keystroke(2, 0, 0, 0, "¥u20ac", true)');</code>
パーセント	<code>AFPercent_Format()</code>	
日付	<code>AFDate_FormatEx()</code>	
時間	<code>AFTime_Format()</code>	
特殊	<code>AFSpecial_Format()</code>	
カスタム		表の注意事項を参照

表の注意事項

- 数値: 表の例は、UI で小数点以下の桁数を 2 にし、3 桁ごとの区切りにコンマを使用し、通貨記号としてユーロを選択した場合に相当します。

- カスタム：上に示すフォーマット関数を使用しないフォーマットスクリプトは、すべてカスタムフォーマットスクリプトとして分類されます。カスタムキーストロークスクリプトを設定するには、**setAction** に「Keystroke」というトリガー名を指定します。

「検証」タブ

コンボボックスのアクションは、フィールドレベルの **setAction** に「Validate」というトリガー名を指定して設定することができます。検証の方法としては、入力値が一定の範囲に入っているか確認する方法と、カスタムスクリプトを実行して検証を行う方法の2つがあります。詳しくは、「Field/Validate」を参照してください。

領域／名前	参照	例
フィールド値の範囲を指定	Javascripts¥aform.js の AFRange_Validate()	<pre>f.setAction("Validate", 'AFRange_Validate(true, 0, true, 100)'); /* value between 0 and 100, inclusive */</pre>
カスタム検証スクリプトを実行		表の注意事項を参照

表の注意事項

- カスタム：**AFRange_Validate()** 関数を使用していない検証スクリプトは、すべてカスタムとして分類されます。

「計算」タブ

コンボボックスのアクションは、フィールドレベルの **setAction** に「Calculate」というトリガー名を指定して設定することができます。UI には 3 つの計算のカテゴリーがあります。これは次の表に示すように、JavaScript では **AFSimple_Calculate()** メソッドに対応しています。

領域／名前	参照	例
次のフィールドの和（積、平均、最小、最大）の値を計算	Javascripts¥aform.js の AFSimple_Calculate()	<pre>f.setAction("Calculate", 'AFSimple_Calculate("SUM", new Array ("line.1", "line.3"))');</pre>
カスタム		表の注意事項を参照

表の注意事項

- カスタム：**AFSimple_Calculate()** 関数を使用していない計算スクリプトは、すべてカスタムとして分類されます。

プログラミングに関するその他の注意

- コンボボックス（またはリストボックス）にある項目数を取得するには、**numItems** プロパティを使用します。
- 項目の表示名（項目名）や書き出し値を取得するには、**getItemAt** を使用します。

フォームでの Acrobat JavaScript の使用

プログラムによるフォームフィールドの作成

- コンボボックス（またはリストボックス）に新規の項目を追加するには、`insertItemAt` を使用します。
- コンボボックス（またはリストボックス）から項目を削除するには、`deleteItemAt` を使用します。
- コンボボックス（またはリストボックス）からすべての項目を削除するには、`clearItems` を使用します。
- コンボボックス（またはリストボックス）の現在値を変更するには、`currentValueIndices` を使用します。例えば、`f.currentValueIndices = 2` と指定すると、（0 ベースなので）3 つ目の項目がコンボボックスの現在値になります（フォームを送信した場合、その書き出し値がエクスポートされます）。

リストボックス

リストボックスのプロパティダイアログボックスには、「一般」、「表示方法」、「オプション」、「アクション」、「選択の変更」の 5 つのタブがあります。

「一般」タブ

「一般」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[一般](#)」タブを参照してください。

「表示方法」タブ

「表示方法」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[表示方法](#)」タブを参照してください。

「オプション」タブ

「オプション」タブのプロパティは、「カスタムテキストの入力を許可」と「スペルチェック」が使用できないこと、および「複数選択」が使用できることを除いて、コンボボックスの場合と同じです。

領域/名前	参照	例
複数選択	<code>multipleSelection</code>	<code>f.multipleSelection = true;</code>

表の注意事項 コンボボックスの[表の注意事項](#)を参照してください。

「アクション」タブ

「アクション」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[アクション](#)」タブを参照してください。

「選択の変更」タブ

リストボックスのアクションは、フィールドレベルの `setAction` に「Keystroke」というトリガー名を指定して設定することができます。

例：

```
f.setAction("Keystroke", "ProcessSelection()");
```

プログラミングに関するその他の注意

コンボボックスの[プログラミングに関するその他の注意](#)を参照してください。

ラジオボタン

ラジオボタンフィールドの作成には、Acrobat の UI または **Doc Object** の `addField` を使用できます。ラジオボタンフィールドは他のフォームフィールドと異なり、複数のフィールドを同じ名前で作成する必要があります。ラジオボタンフィールドをプログラムで作成するには、次のようにします。

```
this.addField("myRadio", "radiobutton", 0, [400, 442, 412, 430]);
this.addField("myRadio", "radiobutton", 0, [400, 427, 412, 415]);
var f = this.addField("myRadio", "radiobutton", 0,
    [400, 412, 412, 400]);
f.setExportValues(["Yes", "No", "Sometimes"]);
```

このようにすると、ページ 0 に 3 つのラジオボタンが作成されます。各ラジオボタンの幅は 12 ポイント、高さは 12 ポイントになり、フィールドにはデフォルトの外観が与えられます。各ボタンの書き出し値を定義するには、`setExportValues` を使用します。

