



---

# Report 1: JISC Good APIs Management Report

---

A review of good practice in the  
provision of machine interfaces and  
use of services

## Document details

Author:	Marieke Guy
Date:	May 2009
Version:	Final
Document Name:	good_api_JISC_report_final.doc
Notes:	

## Acknowledgements

UKOLN is funded by the MLA: The Museums, Libraries and Archives Council, the Joint Information Systems Committee (JISC) of the Higher and Further Education Funding Councils, as well as by project funding from the JISC and the European Union. UKOLN also receives support from the University of Bath where it is based.

The project team are grateful to all those who gave up time to help with the report. Vital to this work were the people who filled in the questionnaire, those who responded the request for interviews and everyone else who made documentation available.

This report was commissioned by JISC.

# Table of Contents

<b>1</b>	<b>EXECUTIVE SUMMARY</b> .....	<b>1</b>
1.1	INTENDED TARGET AUDIENCE .....	1
1.2	DEFINITIONS.....	1
1.3	LICENCE.....	1
1.4	RECOMMENDATIONS.....	1
<b>2</b>	<b>INTRODUCTION AND TERMS OF REFERENCE</b> .....	<b>2</b>
2.1	THE NATURE OF THIS REPORT.....	2
2.2	QUOTATIONS USED IN THE REPORT .....	2
2.3	WHAT ARE APIS? .....	2
2.4	AIMS AND OBJECTIVES OF THE PROJECT.....	2
2.4.1	<i>The Developer Perspective</i> .....	3
<b>3</b>	<b>METHODOLOGY</b> .....	<b>3</b>
3.1	LITERATURE REVIEW .....	3
3.2	CONSULTATION MECHANISMS .....	3
3.2.1	<i>Online Survey</i> .....	3
3.2.2	<i>Interviews</i> .....	3
3.3	EVENTS.....	3
3.3.1	<i>CETIS Conference</i> .....	3
3.3.2	<i>JISC Developer Happiness Days (dev8d)</i> .....	3
3.4	BLOG .....	3
3.5	DISSEMINATION .....	3
3.6	IE DEMONSTRATOR .....	3
<b>4</b>	<b>ABOUT APIS</b> .....	<b>3</b>
4.1	DEFINITION.....	3
4.2	TYPES OF APIS .....	3
4.3	PROCESSES IN THE USE OF APIS .....	3
<b>5</b>	<b>BACKGROUND TO API USE IN THE UK HE SECTOR</b> .....	<b>3</b>
5.1	PROVISION OF APIS.....	3
5.2	OTHER SECTORS.....	3
<b>6</b>	<b>POTENTIAL BENEFITS OF PROVISION OF APIS</b> .....	<b>3</b>
6.1	REUSE .....	3
6.1.1	<i>More Access</i> .....	3
6.1.2	<i>Wider Audience</i> .....	3
6.1.3	<i>Extensibility</i> .....	3
6.1.4	<i>Mashups</i> .....	3
6.2	UNEXPECTED USE .....	3
6.3	DEVELOPING FOR NICHE MARKETS.....	3
6.4	CODE IMPROVEMENT .....	3
6.5	CREATION OF CODE BASE.....	3
6.6	GRAVITATION TOWARDS STANDARDS.....	3
6.7	ENCOURAGES DOCUMENTATION.....	3
6.8	INNOVATION AND CONFIDENCE .....	3
<b>7</b>	<b>CHALLENGES OF PROVISION OF APIS</b> .....	<b>3</b>
7.1	MAINTENANCE AND RESOURCE IMPLICATIONS.....	3
7.2	DEVELOPMENT FREEZE.....	3
7.3	FUNCTIONALITY LIMITATIONS.....	3
7.4	BADLY WRITTEN APIS .....	3
7.5	FRAGILITY .....	3
7.6	SECURITY.....	3
7.7	LICENCE ISSUES .....	3

7.8	ABSTRACTION LEVEL .....	3
7.9	EXPECTATIONS .....	3
7.10	GETTING PEOPLE TO USE IT .....	3
7.11	SKILL LEVEL NEEDED.....	3
7.12	FORMALITY.....	3
<b>8</b>	<b>GOOD PRACTICE FOR PROVISION OF APIS .....</b>	<b>3</b>
<b>9</b>	<b>PROBLEMS WHEN USING THIRD-PARTY APIS.....</b>	<b>3</b>
9.1	POOR DOCUMENTATION.....	3
9.2	POOR TECHNOLOGY .....	3
9.3	BADLY WRITTEN APIS .....	3
9.4	LEGAL ISSUES .....	3
<b>10</b>	<b>GOOD PRACTICE FOR CONSUMING APIS .....</b>	<b>3</b>
<b>11</b>	<b>SUGGESTIONS FOR FUTURE WORK.....</b>	<b>3</b>
11.1	SUPPORT DEVELOPERS .....	3
11.2	SUPPORT DEVELOPMENT OF APIS.....	3
11.3	FUND MORE RESEARCH .....	3
11.4	MONITOR USAGE .....	3
11.5	DISSEMINATE.....	3
11.6	API SPECIFICATION/STANDARDS .....	3
11.7	LICENSING.....	3
11.8	API REPOSITORY.....	3
11.9	SUSTAINABILITY OF OUTPUTS OF PROJECT .....	3
<b>12</b>	<b>CONCLUSIONS.....</b>	<b>3</b>
<b>13</b>	<b>CASE STUDIES.....</b>	<b>3</b>
13.1	SPLASH, UNIVERSITY OF SUSSEX .....	3
13.2	THE OPEN SOURCE DEBATE: A GOOD API IS NOT ENOUGH.....	3
<b>14</b>	<b>APPENDIX A: GOOD APIS SURVEY QUESTIONS .....</b>	<b>3</b>
<b>15</b>	<b>APPENDIX B: STATISTICS FROM THE GOOD APIS SURVEY .....</b>	<b>3</b>
<b>16</b>	<b>APPENDIX C: PEOPLE CONSULTED DURING THE STUDY .....</b>	<b>3</b>
<b>17</b>	<b>APPENDIX D: POTENTIAL TOPICS FOR FUTURE JISC REPORTS .....</b>	<b>3</b>
17.1	CASE STUDIES .....	3
17.2	LISTS .....	3
17.3	GUIDES .....	3
17.4	SURVEYS .....	3
17.5	STATISTICAL DATA.....	3
17.6	STANDARDS INFORMATION .....	3
17.7	DISCUSSION .....	3
17.8	TOOLKITS.....	3
17.9	THE BIGGER PICTURE.....	3
<b>18</b>	<b>APPENDIX E: DEFINITIONS .....</b>	<b>3</b>
<b>19</b>	<b>AUTHOR CONTACT DETAILS.....</b>	<b>3</b>
<b>20</b>	<b>REFERENCES.....</b>	<b>3</b>

# JISC Good APIs Management Report

## *A review of good practice in the provision of machine interfaces and use of services*

### 1 Executive Summary

The JISC funds research and innovation programmes in the use of ICT in teaching, learning and in the building of knowledge and service development. Recent JISC programmes have supported development in the provision of machine interfaces and use of services and the JISC recognises that use and provision of APIs by developers working on JISC-funded projects is on the increase.

The Good APIs project was initiated as a review of API activity in the Higher Education arena.

This report is primarily a management report. It is an account of the Good APIs project methodology through two key consultation mechanisms: an online survey of and interviews with the HE developer community. Based upon the information gathered it offers a background to API use in the UK HE sector, the potential benefits of the provision of APIs and the challenges this provision can instigate. The report also reviews potential problems developers can face when using third-party APIs.

Through the consultation carried out a set of best practice techniques for API creation and consumption were established, these are provided in a second report.

Some conclusions have been tentatively drawn from the work carried out and a small set of recommendations made for JISC. The key recommendation is that to better support developer activities more work is need in this area (wider consultation etc.) To support these recommendations a larger number of suggestions for future work have been provided. A significant number of ideas for potential topics for future JISC reports have also been made available in the form of an appendix.

#### 1.1 Intended Target Audience

This report is intended for use by the report commissioners, JISC. It will also have some relevance for managers by providing them with a better understanding of the benefits and challenges of API use in the Higher Education Sector. Some sections may be of interest to HE developers.

#### 1.2 Definitions

Definitions of key terms as they are used in this report appear in Appendix E. The meaning of the term API is better defined in the *Introduction* and the section *About APIs*.

#### 1.3 Licence

This report is licensed under a Creative Commons Attribution-Non-Commercial 2.0 UK: England & Wales Licence<sup>i</sup>.

#### 1.4 Recommendations

The following recommendations are made for the JISC.

