事例に見る SCORM 相互運用性向上のための応用技術

第 1.2 版

2005 年 8 月

日本イーラーニングコンソシアム

標準化推進委員会

改訂履歴

日付	バージョン	改訂内容
2004 年 6 月	1.0	新規作成
2004 年 6 月	1.02	● 以下を追加. 3.1.1 脚注. ● 誤字修正
2005 年 1 月	1.1	● 以下を追加. 2.1.4, 2.4.9, 2.5.2, 2.5.3, 3.1.3, 3.4.6, 3.5.2, 3.5.3.
2005 年 8 月	1.2	● 以下を追加. 2.1.1 BOM の記述，2.2.2, 2.4.10, 3.1.1 脚注を本文に移し BOM の記述を追加，3.2.2, 3.4.7.

目次

1. はじめに **1**

2. 事例 **2**

2.1 manifestファイル **2**

2.1.1 manifestファイルの文字コードの相違 2

2.1.2 SCOのURLの大文字・小文字の区別 2

2.1.3 SCOのURLの絶対パス・相対パス 2

2.1.4 LMSの教材階層数の制限 2

2.2 SCOの起動 **2**

2.2.1 FindAPIの検索順 2

2.2.2 APIアダプタのロード手順 2

2.3 API関数 **2**

2.3.1 LMSInitializeの引数がない 2

2.3.2 LMSInitializeの返り値がBoolean 3

2.3.3 LMSInitializeとLMSFinishの繰り返し 3

2.4 データモデル **3**

2.4.1 lesson_status のボキャブラリーが不正 3

2.4.2 lesson_statusの動作が実装されていないLMS 3

2.4.3 lesson_statusの値で学習時間を計時するLMS 3

2.4.4 前提条件とlesson_statusの値 3

2.4.5 cmi.core.score.rawの値が範囲外 3

2.4.6 masteryscoreが 0 の時のLMSの動作 3

2.4.7 masteryscoreが空の時のLMSの動作 3

2.4.8 実装していないオプションデータ要素の扱い 3

2.4.9 リストデータ項目の添え字の順序 4

2.4.10 lesson_statusの解釈 4

2.5 その他 **4**

2.5.1 SCORM準拠LMSなのに教材が移植できない 4

2.5.2 コマンドGUIがLMSによって異なる 4

2.5.3 既存コンテンツの動作が移植によって再現できなくなる 4

3. 解説 **5**

3.1 manifestファイル **5**

3.1.1 **文字コード** 5

3.1.2 **リソースの**URL 7

3.1.3 **コンテンツアグリゲーションの階層レベル数** 11

3.2 SCOの起動 **12**

3.2.1 findAPI 12

3.2.2 API**アダプタのロード手順** 14

3.3 API関数 **16**

3.3.1 API**関数の引数と返り値（１）** 16

3.3.2 API**関数の引数と返り値（２）** 16

3.3.3 API**関数の状態遷移** 17

3.3.4 API**関数によるデータの取得と書き込み** 18

3.4 データモデル **20**

3.4.1 **データ型** 20

3.4.2 session_time, total_time 20

3.4.3 lesson_status, score, mastery_score 21

3.4.4 **前提条件と**lesson_status 24

3.4.5 Mandatory**と**Optional 25

3.4.6 **リストデータ項目の添え字の順序** 26

3.4.7 lesson_status**の解釈** 27

3.5 その他 **28**

3.5.1 SCORM**準拠**LMS**の定義** 28

3.5.2 LMS**のコマンド**GUI 29

3.5.3 SCORM**非準拠コンテンツの移植** 30

1. はじめに

SCORM 規格が国内で実用的に使用されるようになって数年が経ちます．現在ではSCORM に準拠した LMS，コンテンツが数多く市場に流通しています．しかし，それらの製品間の相互運用性は必ずしも確立されているといえない状況です．この原因はさまざまですが，開発者の規格に関する理解が不足していたり，規格自体がわかりにくく結果として誤った実装をしているといった場合が多いようです．

本書は, 日本イーラーニングコンソシアムが 2003 年以降継続して行っている SCORM 規格の相互運用性に関する調査で得られた実際のトラブル事例に基づき，このようなトラブルが規格のどの部分に該当するのか，規格をどのように解釈すれば同じようなトラブルを起こさずに済むかを説明したものです．

実際に SCORM 規格の製品に開発に携わった経験のある方が本書を読まれることで，製品開発のための規格に関する知識をより深めていただき，相互運用性の向上に役立てていただければと思います．

本書は SCORM Ver.1.2 規格に対応した内容になっています．本書の構成は以下のようになっています．はじめに「2.事例」で調査で得られたトラブルの事例を示します．次に,「3.解説」で，対応する規格の解説，製品の実情，開発における推奨対応策，を述べます．

なお，本書中で以下の記号は，SCORM1.2 規格書の各分冊を示します．

OVW: SCORM Ver.1.2 Overview

CAM: SCORM Ver.1.2 Content Aggregation Model

RTE: SCORM Ver.1.2 Run-time Environment

CNF: SCORM Ver.1.2 Conformance Requirements

2. 事例

2.1 manifest ファイル

2.1.1 manifest ファイルの文字コードの相違

- manifest ファイルの文字コードが統一されていない．Shift-JIS のものと UTF-8 のものが存在する．【3.1.1 参照】
- LMS によっては Shift-JIS と UTF-8 のどちらか一方しか読み込めない．【3.1.1 参照】
- LMS によっては XML ファイルの BOM(Byte Order Mark)の有無によって読み込めない場合がある．【3.1.1 参照】

2.1.2 SCO の URL の大文字・小文字の区別

- 他サーバで動いていたコンテンツを別サーバに移植したところ, Web サーバがリソースの大文字・小文字を区別したため，404 エラーが発生した．【3.1.2 参照】

2.1.3 SCO の URL の絶対パス・相対パス

- manifest ファイルに書かれている SCO の URL がサイトのトップからの URL（“/”で始まる URL. 絶対パス）になっていた. LMS がこれを解釈できずに動作しなかった.【3.1.2 参照】

2.1.4 LMS の教材階層数の制限

- LMS で扱えるコンテンツアグリゲーションの階層レベル数が固定されていて，レベル数が合わないコンテンツを読み込めない LMS がある．【3.1.3 参照】

2.2 SCO の起動

2.2.1 FindAPI の検索順

- FindAPI の検索順が，SCO から親フレームに遡るのではなく，Top フレームから下に検索するようになっている SCO があった．このため API フレームが見つからずエラーとなった．【3.2.1 参照】

2.2.2 API アダプタのロード手順

- API アダプタのロードのタイミングが LMS によって異なり，特定の LMS で FindAPI がうまく動作しない【3.2.2 参照】

2.3 API 関数

2.3.1 LMSInitialize の引数がない

- 引数無しで LMSInitialize を呼び出す SCO があった．このため LMS がエラーを検出して動作しなかった．【3.3.1 参照】

2.3.2 LMSInitialize の返り値が Boolean

- LMSInitialize の返り値が Boolean となることを想定して作られた SCO があった．ある LMS に付属したオーサリングツールで作成された SCO で, このようになっている．【3.3.1 参照】

2.3.3 LMSInitialize と LMSFinish の繰り返し

- LMSFinish を呼ばずに何度も LMSInitialize を呼び出す SCO があった．LMS がこのような動作を想定していなかったのでエラーとなった．【3.3.3 参照】

2.4 データモデル

2.4.1 lesson_status のボキャブラリーが不正

- SCO から送られる lesson_status の値が “pass”, “fail” となっていて, LMS が正しく値を取得できなかった．【3.4.1 参照】

2.4.2 lesson_status の動作が実装されていない LMS

- lesson_status が SCO の書き込み値だけを反映するようになっていて, 初期化やその後の状態変化を正しく実装していない LMS があった．【3.4.3 参照】

2.4.3 lesson_status の値で学習時間を計時する LMS

- lesson_status に incomplete を設定してから completed を設定するまでの時間を学習時間として計時する LMS があった．このため学習時間がただしく測定できなかった．【3.4.2 参照】

2.4.4 前提条件と lesson_status の値

- LMS 側で前提条件を満足するために Passed を受け取る必要があったが，SCO が completed を出力した．【3.4.4 参照】

2.4.5 cmi.core.score.raw の値が範囲外

- SCO の出力する cmi.core.score.raw の値が範囲内（0〜100）に入っていない．【3.4.1 参照】

2.4.6 masteryscore が 0 の時の LMS の動作

- masteryscore が 0 の時の動作が LMS によってまちまちである．【3.4.3 参照】

2.4.7 masteryscore が空の時の LMS の動作

- masteryscore が空だと動作しない LMS がある．【3.4.3 参照】

2.4.8 実装していないオプションデータ要素の扱い

- オプション要素を実装していない LMS で，オプション要素を使用する SCO を動かしてエラーが出た．【3.4.5 参照】

2.4.9 リストデータ項目の添え字の順序

- cmi.interaction のようなリストデータ項目で，添え字の順番が乱れたときに LMS によって動作が異なる．【3.4.6 参照】

2.4.10 lesson_status の解釈

- lesson_status で完了と習得の概念が混在している．未完了だが習得，といった状況が表現できない【3.4.7 参照】

2.5 その他

2.5.1 SCORM 準拠 LMS なのに教材が移植できない

- SCORM 準拠 LMS に付属のオーサリングツールで教材を作成した．このデータを他の LMS に移植しようとしたところ，独自形式の教材で動作しなかった．【3.5.1 参照】