ラジオボタンの UI はチェックボックスとまったく同じです。ラジオボタンフィールドの外観の変更方法や、オプションおよびアクションの設定方法については、「[チェックボックス](#)」の節を参照してください。

「一般」タブ

「一般」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[一般](#)」タブを参照してください。

「表示方法」タブ

「表示方法」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[表示方法](#)」タブを参照してください。

「オプション」タブ

ラジオボタンの「[オプション](#)」タブは、チェックボックスの場合と同じです。[表の注意事項](#)も参照してください。

「アクション」タブ

「アクション」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[アクション](#)」タブを参照してください。

電子署名

電子署名のプロパティダイアログボックスには、「[一般](#)」、「[表示方法](#)」、「[アクション](#)」、「[署名](#)」の 4 つのタブがあります。

「一般」タブ

「一般」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[一般](#)」タブを参照してください。

「表示方法」タブ

「表示方法」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「表示方法」タブを参照してください。

「アクション」タブ

「アクション」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「アクション」タブを参照してください。

「署名」タブ

署名フィールドのアクションは、フィールドレベルの `setAction` に「Format」というトリガー名を指定して設定することができます。フォームへの署名と関連する各フィールドの変更は、プログラムでは再フォーマットによって行うことができます。UI にはいくつかの署名のカテゴリがあります。次の表に JavaScript との対応を示します。カスタムを除くすべてのフォーマットは、`Aform.js` に含まれている関数で実現することができます。

領域/名前	参照	例
署名時に何も実行しない		デフォルトの動作。値はヌル。
読み取り専用指定	<code>setLock</code>	<code>oLock.action = "Include";</code>
選択	<code>setLock</code>	
署名フィールドに署名後に実行するスクリプト	<code>setAction</code>	

Acrobat 5 の例

次に、Acrobat 5 以降の `Aform.js` に定義されている `AFSignature_Format` メソッドを使用して、JavaScript で署名フィールドを作成し、署名し、ロックする例を示します。

```
// Create signature field dynamically
var f = this.addField("mySignature", "signature", 0,
    [200, 500, 500, 400]);
f.strokeColor = color.black;

// set it to lock when signed
f.setAction("Format",
    'AFSignature_Format("THESE", new Array ("mySignature"));');

var ppk-lite = security.getHandler("Adobe.PPKLite"); // choose handler
ppk-lite.login("dps017", "/C/signatures/DPSmith.pfx"); // login
f.signatureSign(ppk-lite, // sign it
    { password: "dps017",
      location: "San Jose, CA",
      reason: "I am approving this document",
      contactInfo: "dpsmith@adobe.com",
      appearance: "Fancy"});
ppk-lite.logout(); // logout
```

テキスト

テキストフィールドのプロパティダイアログボックスには、「一般」、「表示方法」、「オプション」、「アクション」、「フォーマット」、「検証」、「計算」の7つのタブがあります。

「一般」タブ

「一般」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[一般](#)」タブを参照してください。

「表示方法」タブ

「表示方法」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[表示方法](#)」タブを参照してください。

「オプション」タブ

「オプション」タブでは、さまざまな属性を使用してデフォルトテキストを設定できます。

領域/名前	参照	例
整列	alignment	<code>f.alignment = "center";</code>
デフォルト	defaultValue	<code>f.defaultValue = "Name: ";</code>
複数行	multiline	<code>f.multiline = true;</code>
長いテキストをスクロール	doNotScroll	<code>f.doNotScroll = true;</code>
リッチテキストフォーマットを許可	richText	<code>f.richText = true;</code>
最大文字数	charLimit	<code>f.charLimit = 40;</code>
パスワード	password	<code>f.password = true;</code>
ファイルの選択に使用する	exportValues	<code>f.fileSelect = false;</code>
スペルチェック	doNotSpellCheck	<code>f.doNotSpellCheck = true;</code>
マス目で区切る	comb	<code>f.comb = true;</code> 注意：マス目で区切った場合の最大文字数は、 <code>f.charLimit</code> で指定します。

「アクション」タブ

「アクション」タブに表示されるプロパティは、ボタンの場合と同じです。詳しくは、ボタンの「[アクション](#)」タブを参照してください。

「フォーマット」タブ

「一般」タブは、コンボボックスの場合と同じです。

フォームでの Acrobat JavaScript の使用

プログラムによるフォームフィールドの作成

「検証」タブ

「検証」タブは、コンボボックスの場合と同じです。

「計算」タブ

「計算」タブは、コンボボックスの場合と同じです。

A

Acrobat JavaScript に関する簡単な FAQ¹

JavaScript はどこにあり、どのように使用するのですか？

JavaScript は、Acrobat のさまざまなレベル（フォルダレベル、文書レベル、フィールドレベル、およびバッチレベル）で動作します。それぞれのレベルで可能な処理は異なり、ロードされるタイミングも違います。

フォルダレベルの JavaScript

拡張子を「.js」とした個々のファイルに JavaScript を配置することができます。ビューアの起動時にこれらのファイルを読み込むには、Acrobat アプリケーションの **JavaScripts** フォルダまたはユーザの **JavaScripts** フォルダ（グローバル変数の保存ファイル `glob.js` を JavaScript で生成すると、ユーザの JavaScripts フォルダが作成されます）にファイルを入れておく必要があります。アプリケーションの起動時にファイルを読み込む順序について詳しくは、『[Acrobat JavaScript Scripting Reference](#)』の「[Event オブジェクト](#)」の節の [App/Init](#) の説明を参照してください。