1. Future research into API creation and consumption should be funded.
2. Consideration should be given to supporting the growth of the Higher Education development community through events and community engagement.
3. There should be encouragement of greater provision of APIs by funded projects.
4. A knowledge-base of resources to aid development should be established.

## 2 Introduction and Terms of Reference

### 2.1 The Nature of this Report

This report is based upon information obtained from a number of consultation mechanisms including an online survey. Many of the points made are drawn directly from the survey. While the use of a survey aids towards consensus it also offers one-off individual advice. Such advice may not have been discussed by the community and may not have been agreed upon by experts. It is hoped that future activity will allow this discussion and agreement to take place.

### 2.2 Quotations used in the Report

The quotations given in *block quote style* in this report are from the Good APIs survey. When conducting the survey it was indicated that all information would be anonymised and so all quotations are not attributed.

### 2.3 What are APIs?

Application programming interfaces (APIs) are a set of functions, procedures or classes that an operating system, library or service provides to support requests made by computer programs. An API is a machine-to-machine interface rather than a user interface. It allows developers to access the functionality of other software modules through well-defined data structures and subroutine calls.

Web sites have in recent times evolved from static HTML pages to dynamic applications. Much of a sites activity involves cross linking information. The Web 2.0 premise of openness has sparked a trend among software companies and organisations of opening up their APIs for use by others.

*Someone once said "Application Interfaces are Social Constructs"; I find it helpful to keep this in mind. It's only an API while it's "my system" and "your system".*

### 2.4 Aims and Objectives of the Project

The 'Good APIs' project aims to provide JISC and the sector with information and advice on the factors that encourage use of machine interfaces, based on existing practice and consensus within the development community.

The main objectives of the project are defined as:

- to define a set of criteria to use when selecting instances of the effective use of third party machine interfaces by the Higher Education (HE) community

- to identify instances of the effective use of third party machine interfaces, and document them

- by reference to the above, to draw general lessons with respect to good practice making services and data available via machine interfaces, in particular reviewing how developers within the HE sector are using Web APIs offered by both commercial sites such as Twitter, Facebook, Google, and non-commercial sites such as SherpaRoMEO.

- to document the extent to which Web-based data and services managed within the UK HE community have learned the lessons and conform to the good practice so identified

- if Web-based data and services managed within the UK HE community have not learned the lessons and/or do not conform to the good practice so identified, to explain why, and to suggest ways to encourage improvements where appropriate

The main proposed deliverables for the project have been defined as:

Documented good practices and lessons with respect to making data and services available via machine APIs

A report reviewing current practice in UK HE on these issues

A small set of practical recommendations which, if implemented, should result in a demonstrable improvement in the provision of services.

### 2.4.1 The Developer Perspective

When asked what they felt a report on good APIs could offer the development community developers provided a variety of responses:

**Knowledge:** Many developers felt that currently very few people know what to do in this area. This void of knowledge is not in the area of development or programming skills but in the areas of management and direction. At this point in time there is no model or framework for API activity in the HE sector. People are often just '*chucking an API in there*' with little regard for what fellow developers or users want or how users can use APIs as part of their learning experience.

**Community:** Building an HE development community requires making contact with other people, finding out how they are thinking about using APIs, forming solid developer relationships and bringing content together. Some of this requires new ways of learning and engagement with external services. The report could aid in the establishment of this community and through its recommendations provide support for this community in the future.

**Inspiration:** Many developers consider coding to be a highly creative activity and feel that the creation of useful APIs often needs an inspirational moment. For projects to interact APIs shouldn't be a special add on. Creativity requires space and support and JISC could again support activity here.

## 3 Methodology

### 3.1 Literature Review

A vast amount of information about APIs and their use is available on the Web. Many of these resources have been bookmarked on the delicious social bookmarking service using the good-apis-jisc tag<sup>iii</sup>. However for the purpose of this report although many of these resources have been consulted it has been felt necessary to keep the referral to them to a minimum. The reason is that the main body of this report has been drawn from the experiences and feedback of those working primarily in an HE environment. The Higher Education sector offers different challenges to developers than the commercial world. The literature review also involved monitoring of related Twitter posts and discussion lists.

### 3.2 Consultation Mechanisms

#### 3.2.1 Online Survey

The main body of information for this report was acquired through the conduct of an online survey. The survey was conducted in late January and hosted on the SurveyMonkey<sup>iv</sup> site. SurveyMonkey is a private company that enables users to create their own Web-based surveys. It is well used in the HE/FE arena. The survey<sup>v</sup> (questions listed in appendix) was advertised on a number of Academic developer lists and the link posted on several prominent blogs. In total 240 people filled in the survey. Statistical data on the spread of participants is given in Appendix B.

### 3.2.2 Interviews

After discussion among the team a number of potential key interviewees were identified. These people were interviewed by email or at events; a list is included as Appendix C.

## 3.3 Events

In parallel to the survey the Good APIs team attended a number of relevant events and solicited feedback at them. Establishment of a dialogue with the development community was agreed early on by the project team to be a necessary constituent if the project was to be a success. Informal interviews and discussions with developers provided a practical and cost-effective way of obtaining new and fresh comment.

### 3.3.1 CETIS Conference

The Good APIs project was represented at the recent CETIS Conference 2008<sup>vi</sup>. The conference was organised by JISC CETIS is a JISC Innovation Support Centre providing advice to the UK Higher and Post-16 Education sectors on educational technology and standards. Its theme was Technology for Learning, Teaching and the Institution.

The project team facilitated a half-day session on Innovation in a World of Web APIs. In the first part of the session people were asked to talk about what they were currently working on. The second part of the session was spent looking at the future. This also involved exploring best practices for Web APIs. The set up was a quasi focus group with developers discussing a number of different issues in the publication and consumption of APIs. All the comments made by developers were documented and feedback from the session was added to the session wiki<sup>vii</sup>. Further resources and handouts from the session, including a brief presentation on the Good APIs project, are available from the UKOLN Web site<sup>viii</sup>.

### 3.3.2 JISC Developer Happiness Days (dev8d)

The Good APIs project team was also represented at the JISC Developer Happiness Days (dev8d)<sup>ix</sup> event at Birkbeck, University of London.

The 5 day event co-ordinated by JISC was aimed at developers in UK Higher Education, however there were also places available for developers outside the UK and from outside HE. 'Technological tinkerers' (people who would not classify themselves as developers but enjoy working with applications and code) were also invited to come along, as were a number of what were called 'Uber Users' (highly knowledgeable users). The pre-event day was designed specifically to develop skills and get people up to speed in a number of development languages. The rest of the week provided an opportunity for educational software developers to get together to pool ideas and take part in a developer decathlon: a two-day team coding session with prizes for the best code.

While at dev8d the project team initiated a number of informal discussions with developers using questions from the survey. Many were keen to offer comment on where they want software development to go in the education world, and how they feel JISC could help take them there. Most were very 'excited' to have been given the opportunity to spend such a big amount of time 'just making stuff'. The event acknowledged JISC's commitment to developers and there was a recognition of the current opportunity HE developers have to be heard. As Paul Walk explained on the event blog when asked what he was looking for as a Developer Decathlon judge<sup>x</sup>:

*"I'm looking to demonstrate that developers, especially when working directly with users, have much to offer the community. It's about giving developers a voice – some developers have had to take a day's leave to come here because there can be a lack of appreciation from managers of the value of allowing their developers to get together with other developers to share ideas."*

The event provided a number of useful resources including a series of five-minute interviews on the blog<sup>xi</sup> and an announcement Twitter channel<sup>xii</sup>.

### 3.4 Blog

The Good APIs blog<sup>xiii</sup> was created as a lightweight facility for reporting on activities. It has also been a medium in which to allow people to give feedback on current 'thinking'. Different ideas with regard to best practice have been 'put out' in the community to allow comments.

### 3.5 Dissemination

It is anticipated that the good practice elements of this work will be shared with the HE developer community by chunking into a blog, either the writetoreply.org blog<sup>xiv</sup> or the Good APIs blog. Each section will be one post, and each post is open to comments from the public. This approach will probably take place over a specified timescale. It is also intended to use the report as material for a number of blog posts, online articles briefing papers and workshop events.

### 3.6 IE Demonstrator

The IE Demonstrator is a JISC-funded initiative to provide a focus for and showcase of development activities funded under the JISC Information Environment Programme. A knowledge-base of resources to aid development in this space is under development. Some outputs from the Good APIs report will be presented in the context of the IE Demonstrator for further comment, allowing ongoing development and review.