2.5.2 コマンド GUI が LMS によって異なる

- 前画面，次画面なのどのコマンド GUI が LMS によって異なる．【3.5.2 参照】

2.5.3 既存コンテンツの動作が移植によって再現できなくなる

- SCORM 非準拠コンテンツを SCORM 準拠 LMS に移植しようとすると再現できない機能が出てくる．【3.5.3 参照】

3. 解説

3.1 manifest ファイル

3.1.1 文字コード

► 規格

SCORM 規格自体には利用可能な文字コードの規定はありません．しかし，SCORM の manifest ファイルは XML 規格に準拠しています．XML 規格では，XML ファイルの先頭に，

```
<?xml version="1.0" encoding="Shift_JIS" ?>
```

のように記述することで，そのファイルで使用している文字コードを指定できることになっています．このように規格上は，XML ファイルで使用する文字コードは自由に指定できるのですから，本来 manifest ファイルにはどのような文字コードを使用しても良いはずです．

では，なぜLMSによってmanifestファイルに使用可能な文字コードが異なるのでしょうか? 実はXML規格によると，XMLファイルを読み込むプログラムは最低限UTF-8 とUTF-16 を扱えればよいことになっています（Extensible Markup Language（XML）1.0, http://www.w3.org/TR/2004/REC-xml-20040204）．つまり，日本国内でよく使われるShift JISやEUC-JPなどの文字コードは，プログラムによって読み込めたり読み込めなかったりするのです．従って，Shift JISで書かれたXMLファイルを読み込めないLMSでも規格違反ということにはなりません．

以上を整理すると表 3-1 のようになります．

表 3-1 manifest ファイル文字コードに関する規格上の規定

文字コード	manifest ファイル	LMS
UTF-8, UTF-16	使用可	サポート必須
その他（Shift JIS など）	使用可	サポート任意

また，XML の文字コード指定と実際に作成されたファイル自体の文字コードは別ですので注意が必要です．ファイル自体の文字コードはファイルを作成するプログラム（例えばテキストエディタ）の指定で決まります．例えば，Windows2000 以降の「メモ帳」では出力するファイルの文字コードを指定できます．従って，XML の文字コード指定とファイル自体の文字コードが食い違っていたら（ファイル自体は Shift-JIS なのに UTF-8 と指定した場合など）LMS は正しく動作しません．

またこれに関連して，ファイルの先頭にBOM(Byte Order Mark)と呼ばれる特別のデータを付加する場合があります．BOMはUTF-16 のような多バイトの文字コードを使用する

際，下位バイトを先に並べるか，上位バイトを先に並べるかを指定するために使用されます．BOMの詳細についてはRFC 2781 “UTF-16, an encoding of ISO 10646” (http://www.ietf.org/rfc/rfc2781.txt)などを参照してください．BOMは0xFEFFという2バイトのデータです．RFC 2781では，

- ファイルの先頭の2バイトが0xFE, 0xFFなら，ファイルはビッグエンディアン
- ファイルの先頭の2バイトが0xFF, 0xFEなら，ファイルはリトルエンディアン
- ファイルの先頭の2バイトがどちらでもなければ，ファイルはビッグエンディアン

と解釈しなくてはならない，としています．

なお，BOMはエンディアンの判別だけでなく，文書がUnicodeで記述されているかどうかを判別するために用いられることもあり，UTF-8などでも先頭にBOMがついている場合があります。

► 実態

規格上，manifestファイルでは任意の文字コードを使用することができますが，LMSとしてかならず対応しないといけない文字コードはUTF-8とUTF-16だけです．従って，Shift JISなどで書かれたmanifestファイルは，LMSによって文字化けしたり，全く読み込めないトラブルが発生する可能性があります．

では実際に国内で使われている文字コードはどうなっているでしょうか．国内ではコンテンツ作成環境もLMSもWindowsプラットフォーム上で稼動するものが多いせいか，Shift JISを用いたコンテンツがかなり出回っており，LMSもShift JISのmanifestファイルを読み込める場合が多いようです．なお，まれに，Shift JISで書かれたXMLファイルだけしか読み込めないLMSが存在しますが，これはXML規格に違反しています．

► コンテンツ開発者へのお勧め

以上から，規格上保証されていてmanifestファイルで安全に使用できる文字コードはUTF-8とUTF-16ということになります．従って，コンテンツを作成する際に使用する文字コードとしては，UTF-8かUTF-16を選択するのがお勧めです．もし，念を入れるのであれば，同じコンテンツで，UTF-8かUTF-16のmanifestファイルと，Shift JISのmanifestファイルを両方用意しておけば，規格に違反したLMSにも対応できるでしょう．

► LMS開発者へのお勧め

まず規格に忠実にUTF-8とUTF-16のmanifestファイルを扱えるように設計しましょう．また，国内の実情からShift JISのmanifestファイルも読み込めるようにしておくのが良いでしょう．なお，一般にXMLファイルを扱うために用いられているXMLパーサ（DOM，SAXなど）と呼ばれるモジュールは，これらの文字コードを扱えるように作られている場合がほとんどです．これらのモジュールをうまく活用してシステムを設計しましょう．BOMの有無に関してもトラブルが起きる場合があるようですので，念のためテストしておきましょう．

3.1.2 リソースの URL

► 規格

SCORM コンテンツで用いられる SCO やアセットは一般的な WWW コンテンツで，manifest ファイル中の URL(Uniforfm Resource Locator）によって指定されます．URL はインターネット上のリソースを指定するために用いられる標準規格で，WWW で用いられるいわゆるホームページの指定（http://www.～）は URL の例です．

URL の標準規格は IETF（Internet Engineering Task Force）という団体の出している RFC (Requiest For Comment）という文書で規定されています．URL の記述法については RFC2396 “Uniform Resource Identifiers (URI): Generic Syntax” (http://www.ietf.org/rfc/rfc2396.txt）で規定されています．URL の表記は概ね，表 3-2 のように分類されます（実際にはもっと詳しい規定があります)．

まず，URL における大文字・小文字の区別についてです．スキーマとホストの部分は区別しません．つまり，HTTP://www.ELC.or.jp と書かれても http://www.elc.or.jp として扱かわれます．パスの部分は，大文字・小文字が区別されます．つまり，http://www.elc.or.jp/index.html と http://www.elc.or.jp/INDEX.HTML は別のリソースの指定として扱われます．

次に相対 URL の補完についてです．相対 URL は実行時に，実際のリソースを指すために，絶対 URL に変換されなくてはなりません．この変換を行うためにベース URL という概念が用いられます．ベース URL は相対 URL を補完して絶対 URL に変換するために必要な情報を補う元となるものです．

表 3-2 URL の分類

分類	一般形	例
絶対 URL	スキーマ ： パス	http://www.elc.or.jp/index.html http がスキーマ．//www.elc.or.jp/index.html がパス．
相対 URL		
ネットパス	// ホスト [絶対パス]	//www.elc.or.jp/index.html www.elc.or.jp がホスト．
絶対パス	/ パス記述	/course/index.html
相対パス	パス記述	course/index.html

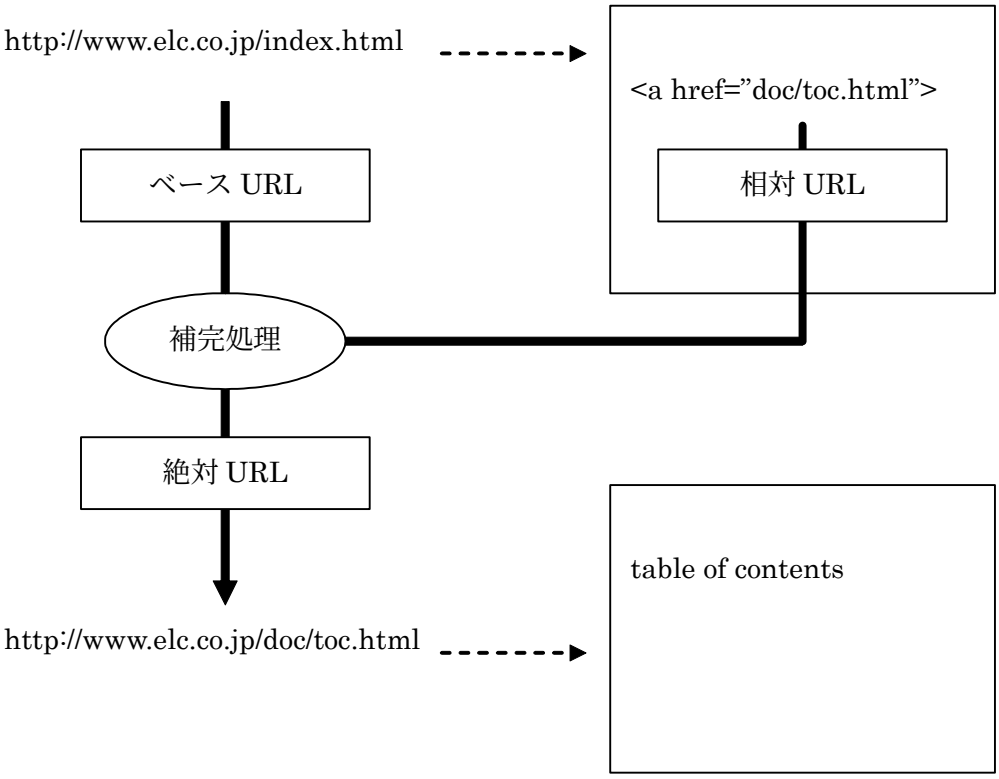


図 3-1 ベース URL による相対 URL の補完

この関係の一般的な例を図 3-1 に示します．http://www.elc.or.jp/index.html という HTML ファイルがブラウザに表示されているとします．この HTML ファイルの中に doc/toc.html という相対 URL のハイパーリンクが書かれているとします．このハイパーリンクをクリックすると http://www.elc.or.jp/doc/toc.html という別の HTML がブラウザに表示される，というのは，HTML を作成したことがある人なら誰でもご存知だと思います．