アプリケーションで使用する便利な変数や関数は、これらのフォルダに配置してください。JavaScript メソッドの中には、[addItem](#) や [addSubMenu](#) のように、起動時にフォルダレベルの JavaScript ファイルからしか実行できないものもあるので注意してください。

JavaScript ファイルを作成する場合、いくつかの制約がありますが、特にデフォルトの [this オブジェクト](#) の使用に関して注意が必要です。

Acrobat JavaScript の標準の実装には、5 つの JavaScript ファイルが含まれています。組み込みの関数を含む `Aform.js`、注釈プラグインで 사용되는 `Annots.js`、そして ADBC プラグインで 사용되는 `ADBC.js` などです。これらのファイルは Acrobat の JavaScripts フォルダに入っています。

`glob.js` はプログラムによって生成されるファイルで、グローバルオブジェクトの [setPersistent](#) によってセッション間のアプリケーション設定が保存されます。

`Config.js` ファイルがある場合、これはビューアのツールバーボタンやメニュー項目を削除して、外観をカスタマイズするために使用されます（[hideMenuItem](#) および [hideToolBarButton](#) を参照）。

文書レベル

Adobe Acrobat の [アドバンスト / JavaScript / 文書レベル JavaScript の編集](#) を使用して、文書レベルのスクリプトを追加、変更、削除することができます。ここには文書外に適用する関数ではなく、文書内全般から使用するのに便利な関数を定義します。文書レベルのスクリプトは、文書が開かれてフォルダレベルのスクリプトが読み込まれた後に実行されます。文書レベルのスクリプト

1. よくある質問

Acrobat JavaScript に関する簡単な FAQ

フォームフィールドに名前を付けるときにはどのようにすればいいですか？

リプトは PDF 文書内に保存されます。フォームを作成するプログラマは、スクリプトをできるだけ効果的にパッケージする必要があります。

また、FDF ファイルにも文書レベルのスクリプトを含めることが可能です。さらに、FDF がインポートされる直前または直後に実行されるように指定したスクリプトを含めることもできます。

フィールドレベル

フォームの編集ツールにあるダイアログボックスを使用して、スクリプトをフォームフィールド定義に追加できます。これらのスクリプトは、イベント（マウスボタンを放す、または計算など）が発生すると実行されます。フィールド固有のスクリプトはこのレベルで作成する必要があります。これらのスクリプトは通常、フィールドの値を検証、フォーマット、あるいは計算するものです。

文書レベルおよびフィールドレベルのスクリプトは、フォルダレベルスクリプトとは異なり、PDF 文書内に保存されます。したがって、パフォーマンスやファイルサイズの原因により、フォームを作成するプログラマはスクリプトをさまざまなレベルで、（コードの再利用などをして）できるだけ効果的にパッケージする必要があります。

フォームフィールドに名前を付けるときにはどのようにすればいいですか？

フォームフィールドには、**FirstName** や **LastName** というような名前を付けるのが一般的です。このような形式の名前をフラットネームと呼びます。多くのフォームアプリケーションでは、この名前の階層だけで十分であり、問題なく動作します。フラットネームの問題点は、フィールド間の関連付けがないということです。

階層構造を作成すると、フォームフィールド名をさらに便利に使えるようになります。例えば **FirstName** を **Name.First** に変更し、**LastName** を **Name.Last** に変更することで、フィールドのツリーを形成することができます。ピリオド（「.」）は、Acrobat Forms で階層の区切りを表すために使用されるセパレータです。これらのフィールドの **Name** の部分が階層の親で、**First** および **Last** が子になります。名前の階層の深さに制限はありませんが、管理可能な範囲にしておくことが重要です。

この階層はさまざまな用途に便利に使えます。たとえば作成作業の効率を上げたり、JavaScript でフィールドのグループを簡単に操作することができます。また、文書に多数のフィールドが含まれている場合、フォームフィールドの階層を使用することで、フォームアプリケーションのパフォーマンスを向上させることができます。

フラットネームである **FirstName**、**MiddleName**、**LastName** を使用して、複数のフィールドの背景色を黄色に変更するとします（データが欠落していることを示したり、重要な部分を強調するため）。この場合、各フィールドの処理ごとに 2 行の JavaScript コードを必要とします。

```
var name = this.getField("FirstName");
name.fillColor = color.yellow;
name = this.getField("MiddleName");
name.fillColor = color.yellow;
name = this.getField("LastName");
name.fillColor = color.yellow;
```

これを上記の `Name.First`、`Name.Middle`、`Name.Last`（他に `Name.Title` などを付け足すこともできます）という階層を使用すると、6 行のコードをわずか 2 行にして背景色を変更することができます。

```
var name = this.getField("Name");
name.fillColor = color.yellow
```

この階層を JavaScript で操作する場合、変更できるのは親フィールドの外観に関するプロパティだけであり、外観に加えた変更はすべての子に適用されることに注意してください。フィールドの値は変更できません。例えば、次のようなスクリプトを試してみます。

```
var name = this.getField("Name");
name.value = "Lincoln";
```

このスクリプトは失敗します。なぜなら `Name` は親フィールドだからです。値を変更できるのは、階層の最後のフィールド（`Last` や `First` など、子を持たないフィールド）だけです。次のスクリプトは正しく実行されます。

```
var first = this.getField("Name.First");
var last = this.getField("Name.Last");
first.value = "Abraham";
last.value = "Lincoln";
```