## 4 About APIs

### 4.1 Definition

During the research carried out for this report it has become increasingly clear that what constitutes an API is ill defined and confusing. As all activity in the provision of machine interfaces to Web data and services is of interest a useful term that has arisen is 'API-ness', i.e. connected to APIs. The term is used in various blog posts and presentations available on the Web. It appears that there is a spectrum of activities that have elements of API-ness, from sharing of code libraries to use of Web and traditional APIs. This report attempts to be non discriminatory in the importance of various types of activity. However the nature of the academic world in which we work in means that much of the information offered has tended to come from on the area of Web APIs.

### 4.2 Types of APIs

Although some have found it useful to make a distinction between traditional and Web type of API for this report some better distinctions might include:

- projects which provide an API versus projects which are actually developing a Standard API (e.g. OpenDOAR offers an API, SWORD is a project defining one).
- APIs which are about offering simple access to resources (e.g. RESTful) versus ones which are about exploiting some re-usable function (e.g. APIs for workflow systems)
- APIs which are tied to programming languages versus those which are not (SOAP/ HTTP REST/XML RPC)

### 4.3 Processes in the Use of APIs

Throughout the report it is necessary to refer to two different processes in the use of APIs. The first: *provision of* (or providing/publishing) refers to the release of an API by a developer for others to use. The second: *consuming* is the use of an external (or third party) API by a developer for the creation of an application, or other.

Although both processes require skill it has been observed that constructing and provision of a useful API is more difficult technically than consuming one. Although some developers may find themselves carrying out both processes it should also be acknowledged that they may be of interest to different audiences. It is likely that there will be more developers consuming APIs than providing them. Also the people primarily consuming APIs may have different skill sets than those constructing and publishing them. This report aims to offer advice for both types of developer and so has divided up much of its content using the two processes as separators.

## 5 Background to API use in the UK HE Sector

Educational software development, with or without the Web as a platform, continues to be a growing area. With the Web being the primary marketing tool for the University development teams are increasingly moving from back room research activities to front-line projects that are more visual and have greater 'impact'. The current economic situation and the security of the public sector is likely to see well-qualified commercial developers moving into the HE arena. Development teams are growing and Web development teams especially are being recognised as a potentially lucrative workforce that can bring in money for institutions. Within JISC projects there is recognition that use of and provision of APIs is increasingly taking place.

Currently the technical understanding required to get involved in development excludes most educationalists but simpler environments (like Yahoo pipes<sup>xv</sup>) are slowly making entry easier. Yahoo pipes provides a graphical user interface for building data mashups meaning potential developers can create applications without using code.

These factors all mean that the release and consumption of APIs is a growing area in HE yet there are still few standards, little consistency and few recognised API best practice.

### 5.1 Provision of APIs

Of those 130 who answered the survey question on 'provision of APIs for development work' 45% answered that they do currently publish APIs. Many of those surveyed answered that although they weren't currently doing this they intend to do so in the very near future.

Of those APIs being made available, many at this stage are purely for in-house/internal use. However, once trialed, a number of these will be released for public consumption. Some people surveyed revealed that they create APIs for non-work-related applications (both academic and non-academic). This extra curricular development may go alongside current work related development or be carried out by those not involved in development activities at work.

What constitutes provision of an API is a matter for discussion). Some provided APIs by exposing data and functionality via REST or other means thus allowing third party tools to work with hosted versions of their code. Others offered lower level provision by allowing people to extend the functionality of their systems for their own hosting without needing to touch any code directly. Others felt that various data export features such as RSS and OAI could loosely be held under the banner of API release. Some who answered were in the process of launching infrastructures that provide access to resources; and interpretative data via an API.

From these results it seems likely that API release will become an increasingly important area of work.

Some projects already offering APIs include:

OpenDOAR<sup>xvi</sup>

OpenDOAR Prototype Protocol for Statistical Harvesting<sup>xvii</sup>

SHERPA/RoMEO Prototype<sup>xviii</sup>

WWWOPAC<sup>xix</sup>  
Museum of London<sup>xx</sup>  
ArXiv<sup>xxi</sup>  
Oxford University Research Archive (ORA)<sup>xxii</sup>  
GeoCrossWalk<sup>xxiii</sup>  
Celtic coin index for Oxford University<sup>xxiv</sup>  
PROD<sup>xxv</sup>  
WebPA<sup>xxvi</sup>  
Promethean Planet<sup>xxvii</sup>  
Group Manager<sup>xxviii</sup>  
The SWORD deposit API<sup>xxix</sup>  
Splash<sup>xxx</sup>  
Joe Cutting<sup>xxxi</sup>

Note that the projects creating these APIs have all felt themselves to be related to HE in some way (i.e. funded by JISC, based in a University or other).

There was also mention of APIs being used for GRID work, repository work, VREs including portal interfaces, preservation storage repositories, maps and map based solutions, corporate information (e.g. staff, student, module and programme information). Description of API sizes ranged from a few calls to high tens of classes and high hundreds of methods.

At present the biggest list of available Web APIs is on Programmable Web<sup>xxxii</sup>. Although there is a government category there is currently not an education category. Research has not found any other comprehensive Web API Directory for education.

## 5.2 Other Sectors

An in-depth look at what is happening in the other sectors is not part of the remit of this report but obviously many commercial organisations are releasing and using APIs. The extensive number of API directories which includes the Programmable Web, Webmashup<sup>xxxiii</sup>. And WebAPI directory<sup>xxxiv</sup> demonstrates this.

In the public sector library developers have led the way somewhat with the sharing of APIs, possibly because of the vast quantity of data sets that are accessible to them. Roy Tenant collected a list of useful APIs for library services on his TechEssense.info blog<sup>xxxv</sup>. This list was added to during the Mashed Library<sup>xxxvi</sup> event held at Birbeck College in November 2008.

Another set of developers in the public sector that are starting to do more in this area are museum developers. Although there are currently only a small number of APIs available discussion on the Museums and Computers Group<sup>xxxvii</sup> indicates that this is about to change<sup>xxxviii</sup> with big players like the Brooklyn Museum and the Powerhouse Museum releasing APIs and the UK Natural History Museum about to embark on iterative testing. One of the biggest challenges for these groups is working on standards for related data. This is an area which could be the basis for further investigation.

## 6 Potential Benefits of Provision of APIs

The release of an API, or series of APIs, allows smoother data transfer and has enormous potential benefits for an organisation or project.

They key ones identified through the survey were:

## 6.1 Reuse

### 6.1.1 More Access

The concept of open data is far from new but has in recent times become increasingly associated with releasing of APIs. By making APIs available an organisation or project makes it easier for people or programmes to access and use their data and services. This means that API providers benefit from their data and functions being used more widely than those who keep their data closed.

Release of APIs touches upon areas of data portability. DataPortability<sup>xxxix</sup> is the idea that users should be able to move, share, and control their personal data.

### 6.1.2 Wider Audience

Provision of APIs enables a provider's core application to reach a much wider audience, whether they consume the API directly or via third-party applications using the API. A provider might see this as a potential way to gain 'greater market share'. This relationship with a wider world than ones institution or organisation opens up huge amount of possibilities. It creates the potential to integrate and connect many different applications and data sets. An example might be the reuse of 'available flats to rent' data by the University of Bath's *Flat Out* Project. By release of an API the data providers (the letting agency) were allowing their data to reach a much wider audience that they may have initially overlooked, in this case the University of Bath student audience.

### 6.1.3 Extensibility

The concept of reuse runs both ways, releasing APIs allows a greater and more adaptive use to be made of services. Other developers are able to re-use these resources by developing their own custom applications with the released data. This in turn adds value to the original work. This functionality and data is quite often material the original developers could not generate themselves ultimately releasing an API has the potential to allow development of additional user interfaces using the API which may be better or more useful than the default interface.

*The Web is a universal platform for building networked applications. APIs allow for rapid development of such applications. Creating an API for external consumption can enable a third party (with time, money and a certain inclination) to extend the capabilities of a service. APIs foster innovation because they enable people to build upon the available service rather than clone existing functionality.*

### 6.1.4 Mashups

An API in combination with other APIs allows "remixing" of services from different sources to create an aggregate which may contain additional functionality that otherwise wouldn't be feasible. Mashups are defined by Wikipedia as "a *Web application that combines data from more than one source into a single integrated tool. The term Mashup implies easy, fast integration, frequently done by access to open APIs and data sources to produce results that were not the original goal of the data owners.*" Realising ones APIs opens up the possibility of others using it to produce mash-ups. Mashups also open up the field by allowing people with a reasonable level of technical skill but as competent as developers to use APIs to construct new Web-based applications. They can make complex things like visualisations accessible to the people to use. As one developer commented "*In the best API mash-ups, the benefits of the combined application are greater than the sum of the constituent APIs.*"

*APIs are the door of extension and integration with others services. Providing APIs to your application make it a "service" and an add value for*

*others applications. The benefits are reciprocal because becoming useful to others services/application you increase the users of your own app.*

## 6.2 Unexpected Use

As explained in the previous point, exposure of APIs spreads your content and brand to areas you wouldn't normally reach and encourages these people to make use of your data and services. This reuse may often be in ways you expected, but may also be in novel ways you hadn't anticipated. Opening up your data and services offers people the flexibility that encourages them to build interesting things on top of our platform. People may be able to do more interesting and innovative things with your data and service which you hadn't thought of or don't have the resources to do. As Rufus Pollack famously said "*The coolest thing to do with your data will be thought of by someone else*". History has often shown this to be the case. Someone with a different perspective and role from you may well invent a more effective use of your service.