この時の動作をもう少し厳密に書くとブラウザは以下の処理を行っています．

1) 相対 URL doc/toc.html がクリックされたことを検知する．
2) ベース URL として，現在表示されている HTML ファイルの URL である http://www.elc.or.jp/index.html を使う．
3) 相対 URL doc/toc.html をベース URL http://www.elc.or.jp/index.html で補完して，絶対 URL http://www.elc.or.jp/doc/toc.html を求める．
4) 絶対 URL http://www.elc.or.jp/doc/toc.html を読み込む．

また，補完の手順は相対 URL の種類によって以下のようになります．

- 相対 URL がネットパス, つまり, // で始まる場合, ベース URL のスキーマに相対 URL を連結する．
- 相対 URL が絶対パス，つまり，/ で始まる場合，ベース URL のスキーマとホストに相対 URL を連結する．

- 相対 URL が相対パスの場合，ベース URL の最後の / より後ろの部分を取り除き，残った部分に相対 URL を連結する．

例を　　表 3-3 に示します．相対 URL とベース URL の網掛けの部分が，絶対 URL を生成するのに使われます．

表 3-3　相対 URL の補完方法

ネットパス	ベース URL	http://www.elc.or.jp/doc/toc.html
	相対 URL	//www.elc.org/index.html
		↓　　↓
	絶対 URL	http://www.elc.org/index.html
絶対パス	ベース URL	http://www.elc.or.jp/doc/toc.html
	相対 URL	/man/index.html
		↓　　↓
	絶対 URL	http://www.elc.or.jp/man/index.html
相対パス	ベース URL	http://www.elc.or.jp/doc/toc.html
	相対 URL	man/index.html
		↓　　↓
	絶対 URL	http://www.elc.or.jp/doc/man/index.html

図 3-１の例では，相対 URL が含まれているファイルの絶対 URL がベース URL として用いられました．ベース URL の指定方法は他にもあり，優先度の高い順に以下のようになります．

i)　相対 URL が含まれているファイルの中で指定する．

ii) 相対 URL が含まれているファイルの置かれた環境（例えばアーカイブファイル）の中で指定する．

iii) 相対 URL が含まれているファイルの絶対 URL を使用する（上記の例）．

iv) 相対 URL を使用するアプリケーションでデフォルトのベース URL を指定する．

では, 次に SCORM の manifest ファイルにおける URL の扱いについて見てみましょう．SCORM の manifest ファイルは IMS のコンテンツパッケージング規格（IMS Content Packaging，http://www.imsproject.org/content/packaging/index.cfm）に準拠したものです．コンテンツパッケージング規格では以下が定められています．

- manifest ファイル中の相対 URL はコンテンツパッケージのルートディレクトリ（manifest ファイル自体が置かれたディレクトリ）からの相対 URL であり，ルートディレクトリの絶対 URL をベース URL として補完する（実際には manifest ファイル内でベース URL を指定する方法，つまり，上記の i)　に相当するオプションもありますがここでは省略します．）．

つまり，LMS は manifest ファイルを読み込んで教材を配信する際に，manifest ファイルに書かれた SCO やアセットの URL が相対 URL だったら，この相対 URL をルートディレクトリの絶対 URL をベース URL として補完し，これらの SCO やアセットがブラウザに表示されるようにしなくてはなりません．これを図 3-2 に示します．

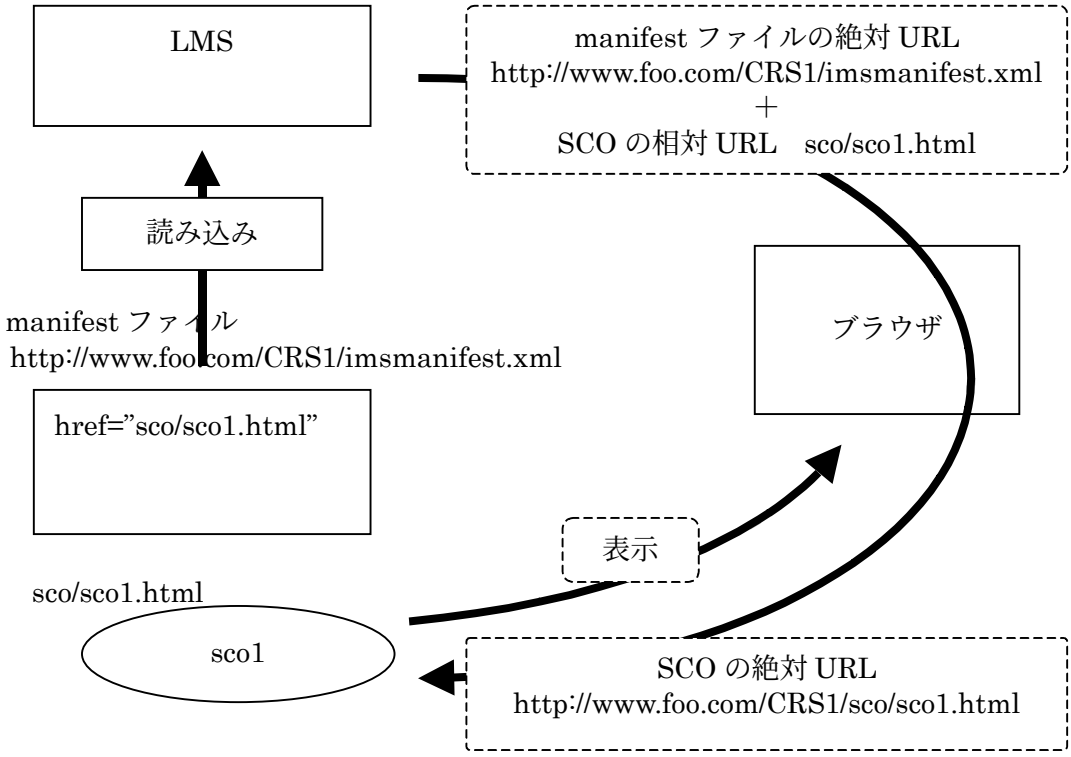


図 3-2　SCO の相対 URL の補完

► 実態

上で説明したようにURLのパスの部分は大文字･小文字が区別して扱われます.従って，manifest ファイル内の URL 表記と Web サーバに実際にアクセスする際の URL は大文字･小文字の違いも含めて正確に一致していなくてはなりません．問題は，Web サーバに Windows 系の OS を用いている場合の動作です．Windows 系の OS ではファイル名の大文字･小文字は区別されず, URL が大文字でも小文字でも同じファイルがアクセスされます．このため，Windows 系 OS の LMS では，manifest ファイル内の URL と Web サーバの実際の URL の大文字・小文字が合致していなくても，SCORM 教材として正しく動作しているように見えてしまう可能性があります．そしてこのようなコンテンツを Unix 系 OS の LMS に持ってくると，Unix ではファイル名の大文字・小文字が区別されるため，404 エラー（File not Found）が発生するのです．

また，manifest ファイル中の URL は，規格上，絶対 URL，相対 URL（ネットパス，絶対パス，相対パス）のいずれも使うことができます．しかし，教材の移植性を考えると，

相対URLのうちのネットパスと絶対パスは使用を避けるべきです. なぜなら移植先のLMSによって，manifest ファイルがどのディレクトリに格納されるかは決まりがなく，ルートディレクトリからの位置がLMSによって変わってしまうからです．しかし，ごくまれに特定の LMS で固有のリソースにアクセスするために，manifest ファイル中に絶対パスを記述している例があり，このような教材は他の LMS に移植した際，エラーを起こす可能性があります.

► コンテンツ開発者へのお勧め

まず，manifest ファイル中の URL で大文字・小文字を区別することは移植性の高いコンテンツを作るために必須の条件です.