日付オブジェクトはどのように使用するのですか？

ここでは、Acrobat の `Date` オブジェクトについて説明します。このマニュアルでは、JavaScript の `Date` オブジェクトおよび日付を処理する `Util` オブジェクト関数を十分に理解していることが前提となります。JavaScript の `Date` オブジェクトは、実際には日付と時刻の両方を含むオブジェクトです。ですから、ここで「日付」と言う場合、日付と時刻の組み合わせを意味しています。

注意： このマニュアルに記載されているメソッドも含め、JavaScript の日付処理はすべて 2000 年問題（Y2K）に対応済みです。

注意：（ヒント）`Date` オブジェクトを使用する場合、現在の年から 1900 を引いた値を返す `getFullYear()` メソッドを使用するのではなく、常に 4 桁の年を返す `getFullYear()` メソッドを使用してください。次に例を示します。

```
var d = new Date();
d.getFullYear();
```

日付から文字列への変換

JavaScript の `Date` オブジェクトに加えて、Acrobat Forms には日付に関連するメソッドがいくつか用意されています。`Date` オブジェクトと文字列との間の変換には、これらのメソッドを使用することもできます。Acrobat Forms は数多くの日付フォーマットを扱っているため、`Date` オブジェクトではこれらを正しく変換できないことがあります。

`Date` オブジェクトを文字列に変換するには、`Util` オブジェクトの `printd` を使用します。組み込みの変換と異なり、`printd` では日付を指定通りにフォーマットすることができます。

```
/* Example of util.printd */
var d = new Date(); // Create a Date object containing the current date
/* Create some strings from the Date object with various formats with
** util.printd */
var s = [ "mm/dd/yy", "yy/m/d", "mmmm dd, yyyy", "dd-mmm-yyyy" ];
```

Acrobat JavaScript に関する簡単な FAQ

日付オブジェクトはどのように使用するのですか？

```
for (var i = 0; i < s.length; i++) {
    /* print these strings to the console */
    console.println("Format " + s[i] + " looks like: "
        + util.printd(s[i], d));
}
```

このスクリプトの出力は、次のようになります。

```
Format mm/dd/yy looks like:01/15/99
Format yy/m/d looks like:99/1/15
Format mmmm dd, yyyy looks like:January 15, 1999
Format dd-mmm-yyyy looks like:15-Jan-1999
```

注意：（ヒント）人間の寿命は延びていますし、Y2K の教訓もあります。4 桁の年で日付を出力するようにして、あいまいさをなくすようにしてください。

文字列から日付への変換

文字列を `Date` オブジェクトに変換するには、`Util` オブジェクトの `scand` を使用します。このメソッドは、文字列に含まれる年、月、日を取得するための手がかりとして、フォーマット文字列を引数に取ります。

```
/* Example of util.scand */
/* Create some strings containing the same date in differing
formats. */
var s1 = "03/12/97";
var s2 = "80/06/01";
var s3 = "December 6, 1948";
var s4 = "Saturday 04/11/76";
var s5 = "Tue. 02/01/30";
var s6 = "Friday, Jan. the 15th, 1999";
/* Convert the strings into Date objects using util.scand */
var d1 = util.scand("mm/dd/yy", s1);
var d2 = util.scand("yy/mm/dd", s2);
var d3 = util.scand("mmmm dd, yyyy", s3);
var d4 = util.scand("mm/dd/yy", s4);
var d5 = util.scand("yy/mm/dd", s5);
var d6 = util.scand("mmmm dd, yyyy", s6);
/* Print the dates to the console using util.printd */
console.println(util.printd("mm/dd/yyyy", d1));
console.println(util.printd("mm/dd/yyyy", d2));
console.println(util.printd("mm/dd/yyyy", d3));
console.println(util.printd("mm/dd/yyyy", d4));
console.println(util.printd("mm/dd/yyyy", d5));
console.println(util.printd("mm/dd/yyyy", d6));
```

このスクリプトの出力は、次のようになります。

```
03/12/1997
06/01/1980
12/06/1948
04/11/1976
01/30/2002
01/15/1999
```

`scand` は日付コンストラクタ (`new Date(...)`) と異なり、渡される文字列に関する規則が厳密ではありません。

注意： `scand` に 2 桁の年が与えられた場合、あいまいさを解決する処理が行われます。つまり、年が 50 未満の場合は 21 世紀（2000 を加える）であるとみなされ、50 以上の場合は 20 世紀（1900 を加える）とみなされます。この手法は Date Horizon と呼ばれることがあります。

日付の演算

日付の演算を行って、2 つの日付の間の経過時間や、数日後あるいは数週間後の日付を求めたいことがあります。JavaScript の `Date` オブジェクトには、この処理を行う方法がいくつか用意されています。最も単純で分かりやすい方法は、日付を数値表現によって処理する方法です。JavaScript では、日付は固定日時からのミリ秒（1000 ミリ秒は 1 秒）の数値で内部的に保持されており、`Date` オブジェクトの `valueOf` メソッドによって取得することができます。`Date` コンストラクタを使用して、この数値から新しい日付を構築することもできます。

```
/* Example of date arithmetic. */
/* Create a Date object with a definite date. */
var d1 = util.scand("mm/dd/yy", "4/11/76");
/* Create a date object containing the current date. */
var d2 = new Date();
/* Number of seconds difference. */
var diff = (d2.valueOf() - d1.valueOf()) / 1000;
/* Print some interesting stuff to the console. */
console.println("It has been "
    + diff + " seconds since 4/11/1976");
console.println("It has been "
    + diff / 60 + " minutes since 4/11/1976");
console.println("It has been "
    + (diff / 60) / 60 + " hours since 4/11/1976");
console.println("It has been "
    + ((diff / 60) / 60) / 24 + " days since 4/11/1976");
console.println("It has been "
    + (((diff / 60) / 60) / 24) / 365 + " years since 4/11/1976");
```