Releasing APIs extends the usefulness of the application enabling more possibilities for use of your data or service beyond the Web interface developers have developed for it for. The Web means that users can be anyone and use can be anyway.

## 6.3 Developing for Niche Markets

Releasing APIs allows developers elsewhere in the organisation to develop functionality to meet specific local needs, which can't be justified (in development or support costs) on an organisation-wide level. They also allow users to prototype and explore new requirements, which can later be rolled back into the main product. Releasing APIs can also assist archival and preservation functions or support particularly rare or proprietary functions which are normally not encouraged for general use. It could also lead to future collaborations in niche areas.

## 6.4 Code Improvement

*Releasing APIs you've effectively given yourself a dev team larger than you could ever hope to afford.*

Along with the reuse of your data and services provision of APIs also opens up the opportunity for improvement of your actual code by others. Openness of systems allows other developers to build on your existing business logic. You are not only enlarging your user base but also your developer base.

Different developers have different ideas on how things can be done. An API allows for flexible presentation and analysis of content by a much greater number of people. The results may be fed back into the service development of the source organisation, thus benefiting the end-users as well as the service providers. They allow software reusability, particularly if software is properly modularised. They allow separation of interface (which will probably become obsolete pretty quickly) and data and underlying algorithms (which hopefully will be longer lasting).

This is an activity inline with the open source ideology, described by Yochai Benkler as "*commons based peer production*". Open source code is code "*made collaboratively and shared publicly by a community of equals*". For the founder of the open source movement, Eric Raymond, the virtue of Open Source is its efficiency. It is well documented that Open Source can sometimes create better products faster than the old closed source model.

Open sourcing the code allows those wishing to modify the core application to do so without overcomplicating the API. An example of this approach is Apache HTTP which handles the essential functions of serving Web pages, whilst an API allows extension of the server through an efficient API. This allows the core to be optimised for the features all users need, and optionally functionality to be included in extensions.

Releasing APIs encourages agile development. The resultant wider use of released APIs can often lead to more suggestions for improvements, bug notification and suggestions for patches. The theory is that the more users an API has, the less buggy it will be.

*A good API is closer to the coal face than much technical documentation, but also insulates upstream developers from downstream technical changes. This allows separate testing and development and also reuse.*

Some might argue that releasing APIs reduces the workload of individual programmers. Others that this is not necessarily the case but what it does do is provide developers with an environment in which they can 'bounce ideas around' with fellow developers. Others might also argue that by building for public viewing APIs encourage good architecture, if that's not already in place, by separating data and functionality from presentation.

All in all releasing APIs lowers the barrier to the developer community contributing to your project's development work.

## 6.5 Creation of Code Base

As well as allowing code to be more easily improved upon releasing of APIs has led to the informal creation of a 'code base' where people can easily reuse the work of others. The reuse of existing code encourages modularity and allows the automation of repetitive and time consuming tasks. APIs allow developers to avoid reinventing the wheel by providing standardised access to a range of functionality. APIs can incorporate lots of detailed understanding to commonly used but problematic areas. Tricky tasks can be made easier and done correctly and bug free by using the API e.g. Hibernate<sup>xi</sup>. This encapsulation of common, often repeated or low-level code helps improve developer productivity and ultimately promotes progress in the industry.

One example of APIs incorporating domain knowledge that it is often unnecessary to replicate is the Class `GregorianCalendar`<sup>xii</sup>, an API built into the core Java language. This subclass of `Calendar` is a hybrid calendar that supports both the Julian and Gregorian calendar systems and provides the standard calendar system used by most of the world. To recreate this calendar implementation would be very complicated and require a great deal of research. For example "is 2000 a leap year?", "does 2009 have leap second etc.?"

Having an effective code base has many knock on effects. It cuts down on bugs because you're able to use previously developed, well tested code. Feedback helps detect errors and mistaken assumptions. It reduces data-synchronisation issues, and can also reduce the development time on future projects. It also allows people to extend systems quickly and easily without having to understand the full architecture of a system. This lowers the barriers to entry for those wishing to extend (but not modify) a system.

Code bases result in the creation of a collection of 'templates for good practice'.

## 6.6 Gravitation towards Standards

The openness of APIs allows developers to work in a more shared environment where good work practices can be established. By sharing their work developers are encouraged to keep their code 'tidy'. This encouragement of best practice will hopefully, in time, encourage gravitation towards informal standards (possibly for data, and hopefully for APIs call/response formats themselves). It seems that the more people who expose their data in a structured form (ideally via some sort of API) the more impetus there will be for people to do so in the same way, even if this is "lowest common denominator".

## 6.7 Encourages Documentation

The release of an API forces good documentation of, at the very least the API, which would hopefully lead to good documentation of the underlying code too.

## 6.8 Innovation and Confidence

APIs promote innovation and encourage confidence. Developers who get to see their work used and built upon are likely to be more proactive than those who don't. Many of the users and developers who interact with your API will be passionate about your services, with ideas on how to extend and improve it, and the know-how to implement those great ideas. More applications related to your Web site means more ways for users to interact with it, which means more chance of a "killer feature" written by a user of your service that ends up driving thousands of new users to your site, any one of which can be a developer that continues the cycle. It's an "upward spiral."

*Many content based, document standards are a dead end. APIs permit innovation on the part of those consuming and using them.*

APIs allows innovation among developers and users leading to competition and improvements. They inspire and build confidence.

## 7 Challenges of Provision of APIs

Although the benefits of providing APIs are great there are naturally many challenges for the user and developer.

They key ones identified through the survey were:

### 7.1 Maintenance and Resource Implications

Provision of APIs obviously has a development overhead. Released software always has a support burden and time will be required for ongoing maintenance and additional documentation. If APIs are not maintained it is quite possible that they will become obsolete.

The maintenance issue has particular consequence when you consider APIs developed with JISC funding. The sustainability of resources created becomes difficult when development teams move on to their next project. Many institutions may well find that they end up with multiple systems to manage. Some developers have commented that they are reluctant to release APIs because they do not want to be responsible for them when they break. APIs mean exposure and some developers may be reluctant to open up their code in this way.

Often providing enough forms of APIs to satisfy most developer needs requires multiple implementations using numerous languages. This results in an even bigger developer overhead and a need for an effective infrastructure and community to provide the technical support, tutorials and documentation needed.

The resource issue also relates to the amount of work and time required to evolve a good API initially, if this hasn't been written into the initial project plan than it can be impossible to justify given the time available in a typical project. Time is not just required for the actual coding but needs to be taken to understand the use case, how it related to surrounding use cases, and how future use cases might affect the domain and for good documentation.

Supporting developers needs a different skill-set to other kinds of support. An increased user base means thinking about upgrade path and while a private API is simple to change, once its 'out-there' you can annoy a lot of people very quickly with the smallest change. In his keynote on designing APIs<sup>xiii</sup> Google's Joshua Bloch said "*Public APIs are forever - one chance to get it right.*"

### 7.2 Development Freeze

To be released and made usable by third-party software APIs must be frozen. This brings to an end most development activity. Thus the release of APIs can be a constraint for the development team.

Once published, any further developments need to be backwards compatible. It can be virtually impossible to manage forward/backward compatibility across a whole range of services. Version control is critical and managing the transition from beta/development API to production without breaking interfaces is paramount. Developers may find that change their data in a way which is fine for their own needs but which breaks third party use.

It could be argued that agile development does not concern itself with re-use, preferring to just develop only what is strictly necessary for the job in hand.

### 7.3 Functionality Limitations

Data or resource-centric APIs offer only partial access to data and it is likely that you will not be able provide all the functionality offered to those accessing your data more directly. It is difficult to know beforehand which functionality to prioritise which may mean that you do not always have the best set of features. It is likely that developers will struggle to decide what to expose and may only be able to provide a limited subset of their data because they are bound by an organisation's copyright or usage guidelines. This may be amplified by the 'unknown' factor of APIs. Most developers cannot predict who will be likely to use their API and on what kinds of things they will want to do with it. The unpredictability of APIs may also mean that your API takes off in an unexpected way. As is with the case with many products of the Web 2.0 era of the Web is often unclear how APIs and their related applications will be used. Many would argue that this field offers so much potential and excitement for developers because at this moment in time all possibilities still exist. When a Web API is released it is likely that their author will not know how other developers will choose to use it, if at all. This unpredictability offers great potential but also a number of issues for developers and consumers alike.