次に URL の指定では，ネットパスと絶対パスは使用を避けるべきです．どうしても必要な場合，例えば，course1 という教材から course2 という教材のリソースを参照したい，というような場合は，../course2/scoA.html のような形の相対 URL を用いることも可能ですが，一般には使用を避けるべきでしょう.

► LMS 開発者へのお勧め

URL の大文字･小文字については manifest ファイル中の URL をそのまま使うべきです.

SCO の絶対 URL の生成には表 3-3 に示した処理が必要になります. 相対 URL とベース URL から絶対 URL を生成するには細かくはいろいろな場合を想定する必要がありますが, この手順は RFC2396 に記載されています．また，上で説明したように，ブラウザはもともとこのような機能を有していますので, SCO をブラウザに読み込ませるための HTML 記述を工夫して，ブラウザでこの処理を行うことも可能です.

3.1.3 コンテンツアグリゲーションの階層レベル数

► 規格

コンテンツアグリゲーションの階層レベル数に関して，SCORM の規格書ではあまり明確に述べられていません.「階層レベル数に制限は無い」というのが正解なのですが，陽にこのように書かれている箇所は無いようです．しかし，素直に規格書を読めば「階層レベル数に制限は無い」と解釈できますし，注意深く読むとコンテンツアグリゲーションの構成要素である item 要素に関して以下のような定義がなされています.

「<item>要素は他の<item>要素中で任意のレベルの深さにネストできる．（CAM 2.3.5.3.1.2.2. <item>要素）」

さらに, 標準準拠レベルの規定では, 以下が準拠LMSに対する要求条件になっています.

「LMSはSCORMコンテンツアグリゲーション規格 2.3 で定義された既知の準拠コンテンツアグリゲーションコンテンツパッケージをインポートし処理できる（CNF LMS-RTE1）」

► **実態**

一部の市販LMSの中には，扱える階層レベル数が固定されていたり，レベル数が制限されているものがあります．また，任意レベル数のコンテンツを読み込んでも，それを内部形式にマッピングする際に，階層構造を勝手に変えてしまう製品もあります．確かにこれでもインポートしたことになりますが，規格では「インポートし処理できる」となっているので厳密には規格準拠とはいえません．

規格書には上記のように任意のレベル数を扱えなくてはならないことが明記されているのですが，ADL の Test Suite に扱える階層レベル数のテストが含まれていないことも混乱の一因と考えられます．

► **コンテンツ開発者へのお勧め**

多くの LMS で稼動するコンテンツを作成するために，一部 LMS の制約に合わせたコンテンツを作成する，という考え方もあるかと思います．しかし，コンテンツの階層構成はインストラクショナルデザインなどの観点からは非常に本質的な問題ですし，このような LMS は遅かれ早かれ淘汰されていくと考えられますので，制約を設けずに利用者サイドの要求条件にあったコンテンツ構成を取るのが正しい選択と思われます．

► **LMS 開発者へのお勧め**

規格に準拠して任意階層のコンテンツを読み込み，コンテンツ構造を崩さずに実行可能な実装をするべきです．もっとも問題となるのは，教材やログの格納にリレーショナルデータベースを用いている場合の任意レベル階層の扱いと考えられます．リレーショナルデータベースを用いて任意レベルの階層構造を扱おうとすると性能上のネックになる場合があります．これを避けるためには，実用的に問題の無い上限（例えば 10 階層程度）を設けた実装として，実効的に制限を無いように見せるという選択肢もあります．また，最近では XML データベースやオブジェクト指向データベースが比較的容易に利用できるようになってきましたので，このような製品を選択するのもひとつの選択肢です．

3.2 SCO の起動

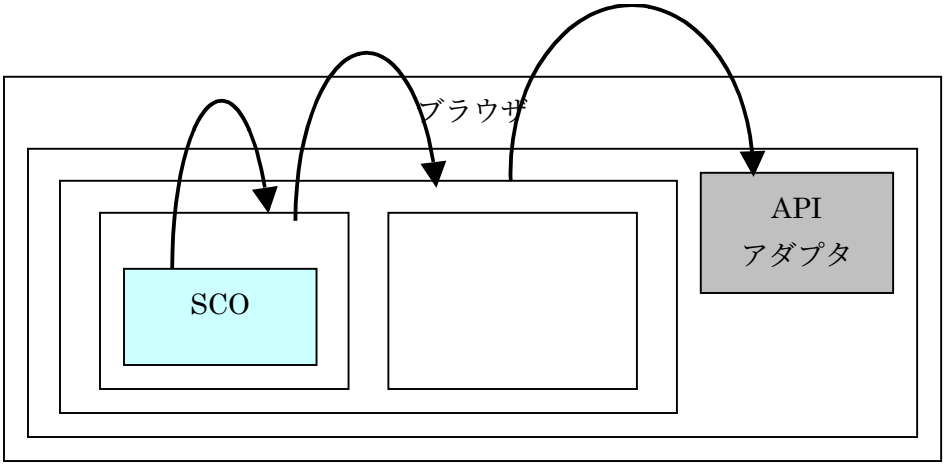
3.2.1 findAPI

► **規格**

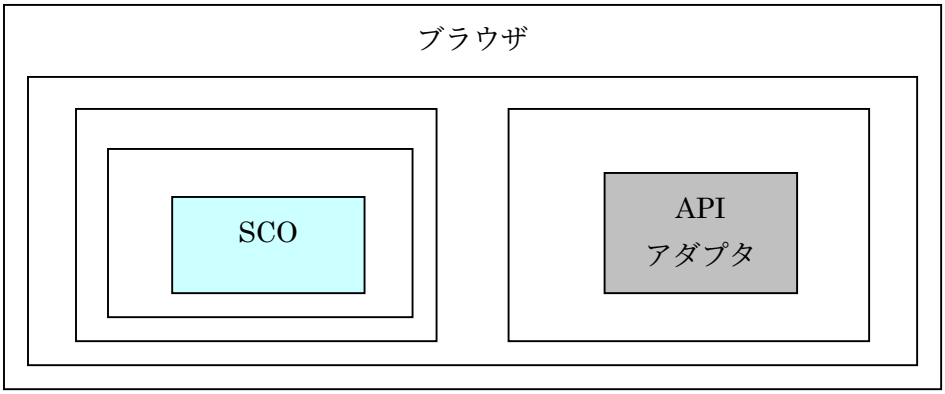
SCO 起動時，SCO はブラウザ中の API アダプタを見つける必要があります．このとき SCO は，ブラウザのウインドウを，自身のウインドウから親ウインドウ／オープナウインドウを順にたどって API アダプタオブジェクトを見つけます. このための JavaScript プログラムの例(findAPI)が SCORM1.2 の規格書に掲載されています（RTE 3.3.6.1 FindAPI).

また，LMS は，SCO を表示するウインドウの祖先ないしオープナウインドウに API アダプタを配置するようにして, 上記の手順で SCO が API アダプタを見つけられるようにし

ます.



a) SCO から検索可能



b) SCO から検索不可能

図 3-3 findAPI

図 3-3 にこれを示します. SCO は a)に示すように, 自身の親ウインドウを順にたどり, ウインドウ中に “API” という名前のオブジェクトがあれば, それを API アダプタと特定します.

LMS は SCO からみて祖先になるウインドウに API アダプタを “API” というオブジェクト名で配置しなくてはなりません. b) のように, 祖先でないウインドウに配置したり, “API” という名前以外のオブジェクト名を使用すると SCO が API アダプタを見つけられなくなってしまいます.

► 実態

この処理で問題が生じている例はあまりないようです. 一部の SCO で上記の規格に反した検索方法（トップウインドウから検索する，など）を用いていて，LMS によってはうま

く動作しないという例が報告されています.

► **コンテンツ開発者へのお勧め**

SCORM 規格書に記載の findAPI あるいはこれと等価の手順で API アダプタを検索するようにすべきです.

► **LMS 開発者へのお勧め**

LMS は SCO からみて祖先になるウインドウに,“API”というオブジェクト名で API アダプタを配置しなくてはなりません.

3.2.2 API アダプタのロード手順

► **規格**

SCO 起動時, SCO がブラウザ中の API アダプタを findAPI によって見つけられるよう, LMS は SCO を表示するウインドウの祖先ないしオープナウインドウに API アダプタを配置しなくてはなりません. このとき, API アダプタと SCO がブラウザにロードされるタイミングに注意する必要があります. もし, 図 3-4 で SCO が API アダプタよりも前にロードされて findAPI を実行すると, findAPI は失敗しエラーが起きてしまいます.