このスクリプトの出力は、次のようになります。

```
It has been 718329600 seconds since 4/11/1976
It has been 11972160 minutes since 4/11/1976
It has been 199536 hours since 4/11/1976
It has been 8314 days since 4/11/1976
It has been 22.7780821917808 years since 4/11/1976
```

次に日付を加算する例を示します。

```
/* Example of date arithmetic. */
/* Create a date object containing the current date. */
var d1 = new Date();
/* num contains the numeric representation of the current date. */
var num = d1.valueOf();
/* Add thirteen days to today's date, in milliseconds. */
/* 1000 ms/sec, 60 sec/min, 60 min/hour, 24 hours/day, 13 days */
num += 1000 * 60 * 60 * 24 * 13;
/* Create our new date, 13 days ahead of the current date. */
var d2 = new Date(num);
/* Print out the current date and our new date using util.printd */
console.println("It is currently: "
    + util.printd("mm/dd/yyyy", d1));
console.println("In 13 days, it will be: "
    + util.printd("mm/dd/yyyy", d2));
```

Acrobat JavaScript に関する簡単な FAQ

どうすれば文書を保護できますか？

このスクリプトの出力は、次のようになります。

```
It is currently:01/15/1999
In 13 days, it will be: 01/28/1999
```

どうすれば文書を保護できますか？

Acrobat が持つセキュリティ機能としては、まず文書に対するアクセス制限、文書が開かれた後はフォームに対する権限の制限、そして電子署名があります。

文書に対するアクセス制限

フォームへのアクセスを完全に制限したい場合、Acrobat の標準セキュリティモデルを使用して、ユーザにパスワード入力を義務付けることができます。サードパーティからプラグインとして提供されているセキュリティハンドラの中には、便利なものもあります。例えば、公開／秘密鍵のインフラストラクチャを使用して、特定のユーザに対してフォームをロックしたり、一定期間が経過した時点でフォームを無効にすることもできます。

ユーザパスワードを設定するには、**Acrobat のファイル／文書のプロパティ ...** を選択し、左側のメニューから**セキュリティ**を選択します。続いて、ドロップダウンメニューから「**パスワードによるセキュリティ**」を選択します。すると、パスワードと権限を希望どおりに設定できるようになります。

権限の制限

Acrobat の標準セキュリティモデルへのアクセスは文書の保存時に行われ、このとき文書に対する制限が設定されます。制限を設定できるものには、文書の印刷、文書の変更、テキストおよびグラフィックの選択、注釈およびフォームフィールドの追加 / 変更があります。

フォームを作成した後は、データの入力はできてもフォームツールで改ざんされないようにするため、フォームをロックしておくのが便利です。例えば、フォームの作成後に Web サイトで公開するような場合がそうです。フォームの信頼性を保つには、フォーム内の式やデータ関数を変更されないように保護する必要があります。

「文書の変更を許可しない」というオプションを選択すると、ユーザはフォームにデータを入力したり注釈を追加したりすることはできますが、フォームフィールドの作成や変更をしたり、TouchUp プラグインで背景のテキストを変更したりすることはできなくなります。

また、フォームにデータを入力し終えた後は、文書を変更できないようにするため、文書全体をロックするのが望ましい場合もあります。税金申告や機密文書にデータを入力した場合は、文書を保存した後で他の変更を加えることができないようにすべきです。データの入力と作成を禁止するには、「文書の変更を許可しない」および「注釈とフォームフィールドの追加や変更を許可しない」というオプションを選択してください。

電子署名

電子署名フォームフィールドは、アクセスや権限を制限することはしませんが、フォーム作成者またはユーザは、文書に署名が付けられた後で変更が加えられていないかを検証することができます。

制約のある Acrobat JavaScript メソッドをユーザが使用できるようにするにはどうすればよいですか？

フォーム作成者はデータ入力用のフォームをリリースする前に、フォームに電子署名を付けることができます。ユーザは、フォームが改ざんされておらず「公式」なものであることを署名によって確認できます。

隠し署名（表示されない署名）はこのような場合に便利で、電子署名メニューのサブメニューから作成することができます。

また、ユーザはフォームにデータを入力した後、電子署名ツールまたは事前に用意されている署名フィールドを使用して文書に署名を付けることができるので、知らない間にフォームが変更されるのを防ぐことができます。

電子署名フィールドをプログラムから作成および署名する方法について詳しくは、[80 ページの電子署名フィールドの説明](#)も参照してください。また、[81 ページの「署名フィールドに署名した後、どうすれば文書をロックできますか？」](#)も参照してください。

制約のある Acrobat JavaScript メソッドをユーザが使用できるようにするにはどうすればよいですか？

Acrobat JavaScript のメソッドの多くは、セキュリティ上の理由により制限されており、バッチ処理、コンソール、またはメニューイベントでのみ実行できるようになっています。この制約があるメソッドは、『Acrobat JavaScript Scripting Reference』のクイックバーにⓈの記号が付けられています。これらのメソッドが必要なソリューションを開発する企業ユーザにとって、この制約は、環境のセキュリティが確保されていると分かっている場合には足かせになります。