*After writing the same type of service for the same audience several times, you get a better idea of what to include.*

When maintaining software, whether or not APIs are internal or external, you have to use existing functions.

In response to the limitation of functions developers may find that they create APIs that are too open and unregulated. It is difficult to strike a balance between functionality and ease of learning and use. There are too many APIs with too narrow a theme or geographical focus and many that are too wide. The compromise between full-function specialisation and generality is tricky to get right.

A formal subsystem boundary is established, forcing developers to decide which side of that boundary any given functionality should reside. The decision isn't always correct. Developers are discouraged from arbitrarily mutating the interface, reducing innovation.

In defining an API to be generic, it is sometimes difficult to serve the complexity required for some custom applications. If accessing resources across a network, speed limitations are obviously present. Also, if accessing resources/data held by a 3rd party, processing/response time may be unacceptable for certain functions due to the way the data is organised. A 3rd party developer, or consumer of a resource will often lack the access/ability to truly have their requirements serviced.

### 7.4 Badly Written APIs

Consumers may find that badly conceived and coded APIs are more of a hindrance than a help and a rod for their developers back. As one developer put it "*a badly thought out API can end up being a straight-jacket.*" An inflexible API offers little to consumers.

One area that can often cause problems is accessibility. It can be problematic as some features of accessibility will be inherent in the API functionality whilst others are to do with the context of its use.

Non- Web based APIs continue to be mainly written by developers and often language dependant, which therefore requires developers to be familiar with that language. Once an

interface is being used, it is very difficult to change and retire resulting in bloated code. API's can sometimes be too far abstracted. This may result in an inefficient implementation to achieve a developer's desired goal.

Using an API can be slower than doing equivalent operations directly such as through a database system. For finding complex sets of data, some APIs make it very difficult to find the information you are seeking or require slow techniques such as retrieving a set of data, iterating over it and retrieving multiple other pieces of data based on the results of an initial set. These may be limitations of the way the API is designed and exposed, but many APIs seem to be unwieldy and very slow for more complex operations.

They might not fully meet the needs of the development. They might also not have appropriate triggers and flags for application development, i.e. a room booking system may deliver information about when a booking is made but not when a booking is altered.

Also it's essential to provide good documentation and feedback mechanisms for API users to get max benefits. APIs without adequate documentation are described as "*being harder to use than starting from scratch*". Documentation is sometimes limited and written in a non-accessible style the support infrastructure is often ineffective or totally absent and feedback mechanisms, and opportunities to influence improvements to the API are likewise limited.

## 7.5 Fragility

*The longer your pipe the more likely it is to break somewhere*

When designing an API a developer might ask "how long does it need to last?" Some might answer that APIs are transient and best practice is to treat them and all mashups as not being too permanent.

*It is almost as if it is becoming so easy to create things that they no longer have as much value. The theory is that you can always re-build it from scratch in 5 minutes*

This could be regarded as a view held by only a few people working only with some Web APIs. It is likely it would not be recognised by developers building more critical systems.

## 7.6 Security

*You need to think about trust between parties. Do you trust external users to use your data sensibly at their end, let alone (over)write the data on your systems?*

With open, public-facing APIs where data can be written back to the service there is always a risk that a careless, trivial or malicious API usage or simply user error may bombard the host server. Also mashups may expose more details about your data than you should be releasing to the world.

Provision of APIs requires you to think about scalability and security more than perhaps you otherwise would. Care must be taken over the exposure of sensitive data and security where the API enables modification or posting to the system. APIs allow users to misuse or abuse your systems in ways which aren't possible via a purely user interface based approach.

*An open SPARQL endpoint, if not properly handled, can be abused and be easy prey to innocent queries that unfortunately bring the whole system to a standstill. The current tactic on powerful (and therefore vulnerable) services*

*such as these is to allow access to a subset, access to a set of canned queries that have been vetted or heavily pre-calculated and cached.*

Providing security through an API may be an additional complication if some of the data is sensitive or private in some way.

## 7.7 Licence Issues

When providing an API you must manage licence provision very carefully to avoid legal liabilities, both direct and consequential. Clear licensing policies are always needed.

*Saying 'not my fault' is no good at all if you have to tie up a team of lawyers to prove that point across borders.*

Release of an API requires thought about provenance, allowed reuse, authentication, secure transport and data privacy issues at the very least. Some developers might be inclined to use an already established licence like Creative Commons, or an open source alternative.

## 7.8 Abstraction Level

Deciding upon appropriate level of abstraction, especially for Web APIs, can be tricky. A good API should be an abstraction of the useful operations of the software and it takes a number of iterations of the same kind of operations before that commonality can be identified and usefully extracted into an API. Many APIs are generally developed around a particular problem space so sometimes it is difficult to get the generic level for an API "just right". Often just one of many possible object models is implemented which may reduce uptake due to disagreement, complexity or not "just right" in its generic-ness. Provision of an API requires taking time to document and provide examples. In some senses you might argue that a program abstraction should be an API in its own right. There is a skill in making an API generic enough to be accessible without being too generic.

## 7.9 Expectations

When writing and providing APIs it's important to provide a clear vision of what can be created to your users and team. People see more traditional standards based documents and functionality as what is presented to them, however if these can be transformed, adapted and integrated (mashed up) in new ways then completely new products and services emerge. This type of development is still alien to many managers and many purchasers. They may think they get what they pay for and then pay for something else later, without realising they could have combined their original technology with others via APIs to have something more customised to their need.

## 7.10 Getting People to Use it

After releasing an API it can be difficult to get other developers to use it. This may be because people have yet to find out about it and some promotion work is required. You may find that only a limited numbers of developers may want to use it, either because you are not the leading provider in an area or because your datasets/functionality are too "niche". There is also a fear of the unknown (especially with production implementations). Developers might lack the imagination to decide upon ways to use it or be weary of the licensing implications: "I could use it but am I allowed?"

Developers may find that they need to evangelise their API and to do a great deal of advocacy.

## 7.11 Skill Level Needed

One of the main drawbacks with providing APIs is the skill level required to understand and implement an interface. Although APIs aim to remove complexity, simplifying the problem to a defined set of methods and variables, a degree of technical knowledge will generally be

assumed and required in order to exploit them. APIs must also be maintained as an additional component to a service - pricing strategies and licensing are difficult to implement.

## 7.12 Formality

Provision of APIs forces you to think before you act: you need to make good architectural and design choices and document well. This enforces a formality that may not be appropriate. For example a Service Level Agreements may be required.

## 8 Good Practice for Provision of APIs

The good practice recommendations have been provided in the separate good practice report.

## 9 Problems when using Third-Party APIs

An increasing number of HE developers are using third-party APIs and there is much to be learnt from the mistakes these APIs make.

### 9.1 Poor Documentation

The main problem developers had encountered when consuming external APIs related to documentation. Many reported bad, out of date or non-existent documentation with a lack of source code examples and no case studies. Using such APIs meant a steep learning curve. The lack of documentation was sometimes mitigated by helpful user comments (when not out of date).

Many developers reported difficulties in getting to know new large APIs. These problems were not necessarily related to how individual operations work but what the range of operations is and whether the API was suitable for their needs. Documentation could have a roll to play here.

Examples of services with bad documentation include the EPrints REST interface.

### 9.2 Poor Technology

Other issues related to technology problems such as unreliable server, transient connectivity, concerns about security and inability to access data when offline.

### 9.3 Badly Written APIs

Other issues relating more specifically to the actual APIs include poor coding practices such as bad code structure, inconsistent object modelling caused confusion. Many reported APIs serving up bad data based on incorrect implementation of standards, namespace problems and inconsistent schemas requiring code to be re-written for each API. Others had stability issues or became very large over time. Issues like unclear rate limits, accessibility and compatibility issues that prevented developers from using them.

### 9.4 Legal Issues

*What would happen if you started scraping the Disney site? Would they be running after you/your institution with the lawyers if they caught you?*

Sometimes third-party API publishers are unclear about what would happen if you used their data in a certain way.

## 10 Good Practice for Consuming APIs

The good practice recommendations have been provided in the separate good practice report.

## 11 Suggestions for Future Work

### 11.1 Support Developers

Draw upon the skills and expertise of software developers in the first instance. Get them to try out and critique services. Disseminate best practice and let developers do the rest. Developers need more events like dev8d, more opportunities to meet up and more time and space to innovate.