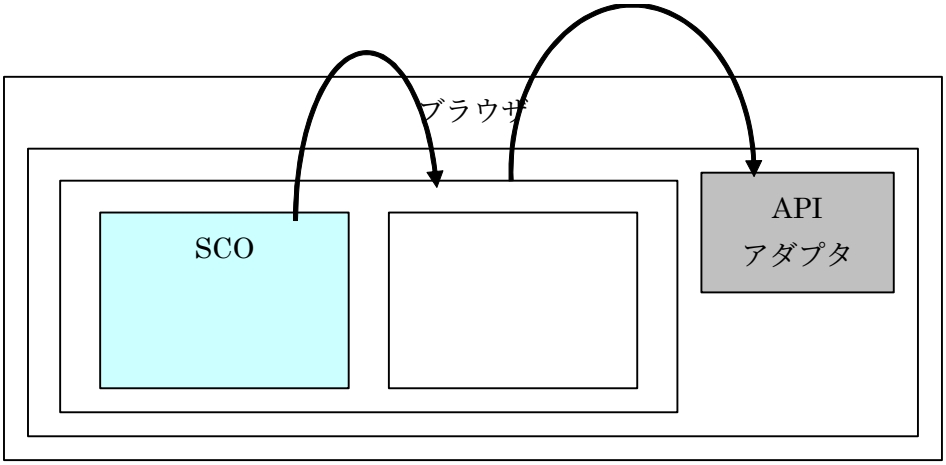


図 3-4 API アダプタのロードタイミング

► **実態**

API アダプタのロードに関する実装については規格書では明確に述べられていません. しかし, ロードのタイミングが不安定だと, ある LMS では動いていたコンテンツが別の LMS では動かなくなるという問題が生じる可能性があります. しかもこの問題はブラウザがフレームをロードする順番にも依存するため再現性に乏しく, 原因の解決に時間が掛かる場合があります.

► **コンテンツ開発者へのお勧め**

SCO 側では findAPI に従って API アダプタを探す以上に, 対処する方法はありません.

ある LMS では findAPI で問題が起きなかったのに，他の LMS で findAPI が失敗するという現象が出たら，以下に述べる対応がなされているかどうか LMS ベンダと相談しましょう．SCO の中で findAPI のタイミングを遅らせるといった対処は，問題を余計にわかりづらくする可能性がありますので避けるべきです．

► LMS 開発者へのお勧め

LMS は SCO がロードされる前に，確実に API アダプタがロードされていることを保証しなくてはなりません．

LMS が以下のフレームの定義を生成したとします．この中では，API アダプタを SCO の親のフレームに配置していますので，API アダプタが先にロードされれば findAPI は成功します．しかし，どのフレームを先にロードするかはブラウザの動作に依存しますので，もし SCO が先にロードされれば findAPI は失敗してしまいます．

```
<HTML> <HEAD> <TITLE>LMS system</TITLE>
      <FRAMESET ROWS="100%,0%">
            <FRAMESET ROWS="43,*">
                  <FRAME NAME="Menu" SRC="MENU GENERATION URL">
                  <!-- SCO フレーム-->
                  <FRAME NAME="Main" SRC="SCO URL">
            </FRAMESET>
            <!-- API フレーム-->
            <FRAME NAME="API" SRC="API ADAPTER URL API.html">
      </FRAMESET>
</HTML>
```

これを避けるためには下に示すように API アダプタがロードされてから SCO のロードが始まるようにします．

```
<HTML> <HEAD> <TITLE>LMS system</TITLE>
      <SCRIPT LANGUAGE="JavaScript">
            function myOnLoad( url )
            { self.Main.location = url; }
      </SCRIPT></HEAD>

      <!-- この FARMESET がロードされると myOnLoad が実行される -->
      <FRAMESET ROWS="100%,0%" onLoad = myOnLoad("SCO URL")>
            <FRAMESET ROWS="43,*">
                  <FRAME NAME="Menu" SRC="MENU GENERATION URL">
                  <!-- SCO 用ブランクフレーム -->
                  <FRAME NAME="Main" SRC="about:blank">
            </FRAMESET>
            <!-- API フレーム-->
            <FRAME NAME="API" SRC="API ADAPTER URL API.html">
      </FRAMESET>
</HTML>
```

このHTMLファイルでは，SCOをロードするフレームは最初ブランクにしておきます．そしてAPIフレームを含むフレームセットのロードが終了したら，myOnLoadという

Javascript関数を実行してSCOをロードするようにしています．このあとロードされたSCOがfindAPIを実行しますので，確実にAPIアダプタを見つけることができます．

3.3 API 関数

3.3.1 API 関数の引数と返り値（1）

► 規格

API 関数の引数と返り値はすべて文字列です．意味的に，論理型，数値型の引数や返り値を扱う場合も，これらのデータを文字列で表わしてやりとりします．また，LMSGetLastError() を除いてすべての API 関数は引数を持ちます．LMSInitialize, LMSFinish, LMSCommit では空文字列(“”)の引数を指定しなくてはなりません（RTE 3.3.2.1 SCO to LMS Communications API Details）.

► 実態

SCORM1.0 では，引数と返り値の型に混乱が見られます．例えば LMSInitialize の引数は NULL，返り値は Boolean（論理型）といった記述が見られます．SCORM1.0 の時代に作られたコンテンツやオーサリングツールで，このような引数と返り値の型の混乱があるようです．

► コンテンツ開発者へのお勧め

引数の使用には十分注意すべきです．特に，LMSInitialize, LMSFinish, LMSCommit を引数無しで使用すると LMS がエラーを起こす可能性があり，移植時に問題を起こすことがあります．

► LMS 開発者へのお勧め

返り値は規格に沿って文字列を使って実装しなくてはなりません．また，API 関数が引数無しで使用されたり，誤った引数の個数で使用される場合も考慮して，このような場合でもシステムが落ちないようにしておくことが望まれます．

3.3.2 API 関数の引数と返り値（2）

► 規格

LMSInitialize, LMSFinish, LMSCommit, LMSSetValue では返り値は，文字列 “true”, “false” です．これによって関数が正しく実行されたか否かを判断できます．

LMSGetValue では LMS からのデータが返り値となりますので，関数が正しく実行されたか否かを判断するためには, LMSGetValue 直後の LMSGetLastError の値が “0” かどうかを確認します．

► **実態**

SCO を作成する際に API 関数が正しく実行されたかどうかを確認していない場合が多いようです．もちろん LMS が期待通りに動作していれば問題は無いのですが，そうでなかった場合，エラーの確認をしていないと，問題が起きていること自体がわからなかったり，問題が起きている箇所の特定に手間取ることになります．

► **コンテンツ開発者へのお勧め**

API 関数を使用している箇所では，かならず上に書いた方法で，関数が正しく実行されたか否かを確認し，エラーが起きている場合はその旨エラーコードを含めて表示するようにします．これによって，問題の有無，問題箇所，原因の特定が早くなります．

► **LMS 開発者へのお勧め**

SCO によるエラーチェックが正しく行われるよう, LMS 側でもエラーチェックを正しく実装しましょう. 特に, エラーの分類とエラーコードの対応を正確に行う必要があります. これが正しくないと，教材作成者はいたずらに混乱することになります．

3.3.3 API 関数の状態遷移

► **規格**

API アダプタは，SCO の起動から終了まで三つの状態を遷移するものとして，その動作が規定されています. また, それぞれの状態で SCO が使用できる関数が決められています. これを表 3-4 に示します(RTE 3.3.2.2 SCO to LMS Communications API Adapter State Transition).

表 3-4 API アダプタの状態遷移と利用可能な API 関数

関数＼状態	not initialized	initialized	finished
LMSInitialize	○→	×	×
LMSFinish	×	○→	×
LMSGetValue	×	○	×
LMSSetValue	×	○	×
LMSCommit	×	○	×
LMSGetLastError	○	○	○
LMSGetErrorString	○	○	○
LMSDiagnostic	○	○	○

○：発行可，○→：発行可 & 次状態へ遷移，×：発行不可

三つの状態は，“not initialized”, initialized”, “finished” です．API アダプタはこれらの状態をこの順に遷移し，戻ったりループすることはありません．

SCO 起動直後の状態が “not initialized” です．この状態で SCO が発行可能な API 関数は表 3-4 の通りです．例えばこの状態で LMSGetValue を発行することはできません．

“not initialized” 状態で SCO が LMSInitialize を発行すると API アダプタの状態は

“initialized” に遷移します．これが実際に SCO と LMS が通信を行う状態です．SCO は表 3-4 のように LMSInitialize を除くすべての関数を使うことができます．

“initialized” 状態で SCO が LMSFinish を発行すると API アダプタの状態は “finished” に遷移します．この状態では API アダプタは動作を終了しており，SCO はエラー処理関連以外の関数を発行できません．

► 実態

上記の状態遷移を遵守せずに, 例えば LMSInitialize を繰り返し呼び出すように作られている SCO が存在します．SCORM1.2 では，このような違法な呼び出しが行われた場合の API アダプタ側（LMS 側）の動作は明確に規定されておらず，たまたま動作する LMS も存在するようです．

► コンテンツ開発者へのお勧め

SCO を作成する際には，上記の状態遷移を明確に意識する必要があります．これに反する関数呼び出しを行うと LMS によって動作しなくなる危険があります．また, LMSFinish を確実に発行するためには，HTML の “onbeforeunload” （Internet Explorer の場合），“onunload” （Internet Explorer 以外のブラウザの場合）イベントを用い，SCO が切り替わるタイミングで LMSFinish を発行することが推奨されます．