制約のある Acrobat JavaScript メソッドをユーザが使用できるようにするには、3つの条件を満足する必要があります。

- 開発者がデジタル ID を取得する。
- 開発者が、制約のある JavaScript メソッドが含まれている PDF 文書に、そのデジタル ID を使用して署名する。

デジタル ID の入手先と、その ID を使用して文書に署名する方法について詳しくは、『Adobe Acrobat 6.0 ヘルプ』の「デジタル ID の作成」を参照してください。

- 文書を受け取ったユーザが、認証された文書と JavaScript について署名者を信頼する。詳しくは、『Adobe Acrobat 6.0 ヘルプ』の「デジタル ID 証明書の管理」を参照してください。

信頼済みのすべての証明書には、Acrobat メインメニューの**アドバンスト／デジタル ID の管理／信頼済み証明書**から「証明書」を選択してアクセスできます。編集をクリックすると証明書が開いて表示されます。JavaScript を実行可能にするには、「JavaScript を信頼する」のチェックボックスにチェックを付けます。

署名フィールドに署名した後、どうすれば文書をロックできますか？

署名フィールドを使用すると、文書に電子署名を付けることができます。文書に署名を付けた後で文書に変更を加えると、「署名後に文書が変更された」ことが署名に示されます。

署名フィールドの値は読み取り専用で、署名されていなければヌル値、署名した場合は非ヌル値になります。署名フィールドに署名がされたときに動作させるカスタムスクリプトを作成する場

Acrobat JavaScript に関する簡単な FAQ

どうすれば文書へのアクセスを簡単にできますか？

合、フォームがリセットされる（フィールドの値がヌル値に設定されるなど）可能性も必ず考慮してください。

signature フィールドに署名がされたときにどのフィールドをロックするかは、**Field オブジェクト**の **setLock** メソッドで制御します。

さらに、**AForm.js** には Acrobat 5 から使用可能な **AFSignatureLock()** という便利な関数があり（「**JavaScript はどこにあり、どのように使用するのですか？**」を参照）、署名のプロパティダイアログで行う単純なロック操作を、プログラムから実行できます。この関数を使用すると、全フィールド、特定のフィールド、あるいは特定のフィールド以外の全フィールドを簡単にロックおよびアンロックできます。以下はこの関数を使用したコード例です。

```
var bLock = (event.value != "");
AFSignatureLock(this, "THESE", "A", bLock);
AFSignatureLock(this, "THESE", "B", !bLock);
```

詳しくは、**AForm.js** のコメントを参照してください。

どうすれば文書へのアクセスを簡単にできますか？

電子情報のアクセシビリティは、重要性がますます高まっている問題です。アクセシビリティに関する以下のヒントに従ってフォームを作成すれば、すべてのユーザにとってより使いやすいフォームになるでしょう。Acrobat のリリース 4.05 およびそれ以降では、身体や視覚に障害のあるユーザでも Acrobat Forms に入力できるように設計されています。Acrobat のバージョン 6 では、音声認識機能がこれまでになく充実しています。

フォームのアクセシビリティを確保するために、少なくとも次のガイドラインに従ってください。

文書のメタデータ

ファイル／文書のプロパティ／概要または**アドバンスド／文書メタデータ**を使用して、文書のメタデータを設定できます。

文書のオープン、保存、印刷、クローズといった操作を行うと、文書のタイトルが音読されるようにすることができます。「文書メタデータ」でタイトルが設定されていない場合はファイル名が音読されますが、ファイル名は短縮されたり変更されたりすることがよくあるので、文書の作成者はタイトルを設定するようにしてください。例えば、ファイル名が「**IRS1040.pdf**」のような分かりにくい名前である場合には、「Form 1040:U.S. Individual Income Tax Return for 1998」のようなタイトルを付けておくとうっかりやすくなります。

また、サードパーティ製のスクリーンリーダーは、通常、ウィンドウのタイトルバーのタイトルを読み上げます。タイトルバーに表示する内容は、**ファイル／文書のプロパティ／開き方を選択**し、「ウィンドウオプション」の「表示」で「**ファイル名**」または「**文書のタイトル**」を選択して設定できます。

文書に関連するすべてのメタデータ（作成者、サブタイトル、キーワード）を入力しておく、Acrobat Search やインターネット検索エンジンでその文書が検索しやすくなります。

フィールドの説明

非表示フィールド以外のすべてのフィールドには、分かりやすい説明（ツールヒント）を含めるようにします。この説明はユーザがフィールドのフォーカスを取得したときに音読されるので、フィールドの目的を示すものでなければなりません。例えば、フィールド名が「**name.first**」であれば、「**First Name**」というような説明が適しています。周囲の状況を確認しないと意味が確定しないような名前は避けてください。例えば、文書の本人欄と配偶者欄の両方に「**First Name**」フィールドがある場合、配偶者（Spouse）欄の名前は「**Spouse's First Name**」のようにします。この説明は、ユーザがマウスポインタをフィールドに置いたときにもツールヒントとしても表示されます。

タブ順序の設定

文書の内容を効率的にたどっていくには、フィールドのタブ順序が論理的に設定されている必要があります。障害を持つほとんどのユーザはタブを使用して文書内を移動するので、その順序は特に重要です。視覚に障害のあるユーザの場合は、マウスポインタを目的の位置に移動させたり、視覚的な合図を確認したりできないので、これは必須条件となります。