*We're finally getting to a point where the barrier to entry for developers to make significant progress or impact has lowered due to new techniques to retrieve, re-use and mash-up data.*

### 11.2 Support Development of APIs

New JISC Web projects should be strongly encouraged to create open APIs. Tender documents for upcoming calls should suggest that projects build the creation of APIs into bids and project plans. Projects could be offered funding specifically to create APIs because more and more projects are interlinking software from JISC projects. They could fund the development of API's and interface to cover common problems.

*A good API is the programmers equivalent of open access.*

Usability testing and the creation of proper documentation should be encouraged.

*Higher education currently doesn't have many resources to make use of something that is difficult or that isn't critical. Funding problems make many non-essential service difficult to implement. The other problem is that certain departments in the educational system have limited technical resources.*

For the CETIS conference Tony Hirst showcased a presentation<sup>xliii</sup> on JISC sponsored APIs he would be happy to experiment with in the future. He was after hackable and RESTful URLs, OpenSearch search data with clear licensing policies. He asked for course details, course syllabus/curriculum information, academics names, course reading lists and much more. For HE developers to innovate there needs to be open data and available APIs to 'play' with.

*I like the term playground, the play bit is important.*

### 11.3 Fund more Research

Appendix D at the end of this report provides some ideas for potential topics for future JISC reports. API use on the Web is still an evolving area and the learning and teaching potential is still unclear. It may possible to learn lessons from the rest of the world but further research could play a big part in HE developer's success.

## 11.4 Monitor Usage

JISC has a role to play in monitoring the usage of APIs: number of accesses, ratings and comments etc.

## 11.5 Disseminate

JISC could support the development of APIs by demystifying their use, to help non-technical managers and in particular project managers to understand what they are, how they can be used, and implications for their decision-making.

They could provide good quality information on related topics (see appendix D).

## 11.6 API Specification/Standards

*The simpler the spec, the more likely it is to be adopted. If you can find something simple, but useful, then you might be on to something. If it's powerful and extensible too, then the skies the limit.*

Standards is a tricky area and developers are divided on the role they would like JISC to take. Many recognise that there is a subtle difference between 'best practice' and 'reality' and JISC should be careful not to get drawn into enforcing standards. Some feel that JISC should concentrate on protocols while others feel that JISC have a role in developing standards.

*If an API spec is to be 'recommended' to HE, then a validator or demonstration client/server would be a real bonus. Something you can test against to make sure your end is performing as it should. e.g. the HTML/CSS/RSS validators on W3C, or Amazon's mock service.*

## 11.7 Licensing

JISC could potentially play a role in addressing the licensing issues of metadata API services from large institutions such as the British Library, and the Library of Congress.

*While the local government authority records are openly accessible and, as they are federally funded, are nominatively public domain, they retain copyright outside of the US - this is a truly untenable situation for a global and border-less system such as the internet. While we, as individual institutions, may be ignored by these places, the JISC may be in a better position to agree on our behalf, or to negotiate for a reduction in licensing restrictions.*

## 11.8 API Repository

The possible provision of an HE API repository is an idea that has been met with mixed response. Although there is currently no easy way to find out what APIs HE projects are providing something specifically for HE might not be embraced by the community. There is great potential for a registry which could gather API services in one place and facilitate their discovery and use, and perhaps the uptake of best practice but this is possibly something that needs to be developer led.

Discussion on the topic suggests that a successful approach should take the form of an evaluated list reviewed by developers working in HE. There should be summary description of capabilities and example uses. One possibility is the creation of a 'space' on Programmableweb either by submitting examples to programmeweb.com (with a tag of our choosing) or by contacting Programmableweb and asking for an education category.

The IE Demonstrator mentioned previously is a potential showcase for development activities.

## 11.9 Sustainability of Outputs of Project

The good practice outputs from this project have potential to be used in a future work. UKOLN are looking to build on this work in forthcoming project plans.

## 12 Conclusions

The research carried out has collated so many opinions, suggestions and speculations that it is difficult to encapsulate this in a brief conclusion. What is clear is that API creation and use is still an emerging area of work for Higher Education and that there are many questions that still need answering. Some of these research areas are listed in Appendix D: Potential Topics for Future JISC reports.

The question "What makes a good API?" remains unanswered. Different contexts have different characteristics, and application architectures used in one context may differ from those used in other contexts. This may mean that defining what makes an API "good" is a difficult and possibly impossible task. What may be "good" in one context isn't necessarily good in another.

However, that said, this report does offer many ways for developers to move closer towards the goal of creating such a thing as a 'good API'. These best practice techniques will hopefully become more concrete as the developer community provides their own input.

*APIs usually reflect, like any designed artefact, a bunch of compromises, i.e. functionality vs security, simplicity vs flexibility. To work in a particular domain these compromises have to be right.*

## 13 Case Studies

### 13.1 Splash, University of Sussex

As part of the JISC E-learning programme, the University of Sussex received funding to develop a system that could help provide a more personalised student learning experience by developing a user-owned, Web-based portal/mashup service that could be integrated with institutionally owned systems and gather content from external sources such as Flickr, YouTube, Facebook, and so on.

We ran evaluation tests on various 3<sup>rd</sup>-party software solutions, most notably Elgg, Mahara and Wordpress. Although these systems were found to be quite good aggregators, they did not fully meet with our requirements. Our functional requirements included the need for: profile page creation; an iGoogle-type dashboard; blog facility, messaging facility, group and friend creation, and the ability to customise templates and add metadata. We also wanted content to be created and aggregated from both internal and external sources. We decided to build in-house using the PHP-based Zend<sup>xiv</sup> Framework. This gave us a well documented, consistent and stable library of components for us to build SPLASH<sup>xiv</sup> on and allow other developers to easily extend it.

The resulting project SPLASH (Student Personal Learning and Social Homepages) is now being used at Sussex and is about to be released as open source. The system is also available to, and being used by staff at Sussex.

We had some initial issues using the Zend Framework on Mac OSX Servers. This was due to slow file traversal on the Macs when compared to other platforms such as Linux or Windows. This was worked around by using absolute paths and APC, an opt-code caching engine.

Some of the Zend Framework libraries we used to interact with external service APIs became out of sync with the services API release. An example of this was Last.fm. Whereas this did not

break SPLASH functionality, it did not allow us to take full advantage of the Last.fm API. We ended up interacting with the API directly and not through a wrapper component.

There is quite a learning curve in implementing various APIs. Some are better documented than others, some are more complicated than others. There is also the issue of keeping on top of changes to APIs – with or without using a component library to interact with that API. There is always the additional concern that if you don't keep up with an API update then it will break application functionality. Generally, the 'bigger names' that produce APIs ensure that they are always backward compatible.

We have developed a system that is extensible and very much open for further development. For the dashboard and profile pages we have created a closed API for SPLASH for the development of widgets. It is closed in the sense that it is not public, but developers at Sussex can get access to the API and implement widgets, which in turn can use APIs from both internal and external sources. Once we release as open source this will be of benefit to any other installation of SPLASH, as then widgets from one SPLASH installation could be used on another, promoting a diverse range of widgets.

From an internal point of view, creating a system that uses external APIs to aggregate content has opened the door to how we might integrate internal services by exposing APIs and allowing them to communicate more fluidly. In addition, it has also helped us think about the use of external sources and how we could use them for the benefit of the University from a learning and marketing perspective.

It is hard to say if there are any things we would do differently. Our programmer has said that he maybe needed more experimentation with the Zend Framework before taking on something this big, but still advocates the use of the framework.

The main advice we would give to other developers is if developing in house use an established framework. This is a happy medium between using an off-the-shelf system that may not work how you want and that you may not be able to change, and writing something completely from scratch.

There is a multitude of open, external APIs out there – if using for a 'mashup' experiment with a few but also get your target audience to feedback on what they think would be useful beforehand. This is also the case for integration and aggregation across internal systems.

#### **Contact details**

Tony Hudson, University of Sussex, t.hudson@sussex.ac.uk,  
<<http://www.sussex.ac.uk/splash/>>

### **13.2 The Open Source Debate: A Good API is not Enough**

We often argue about whether we can trust service X or whether we can hold our data in service Y. This usually boils down to whether the provider of the service is likely to survive, whether the data is secure and whether we can access it in an open format via a documented API. We usually fail to consider the influence we will have on the providers of the service or software exposed by the API, we therefore fail to consider our own futures.

I'm going to limit myself to thinking about Web based APIs (which we should not do, but since the UK HE and FE communities tend to think API means access to a Web service I'll do the same here). In this situation we need to consider access to the source code that implements the API. This should be an important part of our decision making process for many reasons, the most important of which is that it increases the options available.

As an example we can consider microblogging, an area that is getting a fair amount of attention in the education sector.