► LMS 開発者へのお勧め

上記の状態遷移を意識して API アダプタを実装します．また，SCO が規格に反した API 呼び出しを行う場合を想定し，そのような呼び出しが行われてもシステムが落ちないようにしておくことが望まれます．SCORM2004(1.3)では，違法な呼び出しが行われた場合の API アダプタの動作が明確化されていますので，それを参考にするのも良いでしょう．

3.3.4 API 関数によるデータの取得と書き込み

► 規格

API 関数でデータの書き込みと取得に関係する関数は，LMSSetData, LMSGetData, LMSCommit です．また，LMSFinish はその内部で LMSCommit と同じくデータの LMS 側での保存を行います．

SCO が LMSSetValue で読み書き可能なデータモデル要素（例えば cmi.core.score.raw）にデータを書き込んだ場合，それ以降の LMSGetValue では書き込んだ値が読み出せなくてはなりません．これは，SCO の起動から終了までのセッションの中でも，複数のセッションをまたがる場合でも同様です．従って，LMS（API アダプタ）はセッション内でもセッション間でも上記の動作を行う必要があります．

一方，サーバとクライアントの通信の効率を考え，SCO が LMSSetValue した値を一回のセッション中のどのタイミングで, 実際に API アダプタ（クライアント側）から LMS（サーバ側）に転送するかは，LMS の実装に任されています．LMS に要求されるのは，

LMSSetValue 後に LMSCommit および LMSFinish が発行された場合は，LMS（サーバ側）でデータを確実に保存するということです．また，SCO に要求されるのは，データを保存するために LMSCommit ないし LMSFinish を必ず実行することです．SCO は終了前に LMSFinish を必ず実行することになっていますから, LMSCommit は実行してもしなくても構いません（RTE 3.3.2.1 SCO to LMS Communications API Details).

► 実態

上記の規格上の要求により, クライアントとサーバのデータ転送に関する LMS 実装のアプローチとしては大きく以下の二つが考えられます.

1) LMSSetValue で毎回クライアントからサーバにデータを書き込み，LMSGetValue で毎回サーバからクライアントにデータを読み出す. LMSCommit および LMSFinish ではデータの転送は行わない．プログラムの作りはシンプルになるが, SCO が頻繁にデータの読み書きを行った場合，通信回数が増えて性能が落ちる可能性がある．
2) API アダプタにキャッシュを設け，LMSSetValue でキャッシュに書き込み，LMSGetValue でキャッシュから読み出す．LMSCommit および LMSFinish で API アダプタから LMS にデータを転送する．プログラムの作りは複雑になるが，性能面では有利になる．

一部の LMS で，LMSFinish だけでなく LMSCommit を合わせて実行しないとデータが LMS に保存されないものがありますが，これは実装の誤りです．SCO としては，データを保存するために必ず実行しなくてはならないのは LMSFinish だけです．

► コンテンツ開発者へのお勧め

上記のように SCO は処理を終わる前に LMSFinish を発行することが唯一の要求条件です. しかし, データが実際に LMS 側に保存されるタイミングは LMS によって異なります. 演習問題の SCO などで問題数が多いもの, 解答に時間を要するものでは, 適宜 LMSCommit を発行し，万が一，ブラウザが異常終了したり，回線が切れたりしてもできるだけデータがサーバ側で保存されるようにすべきです．ただし，あまり頻繁に LMSCommit を発行すると回線に負荷をかけることになりますので，それなりの注意は必要です．

► LMS 開発者へのお勧め

LMS の実装は，上記のように二つのアプローチが考えられ，どちらがよいかは一概には言えません．強いていえば LMS を利用する環境によって，アプローチを切り替えられるよう，すなわち LAN 環境では 1)を，インターネット環境では 2)を用いるように切り替えができると良いでしょう．

3.4 データモデル

3.4.1 データ型

► 規格

SCORM のランタイム環境でやりとりされるデータ型には，論理型，整数型，実数型，文字列型などがあります. いずれも API 関数でやり取りする際には JavaScript の文字列で表現されます．それぞれの型は，データ長や値の範囲に関する制限を持っています.

また，使用するデータ項目によってさらに制限が加わる場合があります．例えば cmi.core.score.raw は実数型ですが，値の範囲は 0 から 100 の間でなくてはなりません.

特殊な型として語彙型があります．これは，特定のデータ項目に対して制限された語彙を提供するものです．例えば，データ項目 cmi.core.lesson_status に対する語彙は CMIVocabulary(Status) という型で，その値は，“passed”, “failed”, “completed”, “incomplete”, “browsed”, “not attempted” の六つです．語彙型では大文字・小文字を区別します (RTE 3.4.5 Data Types and Controlled Vocabulary).

► 実態

値の範囲や，語彙の綴りの誤り，大文字・小文字の混在，省略，などでトラブルが発生することがあります.

► コンテンツ開発者へのお勧め

値の範囲や語彙についてよくチェックし，事前にテストベッドなどを用いた試験を実施しましょう．得点(cmi.core.score.raw) については，設計の段階で，値が範囲内に入るように配点を決定することが必要です．語彙については，作成時にスペルミスをチェックし，テスト時には設計した語彙すべてをテストしたことを確認しましょう.

► LMS 開発者へのお勧め

各データ項目についてデータ型の範囲から外れたデータを SCO が送信してくることを想定してシステムを実装することが必要です．データ型の範囲外の値に対して適切にエラー処理を行い，システムがハングしたりしないようにしましょう.

3.4.2 session_time, total_time

► 規格

SCORM のランタイム環境で学習時間に関するデータ項目として，session_time と total_time があります．前者は SCO から LMS に送信され，SCO の一回の起動から終了までの時間を記録したものです．後者は LMS から SCO に送信され，SCO のそれまでの実行時間の合計を表わします.

SCO は一回のセッション（起動から終了まで）の間に何回でも session_time を送信できます．送信される値は，SCO の起動から送信時点までの経過時間です．従って，LMS はセ

ッション中に複数回送られてくる session_time のうち最新のものだけを残し，セッション終了時（LMSFinish 発行時）に最新の session_time を total_time に足しこみます(RTE3.4.4 The SCORM Run-time Environment Data Model).

► 実態

SCO 側で session_time を実装するか否かは任意です. 従って, 実際に出回っている SCO で session_time を実装しているものはあまり多くないのではないかと想定されます．このため，SCO 毎の学習時間を LMS 側で把握するためには，SCO を起動してから次の SCO を起動するまでの時間を計測するのが最も確実で，実際にこのような実装になっている LMS が多いようです.

また，これ以外の手段で学習時間を計測する LMS，例えば，LMSInitialize から LMSFinish までの時間を計測する LMS もありますが，解説型の SCO などでは，LMSInitialize の直後に LMSFinish を発行するものもあり，実際の学習時間を確実に把握できる保証はありません.

► コンテンツ開発者へのお勧め

SCO 側で学習時間によって動作を変えたい場合や，合計の学習時間を制限したい場合は session_time を実装し，LMS から total_time を取得することが必要になります．session_time の実装は任意ですが，処理は難しくないので LMSFinish の直前に session_time を送信するようにしておくのがお勧めです．送信するタイミングは onbeforeunload ないし onunload のイベントを使うことができます（3.3.3 参照）.

► LMS 開発者へのお勧め

LMS 側では session_time と total_time に関して特別な注意事項はありません．規格のとおりに実装すればよいでしょう．ただし，上記のように SCO が session_time を送信するか否かは任意ですので，学習時間の計測には別の方法を併用する配慮が必要です.

3.4.3 lesson_status, score, mastery_score

► 規格

SCORMの中でももっともトラブルを起こしやすいのが，lesson_status，score, mastery_scoreの関係です．この関係は表にして整理してしまえばわかりやすいのですが，規格書では文章で書かれており，しかも，原文では “may”, “should”[1] などの助動詞が一般の英文と異なる技術規格文書独特の意味で使われているため，ますますわかりにくくなっています（RTE3.4.4 The SCORM Run-time Environment Data Model, CNF LMS Run-Time Environment Data Model Conformance Requirements 1.6.6).

[1] “may” は一般には「~かもしれない」だか，規格文書では「~してもよい」,「~する場合がある」の意味，“should” は一般には「~だろう」だか，規格文書では「~しなくてはならない」の意味で使います.

ではこの関係を表 3-5 で見ていきましょう.