キーボードフォーカスがフォームフィールドに存在しないとき Tab (Shift+Tab) キーを押すと、現在のページでタブオーダーの最初（最後）にあるフィールドがアクティブになります。ページにフォームフィールドが存在しない場合、このことが音声によって通知されます。

フィールドにフォーカスがあるときに Tab (Shift+Tab) キーを押すと、タブオーダーに従って次（前）のフィールドに移動します。ページの最後（最初）のフィールドで Tab (Shift+Tab) キーを押すと、次（前）のページ（存在する場合）の最初（最後）のフィールドにフォーカスが移ります。次（前）のフィールドが存在しない場合、フォーカスは現在のページで「ループ」して、最初（最後）のフィールドに移動します。

読み上げ順序

文書を読み上げる順序は、タグツリーで指定します。視覚に障害のあるユーザがフォームを効果的に使用するには、ページの内容とフィールドがタグツリーに含まれている必要があります。タグツリーを使用して、ページ上のフィールドのタブ順序を示すこともできます。

どうすれば JavaScript でグローバル変数を定義できますか？

Acrobat には JavaScript 用の拡張機能として **global** オブジェクトが定義されており、これにグローバル変数をプロパティとして格納することができます。「**myVariable**」という新しいグローバル変数を定義し、それにヌル文字列を代入するには、次のようにします。

```
global.myVariable = "";
```

文書内のどのスクリプトからでもこの変数を参照することができます。

グローバル変数の永続化

通常、グローバル変数のグローバルデータは、ユーザセッション間では維持されませんが、**Global** オブジェクトに備わっているメソッドを使うと、セッション間でデータを維持することができます。「**myVariable**」という変数をセッション間で維持するには、次のようにします。

Acrobat JavaScript に関する簡単な FAQ

どうすればフォームデータを電子メールアドレスに送信できますか？

```
global.setPersistent("myVariable", true);
```

このようにすると、以降のセッションでも引き続き変数は存在するようになります（前の値が保持されます）。

どうすればフォームデータを電子メールアドレスに送信できますか？

これには、**Field** オブジェクトの **submitForm** を使用します。

```
var url = "mailto:johndoe@doe.net";  
this.submitForm(url, false);
```

この場合、フォームの内容は変数 `url` が示すアドレスに送信されます。`submitForm()` の 2 つ目の引数は、フォーム内容を URL エンコードしたデータとして POST メソッドで送信するのか、あるいは FDF (Forms Data Format) として送信するのかを示します。「false」は URL エンコード形式で送信することを示します。

どうすれば他のフィールドの値に応じてフィールドを非表示にできますか？

Field オブジェクトの **display** を使用します。

```
var f = this.getField("title");  
if (this.getField("showTitle").value == "Off")  
    f.display = display.hidden;  
else  
    f.display = display.visible;
```

どうすれば別にかかれているフォームのフィールド値を現在のフォームから参照できますか？

Global オブジェクトの **subscribe** メソッドを使用して、目的のフィールドを他のフィールドから参照することができます。例えば、目的のフィールドのグローバル値を設定する文書レベルのスクリプト（文書を初めて開いたときに起動される）をフォーム（文書 B）に設定します。

```
function PublishValue( xyzForm_fieldValue_of_interest ) {  
    global.xyz_value = xyzForm_fieldValue_of_interest;  
}
```

そして、現在の文書（文書 A）から文書 B の目的の値にアクセスするには、その変数を `subscribe` します。

```
global.subscribe("xyz_value", ValueUpdate);
```

`ValueUpdate` は、`xyz_value` の内容が変更されるたびに自動的に呼び出されるユーザ定義関数です。文書 A の `MyField` というフィールドで `xyz_value` の値を使用している場合、コールバック関数を次のように定義します。

```
function ValueUpdate( newValue ) {  
    this.getField("MyField").value = newValue;}  
}
```

どうすればキー入力を1つずつインターセプトできますか？

カスタムキーストロークスクリプトを作成し（テキストフィールドまたはコンボボックスの**プロパティ**ダイアログで「フォーマット」タブを参照）、`event.change` の値を調べます。この値を変更すると、ユーザ入力の内容を入力時に変更することができます。

どうすればネストされたポップアップメニューを作成できますか？

`app.popUpMenu()` メソッドを使用します。メニュー選択の配列を作成し、フィールドの「マウスボタンを押す」または「マウスボタンを放す」イベントから `app.popUpMenu(arrayName)` を呼び出し、メニューをポップアップさせます。Acrobat 6.0 では、`app.popUpMenuEx()` メソッドも使用できます。

例：

```
var cChoice = app.popUpMenu("one", "two", "-",
    [ "three", "three.one", "three.two" ] );
app.alert("You chose " + cChoice);
```

どうすれば独自の色を作成できますか？

色は配列オブジェクトであり、配列の最初の項目は色領域を表す文字列（「G」はグレースケール、「RGB」は RGB、「CMYK」は CMYK）で、その後の項目はそれぞれの色領域の成分を表す数値です。したがって、次のようになります。

```
color.blue = new Array("RGB", 0, 0, 1);
color.cyan = new Array("CMYK", 1, 0, 0, 0);
```

カスタムカラーを作成するには、使用するカラースペースのタイプとチャンネル値の配列を宣言するだけです。

どうすればユーザ入力を促すダイアログを表示できますか？

`App` オブジェクトクラスに定義されている `response` を使用します。このメソッドで表示されるダイアログには、質問と、それに応答するための入力フィールドが含まれます（オプションとして、ダイアログタイトルや、応答のデフォルト値を表示することもできます）。戻り値は、ユーザの応答を含む文字列です。ユーザがダイアログの「キャンセル」ボタンをクリックすると、応答は `null` オブジェクトになります。

```
var dialogTitle = "Please Confirm";
var defaultAnswer = "No.";
var reply = app.response("Did you really mean to type that?",
    dialogTitle, defaultAnswer);
```