Twitter, probably the most popular of microblogging platforms, is Web based and provides a clearly documented and reasonably complete API. Institutions and researchers are currently considering solutions based on microblogging systems like Twitter. The focus of such efforts include questions like "will Twitter be here in 5 years?", "what is their profit model?", "will they

start charging soon?”. These are important questions because the institution will have no control over their own systems if they depend on an external service like Twitter.

For small organisations with no IT support, the use of Twitter is attractive. It is low cost, feature rich and popular. Should the model of use within Twitter change a small organisation can adapt quickly. However, for larger organisations like Universities and Colleges something like Twitter can embed itself, almost unseen, into many different systems across the organisation. For example, it could be integrated into institution wide news networks, the student VLE and and researchers VRE, the student tracking systems and the lecturer feedback mechanisms (to name just a few of the ideas I've heard). Clearly an institution can quickly become dependent on a microblogging service and thus the longevity of Twitters support, at current pricing levels, is very important to medium and long term planning. Unfortunately, there is no way of knowing where Twitter is currently heading and so planning becomes very difficult.

Some commentators argue that depending on any third party solution is too risky and inflexible. Others argue that third party solutions can provide significant cost savings with only a limited sacrifice of flexibility. What we really want is a middle ground. A solution in which we can take advantage of third party solutions for as long as the opportunity cost of doing so falls below the cost of developing our own, independent, solutions. Fortunately, there is such a “third way”, that third way is open source software.

Continuing our example of a microblogging service, an alternative to Twitter is [identi.ca](http://identi.ca)<sup>xlvi</sup>. Like Twitter, Identi.ca provides an API for accessing the system and so it can be integrated with in-house systems. It provides all of the features of Twitter and content can be bridged between the two (via the respective APIs). There is, at least on the surface, no significant difference between the two systems.

However, the Identi.ca platform is built using Laconica an open source microblogging tool. This means that if the hosted identi.ca service becomes unsuitable for any reason one can “simply” install Laconica on a suitable server and continue as before. Of course, it really isn't that simple, by moving away from the Identi.ca server you are taking on the responsibility of maintaining your own server. You may also be forcing your users to rebuild their social networks on yet another system, which presents a barrier to use. Fortunately, there is a middle ground in this case too. Since Laconica is open source, any third party could set up a company providing Identi.ca like services, including, for example, a group of collaborating universities.

It is tempting to say that an API is good if it provides the functionality we need and the ability to export our data if we decide to move on. However, as we have seen, this is not always enough, not when we need to plan strategically. Web based APIs are, in some cases, merely a way to provide the flexibility to customise systems whilst still locking you into a single provider solution. Experience has shown us that a monopoly in any domain is a dangerous thing.

We must be able to avoid lock-in to any individual provider's services. Simply being able to export data from our current service provider is not sufficient. We also need to be able to find an alternative provider. Whilst open source does not guarantee the existence of an alternative it certainly increases the chances of one being created where demand exists. This in turn puts additional pressure on other providers to satisfy the needs of their existing customer base. This pressure often manifests itself through the provision of a more complete and flexible API. This can be seen in the example of the Identi.ca API which not only implements the full Twitter API but goes much further.

#### **Contact Details**

Ross Gardler, JISC OSS (Open Source Software) Watch, University of Oxford  
[ross.gardler@oucs.ox.ac.uk](mailto:ross.gardler@oucs.ox.ac.uk)

OSS Watch <<http://www.oss-watch.ac.uk/>>

## **14 Appendix A: Good APIs survey Questions**

### **About You**

The aim of this survey is to identify best practice which should be adopted when making use of APIs (Application Programming Interfaces). The feedback will inform a report for JISC on best practices related to the development of and use of APIs in JISC's development activities and will be made freely available.

It should take you about 5 to 10 minutes to complete.

1. What type of organisation do you work for?

- HE/FE
- Research
- Commercial
- Government
- Other

Other (please specify)

2. Are you a:

- Developer
- Team Leader
- Project Manager
- Other

Other (please specify)

3. What's your email address?

### **Provision of APIs**

1. Do you provide APIs for development work?

- Yes
- No

If yes give details

2. What do you feel are the potential benefits of providing APIs?

3. What do you feel are the limitations of providing APIs?

4. What examples of good practice (with regard to providing APIs) would you recommend to others?

### **Consuming APIs**

This set of questions is intended for people who use APIs provided by any of the following: Third party services (e.g. Delicious, Twitter etc.) JISC-funded or institutional services (e.g. SHERPA/RoMEO) Other institutional systems (e.g. VLE, staff student databases)

1. Do you consume external APIs?

- Yes
- No

If yes give details

2. List any problems you have encountered when doing so?

3. What examples of good practice (with regard to consuming APIs) would you recommend to others?

#### **General Advice**

1. Give any top tips you have for providing/consuming APIs

2. Do you have any recommendations for JISC with regard to HE use of APIs?

3. Is there any information you feel you would benefit from?

Briefing papers on technical areas (APIs, Java etc.)

Guides to best practice

Top tips

Report on current practice in UK HE

Case studies

Set of criteria to use when selecting APIs to use

Lists of APIs of use to HE

Other (give scope, level of detail, length of material etc.)

4. What topics would you like to see covered in a report for JISC?

5. Would you be interested in contributing a case study for project documentation?

Yes

No

Maybe

Please provide contact details (if you haven't already)

Thank you for filling in this survey.

If you would like to contact me to discuss the subject of APIs in HE further please e-mail using [m.guy@ukoln.ac.uk](mailto:m.guy@ukoln.ac.uk)

## **15 Appendix B: Statistics from the Good APIs survey**

240 people answered the survey, 6 in the test run and 234 in the final run.

What type of organisation do you work for?

HE/FE 63.2%

Research 10.7%

Commercial 12.0%

Government 4.7%

Other 9.4%

Are you a:	
Developer	36.3%
Team Leader	15.4%
Project Manager	17.5%
Other	30.8%

Is there any information you feel you would benefit from?

Briefing papers on technical areas (APIs, Java etc.)	43.2%
Guides to best practice	75.0% <sup>33</sup>
Top tips	61.4% <sup>27</sup>
Report on current practice in UK HE	61.4% <sup>27</sup>
Case studies	65.9% <sup>29</sup>
Set of criteria to use when selecting APIs to use	54.5% <sup>24</sup>
Lists of APIs of use to HE	81.8% <sup>36</sup>

## 16 Appendix C: People Consulted during the Study

UKOLN Systems Team

Pete Johnston

Ian Ibbotson

Wilbert Kraan

Tony Hirst

Sam Easterby-Smith

Phil Wilson

Dave Flanders

And many others...

## 17 Appendix D: Potential Topics for future JISC reports

When asked what documentation or potential reports JISC could support in the area of APIs developers offered many ideas.

These include

### 17.1 Case Studies

There were mixed feeling about case studies – some people find them really useful while others don't.

Lessons learned re the provision or use of APIs, especially any with experience over a period of time.

### 17.2 Lists

Of the popular APIs in use in HE (registry service?)

Of APIs of use to HE and why they are of use.

Of existing services and APIs relevant for people undertaking project development work in a particular context (e.g. VLE or VRE).

Of APIs provided by JISC services.

Of examples where API's have changed.

Of success mash-ups and real-world case studies.

Of useful tools (e.g. Yahoo Pipes, Google Spreadsheets, etc.) Give a feature list. Contextualised for e-learning., library, etc.

### **17.3 Guides**

Examples of best practice.

Define the criteria to use when selecting instances of the effective use of third party machine interfaces by the HE community.

To technical areas with as little jargon and small a use of abbreviations as possible. Lots of diagrams.

Technologies - what they are and what each is best suited for.

Definitions of technical terms including API.

How to design in a way that encourages users to use software.

Methods of improving the accessibility and reliability of APIs.

Methods of designing APIs and the underlying services which increase resilience, efficiency and flexibility.

Sustainability/exit strategy issues.

Frameworks, technologies, legal implications, document templates, open source approaches, development methodologies.

### **17.4 Surveys**

Surveys of use of languages for API programming.

Extensive survey of existing assessment systems and their APIs (if any).

A survey of the main types of feature normally provided in APIs.

### **17.5 Statistical Data**

How much re-use of JISC research project output is actually achieved?

How much of that is use of components via their APIs?

How amenable are these JISC-funded components to such use?

Amazon type ratings and access numbers available when API is accessed.

### **17.6 Standards Information**

Adherence to standards.

Accessibility of APIs.

### **17.7 Discussion**

Some discussion of how API providers engage with API users and determine what these APIs should provide, gauge likely demand, etc.

## 17.8 Toolkits

Establish a set of criteria to assist in the evaluation of APIs, such as currency, resourcing, community size, support structures, release management, etc.