表 3-5　lesson_status, score, mastery_score の関係

項番	manifest→LMS	SCO→LMS		LMS→SCO
	Mastery_score	score.raw	lesson_status	lesson_status
1	なし	なし	なし	変化無し*)
2	なし	なし	規定語彙	SCO 設定値
3	なし	0.5	なし	変化無し*)
4	なし	0.5	規定語彙	SCO 設定値
5	0.8	なし	なし	変化無し*)
6	0.8	なし	規定語彙	SCO 設定値
7	0.8	0.5	なし	failed
8	0.8	0.5	規定語彙(incomplete 以外)	failed
9	0.8	0.5	incomplete	incomplete
10	0.8	0.8	なし	passed
11	0.8	0.8	規定語彙(incomplete 以外)	passed
12	0.8	0.8	incomplete	incomplete
13	0.8	0.9	なし	passed
14	0.8	0.9	規定語彙(incomplete 以外)	passed
15	0.8	0.9	incomplete	incomplete

*) 初期状態は Not Attempted に LMS が設定，SCO 表示後は completed に LMS が設定

- manifest ファイル中の mastery_score，ないし，SCO からの cmi.core.socre.raw が設定されていない場合（表の項番 1-6）：
  - SCO から cmi.core.lesson_status が設定されなければ，LMS 側の cmi.core.lesson_status の値は変化しない. 但し, 初期状態（cmi.core.lesson_status が not attempted）で，SCO を表示したあとは LMS は値を completed に書き換える（表の項番 1, 3, 5）.
  - SCO から cmi.core.lesson_status が規定された語彙（completed, incomplete, passed, failed）のいずれかに設定されれば，それを LMS 側の cmi.core.lesson_status の値とする（表の項番 2, 4, 6）.
- manifest ファイル中の mastery_score，および，SCO からの cmi.core.socre.raw がどちらも設定されている場合（表の項番 7-15）：
  - SCO からの cmi.core.lesson_status が incomplete 以外の場合は，mastery_score と cmi.core.socre.raw を比較して LMS 側の cmi.core.lesson_status の値を決定する：
    - ✧ cmi.core.socre.raw ＜ mastery_score なら，LMS 側の cmi.core.lesson_status を failed とする（表の項番 7, 8）.
    - ✧ cmi.core.socre.raw ≧ mastery_score なら，LMS 側の cmi.core.lesson_status を passed とする（表の項番 10, 11, 13, 14）.

- SCO からの cmi.core.lesson_status が incomplete の場合は，LMS 側の cmi.core.lesson_status を incomplete とする（表の項番 9, 12, 15）.

以上を言い換えると以下のようになります.

- 教材作成者は，mastery_score を用いて SCO からの cmi.core.lesson_status を上書きできる．つまり，SCO からの cmi.core.lesson_status ではなく，mastery_score と cmi.core.socre.raw の結果で cmi.core.lesson_status を決定できる．これは，同じ SCO を用いても，SCO が用いられる状況（manifest 中の位置，ないし，SCO が使われる教材）によって合否の基準を変更できる，ということを意味します.
- SCO から cmi.core.lesson_status が設定されなければ，cmi.core.lesson_status は completed になる．これは，演習問題などを含まない解説型の SCO では，SCO の表示によってその SCO を完了したことにする，ということを意味します.
- 上記以外の場合は，SCO からの cmi.core.lesson_status 設定値をそのまま LMS の cmi.core.lesson_status とします.

なお，SCORM2004(1.3)でも，得点の比較による習得状態の決定という考え方は継承されています．ただし，SCORM2004 では lesson_status が completion_status（completed, incomplete）と success_status (passed, failed）に分離されたため，状態の決定アルゴリズムは簡略化されています.

► 実態

SCORM1.2 準拠の LMS は表 3-5 の動作を正しく実装しなくてはなりません．しかし，実態は,

- mastery_score を反映していない.
- mastery_score が設定されていないと動かない.
- mastery_score が 0 だと動作がまちまち.

といった問題を有する LMS が存在します.

► コンテンツ開発者へのお勧め

上記のように lesson_status, score, mastery_score の関係は，LMS によってきちんと実装されていない場合が多く，トラブルのもとになっています．特に，mastery_score を設定してもそれが反映されない，という場合が多いようです．lesson_status, score はそのまま LMS で記録されるという場合が多いようですので，習得状態決定のアルゴリズムが mastery_score の影響で LMS によって左右されないよう，mastery_score を設定しないのが最も安全な使用方法と考えられます．しかし，中には mastery_score を設定しないと動作しない，というたちの悪い LMS もあり完全な回避策は無いようです．したがって，標準規格の主旨に反しますが，事前に使用する LMS の仕様をよく確認する必要があります.

► LMS 開発者へのお勧め

とにかく表 3-5 の動作を正しく実装するべきです.

mastery_score が設定されていないと動かない LMS があるようですが，教材側で mastery_score を設定するかしないかは任意であり，mastery_score が設定されていない場合は，SCO からの cmi.core.lesson_status 設定値をそのまま LMS の cmi.core.lesson_status としなくてはなりません.

また，mastery_score が 0 というのは，0 点以上を取れば合格ということです．この場合も，もし cmi.core.socre.raw が SCO から設定されなければ SCO からの cmi.core.lesson_status 設定値をそのまま LMS の cmi.core.lesson_status としなくてはなりません.

3.4.4 前提条件と lesson_status

► 規格

SCORM Ver.1.2 では，前提条件（prerequisites）を用いて教材の動作を制御することができます．すなわち，アグリゲーションや SCO に対して，そのリソースを提示するために事前に満たしていないとならない条件を記述することができます．前提条件の記法の詳細は規格書に記載されています(CAM 2.3.2.5.1 Sequencing and Navigation Today)．この条件を満たしていない場合，LMS は当該するリソースを提示してはなりません.

前提条件の記述例を表 3-6 に示します.

表 3-6 前提条件の記述例

記述	意味
S11	S11 が completed ないし passed ならば真
S11 & S12	S11，S12 が共に completed ないし passed ならば真
S11= “passed”	S11 が passed ならば真
S11= “passed | S12= “passed	S11，S12 が共に passed ならば真

► 実態

LMS によって前提条件を実装しているものとしていないものが存在します．また，前提条件が満たされない場合にリソースを提示しないとして，それでは具体的にどのような動作をすればよいのか（エラーメッセージを提示するのか，SCO をスキップするのか，など）が SCORM Ver.1.2 では明記されていません．条件に記述できるのも SCO の状態のみで，アグリゲーションの習得状態は反映されません．実用的なシーケンシングを行うためには SCORM Ver.1.2 の前提条件は機能的に不十分です．このための機能拡張が SCORM2004(1.3)で行われています.

► コンテンツ開発者へのお勧め

規格の記述があいまいで，LMS によって動作が異なる可能性が高いため，前提条件の使用はあまり勧められません.

► **LMS 開発者へのお勧め**

規格の記述があいまいなため明確な LMS の実装指針を出すのは難しい状況です．前提条件が成立しない SCO を選択しようとしたら，エラーメッセージを出して元の画面に戻る，というのがもっともオーソドックスな実装方法と考えられます．

3.4.5 Mandatory と Optional

► **規格**

SCORM ランタイム環境のデータモデル要素は Mandatory（必須）要素と Optional（任意）要素に分かれています．LMS は必須要素をすべて実装しなくてはなりません．任意要素を実装するかどうかは自由です(RTE 3.4.2 Data Model Elements)．一方，SCO はどの要素も自由に使用することができます．

LMS がどの任意要素を実装しているかは，_children（RTE 3.4.1.2 Data Model Elements）というキーワードで知ることができます．SCO が以下のように LMSGetValue を発行すると実装されているデータ項目を知ることができます．

LMSGetValue(“cmi.core.score._children”)

→ “raw” 必須要素のみをサポートしている LMS の場合

→ “raw, min, max” 任意要素をふくめ全てをサポートしている LMS の場合

また，実装していないデータ要素に対して，SCO が LMSGetValue, LMSSetValue を発行した場合，LMS はエラーコードを “401 - Not implemented error” に設定しなくてはなりません．

► **実態**

上記のように LMS が任意要素を実装するか否かは LMS ベンダの判断に任されています．このため市場には，必須要素のみ実装した LMS，任意要素の一部も実装した LMS，全ての要素を実装した LMS が混在します．

このような状況で，LMS が実装していないデータ要素を SCO が使おうとした場合に問題が発生します．LMS が上記の規格通りに，_children や 401 エラーを実装していれば良いのですが必ずしも規格通りに実装されていない場合があります．

また，SCO 側も LMSGetValue, LMSSetValue の前に _children を確認していない，LMSGetValue, LMSSetValue を発行した後のエラーチェックをしていないという場合が多く見られます．

このため，LMS が実装していないデータ要素を SCO が使おうとした場合，LMS，SCO のいずれかがハングしたり，予期しないエラーを起こしたりします．

► **コンテンツ開発者へのお勧め**

まず，データ要素に必須と任意の区別があること，必須要素のみ実装した LMS，任意要素の一部も実装した LMS, 全ての要素を実装した LMS, が存在することを理解しましょう．

その上で，任意要素を使う場合には _children の確認，LMSGetValue, LMSSetValue を発行した後のエラーチェックを行うようにしましょう．任意要素が使えない場合にどのような動作とするかも設計時の注意事項として必要です．

► LMS 開発者へのお勧め

全てのデータ項目を実装できれば問題ないのですが, そうでない場合は, _children や 401 エラーをちゃんと実装し，サポートしていないデータ項目が SCO からアクセスされてもエラー処理が確実に行われるようにしましょう．

3.4.6 リストデータ項目の添え字の順序

► 規格

データ要素にはリスト型（配列型）のものがあります．例えば cmi.interactions のようなデータ要素では，SCO は，cmi.interactions.0.id, cmi.interactions.1.id, cmi.interactions.2.id,...に順次データを書き込んでいきます．規格では，このデータ要素の番号は順番に増加させていかなくてはなりません（RTE 3.4.3 リストの扱い）．