どうすれば JavaScript で URL を取り出せますか？

Doc オブジェクトクラスの `getURL` を使用します。このメソッドは、GET を使用して指定の URL をインターネットから取り出します。現在の文書をブラウザ内で表示しているか、Acrobat Web Capture が使用できない場合は、Weblink プラグインを使用して目的の URL を取り出します。

どうすればフィールドのホットヘルプテキストを動的に変更できますか？

Field オブジェクトの `userName` を使用して、フィールドの「ツールヒント」を文字列として取得 / 設定できます。このプロパティは、マウスカーソルをフィールドに置いたときに簡単な説明（ホットヘルプ）を表示するためのものです。便利なアドバイス（あるいは何らかの説明）を `userName` に設定すると非常に効果的です。

どうすればズーム倍率をプログラムで変更できますか？

Doc オブジェクトクラスの `zoom` を使用します。例えば、次のコードでは、現在のページのズーム倍率を 2 通りの方法で設定しています。

```
// This zooms in to twice the current zoom level:  
this.zoom *= 2;  
  
// This sets the zoom to 73%:  
this.zoom = 73;
```

なお、ズーム倍率として指定する値の単位はパーセントです。したがって、73% は 0.73 ではなく 73 と指定します。

どうすればマウスが特定の領域に入った（あるいは出た）かを判断できますか？

マウスの出入りを調べたい場所に、非表示の読み取り専用テキストフィールドを作成します。次に、フィールドの「ポインタを範囲内に入れる」および「ポインタを範囲外に出す」アクションに JavaScript を割り当てます。

索引

A

Acrobat
アプリケーション 17
プラグイン 14
Acrobat Database Connectivity。「ADBC オブジェクト」も参照。
JavaScript 60
JavaScript で 57
オプションの指定 57
ADBC
「ADBC オブジェクト」も参照
ADBC オブジェクト 19
「ADBC」も参照
App オブジェクト 17

C

Connection オブジェクト 19
Console オブジェクト 18, 22
println() メソッド 22
show() メソッド 22

D

Doc オブジェクト 17

E

ECMAScript 14
eForm 13

G

Global オブジェクト 18

H

HTML JavaScript 14

J

JavaScript
アクションの操作 61
アクションの割り当て 57
アクション、「すべてを編集」62
アクション、「編集」61
エディタ 21, 61
簡単な JavaScript の作成 55
簡略化したフィールド表記 56

計算 56
コア 14
削除 61
自動日付フィールド 55
使用可能にする 29
使用可能または使用不可にする 55
整形 22
非アクティブなフィールド 59
非表示のフィールド 59
フィールドレベル 60
フィールドレベルのスクリプト 9
フォームや文書を電子メールで送信 58
プラグインレベル 9, 60, 61
文書レベル 9, 60
文書を電子メールで送信 58
読み取り専用のフィールド 60
JavaScript エディタ 21
外部の 25
組み込みの 26
JavaScript オブジェクト 17
ADBC 19
App 17
Connection 19
Console 18, 22
Doc 17
Global 18
Statement 19
Util 19
データベース 19
JavaScript コンソール 22, 29
JavaScript コンソールコマンド 61
JavaScript 文書コマンド 60
JavaScript を使用可能にする 29

P

PDF 文書。「文書」も参照

S

Statement オブジェクト 19

U

Util オブジェクト 19

え

永続的データ 18
エディタ 21

お

オブジェクト
「JavaScript オブジェクト」も参照
オブジェクト階層 17

か

階層 17
「カスタム検証スクリプトを実行」オプション 59
関数
 便利な 17
簡略化したフィールド表記 56

け

継承階層 17
検証、JavaScript で 59

こ

コア JavaScript 14
高度な編集ツールバー 42
コード
 整形 22
 テスト 18
 デバッグ 18
コメント 14
コメントレポジトリ 14

さ

算術計算
 計算、算術 56

す

スクリプト 21

せ

整形
 コード 22

ち

チームによるオンラインレビュー機能 14
注釈 14

つ

ツールバー
 高度な編集 42

て

データベース
 オブジェクト 19
テスト 18, 21
デバッグ 18, 21

ひ

非アクティブなフィールド 59
引き算および割り算、フォーム内 56
非表示のフィールド 59
表記、簡略化したフィールド 56

ふ

フィールド表記、簡略化した 56
フィールドレベルの JavaScript 9, 60
フォーム
 計算 56
 「最後のページに移動」ボタン 57
 「次のページに移動」ボタン 57
 非アクティブなフィールド 59
 引き算および割り算 56
 非表示のフィールド 59
 フォームや文書を電子メールで送信 58
 「前のページに移動」ボタン 57
 読み取り専用のフィールド 60
プラグインレベルの JavaScript 9, 60, 61
文書
 操作 17
 「文書のアクションを設定」コマンド 61
 「文書レベル JavaScript の編集」コマンド 55
文書レベルの JavaScript 9, 60

へ

編集ツールバー、高度な 42

ほ

包含階層 17

め

メソッド
 console.println() 22
 console.show() 22

ゆ

ユーティリティ関数 17

よ

読み取り専用のフィールド 60