## 17.9 The Bigger Picture

Non-technical view of future developments and where there may be a consensus view for UK HE/FE.

What commercial software vendors development plans are and how we can all influence the strategy and direction.

Some of this data might be useful in the form of a regular accessible bulletin/blog.

# 18 Appendix E: Definitions

## RSS

RSS or Really Simple Syndication is a set of standardized formats that are used in publishing frequently updated works for example; blog entries, news headlines, audio and video feeds. The RSS format is written in XML and is called a “feed”, users are able to “subscribe” to RSS feeds using what is know as a feed reader. RSS feeds can contain; full or summarised text and/or metadata for example author, publication date, title, URL link.

## ATOM

Atom is a pair of related standards, the Atom Syndication Format and the Atom Publishing Protocol (AtomPub). The Atom Syndication Format is an XML based web feed format like RSS, users are able to subscribe to Atom feeds in the same way as RSS. Atom has a richer set of features for programmers to take advantage of than RSS. AtomPub is as the name suggests a protocol for creating and updating web resources, such as blogs.

## OAI

The Open Archives Initiative’s goal is to develop and promote interoperability frameworks for institutional repositories and digital archives. They currently have two frameworks, OAI-PMH and OAI-ORE

## OAI-PMH

Open Archives Initiative Protocol for Metadata Harvesting is a protocol used to harvest metadata descriptions of records from institutional repositories and digital archives. The protocol uses XML over HTTP to achieve this.

## OAI-ORE

Open Archives Initiative Object Reuse and Exchange is a standard used for the description and exchange of aggregations of Web resources. It achieves this by using Resource Maps (ReMs) these can be represented in several different formats including; Atom feeds, RDF/XML and RDFa.

## Open API

Open API (or OpenAPI) mean an open application programming interfaces (a set of protocols or libraries) that people can use to write their own program to present or interact with data or services provided by the website/web service that has an Open API.

## REST

Representational state transfer is a style of software architecture which has the notion of a resources and identifiers. Any information that can be named such as: a document, image, a virtual object, etc is a resource and is referenced with a global identifier such as a URI. The World Wide Web is a key example of a RESTful design.

## RESTful API

For an API to be considered RESTful it needs to for fill the following actions; Create, Read, Update and Delete. These terms are also know as CRUD. The term “accidentally RESTful” is applied to APIs that only for fill part of the CRUD actions.

## HTTP

Hypertext Transfer Protocol is a transfer protocol for distributed resources. It is based on a request/response standard whereby a client requests data from a server. Such as when a user “requests” web site from a web server the server responds by “sending” back the requested web page.

## API Libraries

An application programming interface is a set of protocols or libraries that can be both Language-dependent and Language-independent provided by software, web services, and operating systems. To support developers in creating both desktop and Web applications.

## JSON

JavaScript Object Notation, is a text-based, human-readable format for representing simple data structures. The JSON format is mainly used in AJAX programming as an alternative to using the XML format.

## XML-RPC

XML Remote Procedure Call is a lightweight protocol which uses XML to encode its calls to an application across a network or the internet to fetch back information.

## 19 Author Contact Details

Marieke Guy

Research Officer

UKOLN, University of Bath, BA2 7AY

Web site: <http://www.ukoln.ac.uk/>

Email: [M.Guy@ukoln.ac.uk](mailto:M.Guy@ukoln.ac.uk)

Phone: 01225 703928

## 20 References

URLs appearing in this report were last accessed on Friday 6<sup>th</sup> March 2009.

---

i Creative Commons Attribution-Non-Commercial 2.0 UK: England & Wales  
<http://creativecommons.org/licenses/by-nc/2.0/uk/>

ii Wikipedia – APIs  
<http://en.wikipedia.org/wiki/API>

iii Delicious bookmarks  
<http://delicious.com/mariekeguy/good-apis-jisc>

iv <http://www.surveymonkey.com/>

v [http://www.surveymonkey.com/s.aspx?sm=yGugtLrayEiRLnGYzIUQ4w\\_3d\\_3d](http://www.surveymonkey.com/s.aspx?sm=yGugtLrayEiRLnGYzIUQ4w_3d_3d)

vi CETIS Conference Wiki  
[http://wiki.cetis.ac.uk/Conference\\_2008\\_Programme](http://wiki.cetis.ac.uk/Conference_2008_Programme)

vii Wiki used at CETIS conference for Innovation in a World on Web APIs  
<http://cetis2008-apis.wetpaint.com/?t=anon&t=anon>

- 
- viii Technological Innovation in a World of Web APIs  
<http://www.ukoln.ac.uk/web-focus/events/conferences/cetis-2008/>
- ix Dev8D Web site  
<http://www.dev8d.org/>
- x Dev8d blog: Five minute Interview; Paul Walk  
<http://dev8d.jiscinvolve.org/2009/02/10/five-minute-interview-paul-walk/>
- xi Dev8d blog  
<http://dev8d.jiscinvolve.org/>
- xii Dev8d Twitter channel  
<http://twitter.com/dev8D>
- xiii Good APIs blog:  
<http://blogs.ukoln.ac.uk/good-apis-jisc/>
- xiv Writetoreply.org  
<http://writetoreply.org/>
- xv Yahoo Pipes  
<http://pipes.yahoo.com/pipes/>
- xvi OpenDOAR API  
<http://www.opendoar.org/tools/api.html>
- xvii OpenDOAR Prototype Protocol for Statistical Harvesting  
Description at [http://www.opendoar.org/demos/psh\\_prototype.php](http://www.opendoar.org/demos/psh_prototype.php)
- xviii SHERPA/RoMEO Prototype API  
<http://www.sherpa.ac.uk/romeo/api.html>
- xix WWWOPAC  
[http://downloads.adlibsoft.com/uk/documentation/WWWOPAC%20reference%20guide\\_A5.doc](http://downloads.adlibsoft.com/uk/documentation/WWWOPAC%20reference%20guide_A5.doc)
- xxMuseum of London  
<http://www.museumoflondon.org.uk/MuseumOfLondon/food/rest.aspx>
- xxi arXiv  
<http://arxiv.org/api>
- xxii Oxford University Research Archive (ORA)  
<http://ora.ouls.ox.ac.uk>
- xxiii GeoCrossWalk  
<http://www.geoxwalk.ac.uk/>
- xxiv Celtic coin index for Oxford University  
<http://www.finds.org.uk/cci/blog>
- xxv PROD  
<http://prod.cetis.ac.uk/>
- xxvi JISC funded WebPA  
<http://webpaproject.lboro.ac.uk/>
- xxvii Promethean Planet  
<http://www.prometheanplanet.com/pdn>
- xxviii Group Manager API, Bath University  
<http://wiki.bath.ac.uk/display/bucswdev/LDAP+Group+Management+API>
- xxix Deposit API  
[http://www.ukoln.ac.uk/repositories/digirep/index/Deposit\\_API](http://www.ukoln.ac.uk/repositories/digirep/index/Deposit_API)

---

xxx Splash  
<http://splash.sussex.ac.uk>

xxxi Joe Cutting  
<http://www.joecutting.com/newsquiz.asp>

xxxii Programmable Web  
<http://www.programmableweb.com/>

xxxiii Webmashup,  
[http://www.webmashup.com/Mashup\\_APis/index.php](http://www.webmashup.com/Mashup_APis/index.php)

xxxiv WebAPI Directory,  
<http://www.webapi.org/webapi-directory/>

xxxv Library Application Program Interfaces (APIs), TechEssence.info, Roy Tenant, 17 July 2008  
<http://techessence.info/apis/>

xxxvi Mashed Library blog  
<http://mashedlibrary.ning.com/forum/topic/show?id=2186716%3ATopic%3A9>

xxxvii Museums and Computers Group  
MCG@JISCMail.AC.UK

xxxviii Brooklyn Museum blog:  
<http://www.brooklynmuseum.org/community/blogosphere/bloggers/2009/03/04/brooklyn-museum-collection-api/>

xxxix <http://www.dataportability.org/>

xl Hibernate  
<http://www.hibernate.org/>

xli GregorianCalendar  
<http://java.sun.com/j2se/1.5.0/docs/api/java/util/GregorianCalendar.html>

xlii How to Design a Good API and why it matters  
<http://www.slideshare.net/guestbe92f4/how-to-design-a-good-a-p-i-and-why-it-matters-g-o-o-g-l-e/>

xliii What I'd like from JISC APIs  
<http://www.slideshare.net/psychemedia/cetis-conf-what-id-like-from-jisc-apis-presentation>

xliv Zend  
<http://framework.zend.com/>

xlvi SPLASH  
<http://www.sussex.ac.uk/splash>

xlvi identi.ca  
<http://identi.ca/>