このために SCO は_count を用いてリストの総要素数を知ることができます. リスト中のデータ要素の番号は 0 から始まるので，_count の値が次の新しいデータ要素の番号になります．規格書には以下のような例が掲載されています．

```
// LMS から interactions の総レコード数を取得
var totalInteractions = LMSGetValue("cmi.interactions._count")
// 得られた総レコード数を次のデータ要素の番号とする．
var request = "cmi.interactions." + totalInteractions + ".id"
// Interaction ID を設定
LMSSetValue(request, "Int_110")
```

番号を飛ばすと LMS はエラーを返します．

► 実態

この内容で問題が起きたという事例はあまり報告されていません．リスト型（配列型）のデータ要素は，cmi.interactions, cmi.objectives のように任意要素のものが多く，あまり利用されていないのが，その理由かもしれません．ADL の Test Suite では，LMS の試験で，SCO がリスト型の番号を飛ばした場合にエラーを返すかどうかをテストしています．番号をチェックしていない LMS は非準拠とみなされます．

► コンテンツ開発者へのお勧め

リスト型（配列型）のデータ要素を使用する際には，上記のサンプルのように_count を利用して，正しく番号を付与する必要があります．

► **LMS 開発者へのお勧め**

LMS では_count に正しい値を設定するとともに, 送信されたデータ要素の番号が正しいかどうかのチェックを実装します．この機能は上記のように ADL Test Suite の検査項目になっています.

3.4.7 lesson_status の解釈

► **規格**

SCORM Ver.1.2 では, lesson_status は, passed, completed, failed, incomplete, browsed, not attempted の 6 つのいずれかの値を取ります. これらのうち, passed, completed, failed, incomplete は規格では以下のように規定されています.

- passed: SCO の中の学習目標が必要数習得された，ないし，必要な得点が達成された．学習者は SCO を完了(completed)し，合格(passed)とみなされる．
- completed: SCO は合格(passed)かどうか不明だが, 学習者は SCO 中の全ての要素を見ている．学習者は SCO を完了(completed)したものとみなされる．例えば, 合格は LMS 側の特定の得点に依存する．SCO は生の得点を知っているが，それが合格に必要かどうかは知らない.
- failed: SCO は合格(passed)していない．学習者は SCO 中の全ての要素を完了(completed)したかどうかは不明である．学習者は SCO を完了(completed)し，不合格(failed)とみなされる．
- incomplete: SCO を開始したが終わっていない.

進捗（完了か未完了か）と習得（合格か不合格か）という状態を考えるとすると，上記は以下のように整理できます.

lesseon_status	進捗	習得
passed	完了	合格
completed	完了	不明
failed	完了	不合格
incompleted	未完了	不明

► **実態**

上記のように lesson_status では，進捗と習得の完全に概念が分離されていません．未完了だが合格，といった状態は実際にはあり得ると考えられますが，それを表現する方法がありません.

► **コンテンツ開発者へのお勧め**

この問題は SCORM1.2 の不備な点のひとつです．しかし規格では一応上記のような規定がなされていますので，これを守るのがもっともベストと考えられます．いったん合格に

なった SCO を再度実行したときに，不合格や未完了に戻すか否かといった判断は lesson_status の値を LMSGetValue して SCO 側で行う必要があります．なお，SCORM2004 では進捗と習得に対応するデータモデル要素を設けて，この問題を解消しています.

► LMS 開発者へのお勧め

この問題に関しては LMS 側で対処する方法はありません. 3.4.3 に述べた Mastery Score との関係を除くと, LMS は SCO から送られてきた lesson_status の値をそのまま格納するべきです. 一旦合格になったら, あとで不合格になっても合格のまま, といった判断は SCO 側に任せるべきです.

3.5 その他

3.5.1 SCORM 準拠 LMS の定義

► 規格

SCORM 規格準拠 LMS は SCORM コンテンツパッケージを読み込めなくてはなりません．ただし，読み込む手順や方法については規格では何も定められていません.

► 実態

LMS が SCORM コンテンツパッケージを読み込む方法には大きく以下のような形態があると考えられます.

形態 1） ブラウザを用いた LMS の管理画面から，SCORM コンテンツパッケージをアップロードする． SCORM コンテンツパッケージを LMS に直接読み込む形態です.

形態 2） LMS 付属のコンバータ，ないし，オーサリングツールに SCORM コンテンツパッケージを読み込み，LMS 独自形式のファイルに変換する．この独自形式ファイルをブラウザを用いた LMS の管理画面からアップロードする．このオーサリングツールは SCORM コンテンツパッケージを作成・出力する機能を有する場合もありますが，LMS が読み込むのはあくまで独自形式ファイルです.

注意が必要なのは形態 2 の場合です．コンバータ，ないし，オーサリングツールを含めたシステムとしては SCORM コンテンツパッケージを読み込んで動作する LMS なのでなんら規格には違反していません．ただし，オーサリングツールが SCORM コンテンツパッケージを読み込むだけで，書き出す機能を持たない場合，このシステムは，SCORM コンテンツパッケージを実行することはできますが，作成することはできません．従って，このシステムで作成したコンテンツを他の SCORM 準拠 LMS に移植することはできません．このような形態は，もともと独自形式コンテンツを採用していたシステムをあとから

SCORM 対応化した場合に良く見られます．繰り返しますが，このようなシステムでも，LMS としてはなんら規格には違反していません．

► コンテンツ開発者へのお勧め

コンテンツを作成する際に，オーサリングツール，LMS が読み書きするコンテンツの形式をよく確認しましょう．特に SCORM 準拠 LMS に付属のオーサリングツールでは SCORM コンテンツパッケージを読み込むだけで，書き出しできない場合があります．

3.5.2 LMS のコマンド GUI

► 規格

SCO を切り替える手段として「前画面」,「次画面」などのコマンドを用いるか，コンテンツ階層を表示した「目次」を用いるか，コマンドの種別，配置，名称，アイコン形状をどうするか,「目次」の表示形式をどうするかなどに関しては SCORM では一切規定がありません．

► 実態

SCO を切り替える機能は LMS 側の実装に任されています．従って，ベンダによって千差万別の GUI が存在することになります．次 SCO に移動するコマンドが，ある LMS では「次画面」，他では「次ページ」，他ではコマンドではなく目次から選択する，といった具合です．LMS 側の機能であるために，コンテンツから制御することは不可能で，LMS が異なるとユーザインターフェースが異なってしまうという，コンテンツベンダにとっては不都合な状況を招いています．

► コンテンツ開発者へのお勧め

SCO 中で「次画面を押してください」といった特定の LMS に依存した表現は避けるようにし，学習者の混乱を招かない配慮をします．

SCORM2004 では SCO 切り替えのコマンドが統一され，SCO から「次画面」,「前画面」のコマンドが発行できるようになるなど，この問題がある程度解決されています．

► LMS 開発者へのお勧め

コマンド GUI がカスタマイズしやすい実装とすることが望まれます．できればコンテンツごとに GUI がカスタマイズできる構成とすることがベストですが，それでも根本的な解決は困難です．

上記のように SCORM2004 では，LMS 側で SCO から発行されたコマンドを解釈する機能の実装が必須となります．

3.5.3 SCORM 非準拠コンテンツの移植

► 規格

SCORM 規格では非準拠コンテンツを移植するガイドラインは定められていません.

► 実態

SCORM 規格は AICC 規格をベースとし, 多くの独自 WBT 実装の共通項を抽出するように作成されました. 例えば, SCORM 規格以前にも多くの WBT システムが存在していましたが, 階層型の教材構造は多くのシステムで共通していました. 階層のページに相当する部分に HTML ファイルを割り当てるという構造も比較的良く見られる一般的な考え方です. 従ってこれらのシステムで作成されたコンテンツは比較的容易に SCORM コンテンツに変換が可能と考えられます.

しかし, ナビゲーション機能はシステムによってまちまちであり, 演習問題の実装方法も古いシステムではクライアント側でなくサーバ側に持たせるのが一般的でした. これらは SCORM の規格と異なるため, 移植の際の難度が高くなります. さらに, シミュレーション機能, ヘルプ機能, レポート機能, 掲示板機能, Web のハイパーリンクを用いた機能などを組み込んだコンテンツでは, クライアントとサーバがより密接に連携した実装形態となることが多く, 移植はより困難になります.

► コンテンツ開発者へのお勧め

上記のようなコンテンツは, SCORM 環境でどの LMS でも同様の機能を満たすように移植することは非常に困難です. 移植の目的を明確にし, どの LMS でも稼動するような相互運用性を追及するのか, 機能面を重視するのかを判断して移植を行います.

► LMS 開発者へのお勧め

この問題に関して LMS 側でできることはあまり多くありません. SCORM2004 に対応すれば, ナビゲーション機能に関してはかなりの再現が可能となります. その他の機能を標準的な環境で実現するには, Web サービスなどに基づいて各種機能を提供する標準規格の策定が必要となります.