HOW DATA SHOULD BE MADE PUBLICLY AVAILABLE

RESOURCES COLLECTED BY THE SUNLIGHT FOUNDATION

WHY EVERYONE SHOULD KNOW WHAT MAKES A GOOD DATA SET; IT'S NOT AS HARD AS YOU THINK

In many offices, when technology questions arise, the answer is to reflexively trust the technologists. These are often the folks who link to Venn diagrams of the fine distinctions between nerds, geeks, and dweebs; who prefer the comic xkcd to the Far Side; and who trust slashdot over NPR. So when it comes to the question of how the government should make information available online — in particular, how data should be made available online — most people's first inclination is to nod to the technologist and slowly back away. That disengagement is a mistake.

How information is made available online fundamentally controls what can be done with it. Fortunately, an intelligent layperson can understand how structure makes data usable. That's important, as the intelligent layperson is likely the one writing the rules on how government data will be made available: whether as a congressional staffer, a federal agency employee, or a citizen making a request. Awareness about data structure encourages smarter specs, and the ability to get more out of your information.

The following are a series of articles drawn from the Sunlight Foundation, Sunlight Labs, and Princeton's Freedom to Tinker Blog that discuss how data should be made publicly available to assure a high level of data quality.

The articles include below are:

- "A Study in Transparency: The Open Government Directive, the Department of Labor, and the Open Data Principles," Sunlight Foundation Blog (4/12/2010)
- "Government Datasets That Facilitate Innovation," Freedom to Tinker Blog (3/1/2010)
- "Basic Data Format Lessons," Freedom to Tinker Blog (3/2/2010)
- "Labeling Dataset Contents," Freedom to Tinker Blog (3/3/2010)
- "Correcting Errors and Making Changes," Freedom to Tinker Blog (3/8/2010)
- "Best Practices for Government Datasets: Wrap-up," Freedom to Tinker Blog (3/12/2010)
- "Drafting Guidelines for Government Data Catalogs," Sunlight Labs Blog (3/29/2010)

Links to all of these articles are available from here: http://bit.ly/9alnLq

A STUDY IN TRANSPARENCY: THE OPEN GOVERNMENT DIRECTIVE, THE DEPARTMENT OF LABOR, AND THE OPEN DATA PRINCIPLES

By Daniel Schuman on 04/12/10

Cabinet agencies (and others) released their Open Government Plans last week with much fanfare, mixed reviews, and many promises for the future. I want to focus on one initiative — the Department of Labor's "Online Enforcement Database" — to highlight the strengths and weakness of what we've seen, and suggest some guidelines for going forward.

Online Database Strengths and Weaknesses

With the explosion at a mine in West Virginia last week, many questions are being asked about federal safety inspections. My colleague Anu Narayanswamy wrote on Monday, before the Online Enforcement Database was released, that the way the federal government releases data on mine safety makes it impossible to see how safety violations at one mine stack up against others. You cannot tell if the 500 safety violations in 2009 at this particular mine, for example, are typical for this industry.

On Wednesday, the Labor Department released the Online Enforcement Database, which contains five major data sets, including one on mine safety. Anu's follow-up article on Friday explained that "with mine safety data, released for the the first time in bulk [on Wednesday], users can search for mine inspection data by state or even zip code." But she also reported the data sets are only in a partially downloadable format, and do not include "the kinds of violation and penalties levied on mines across the country." In other words, it's difficult to figure out what's going on.

It is the search results, and not the underlying database, that are downloadable in bulk. ("Bulk" access means that you can download all of the information at once, and not piecemeal.) The only way to get at the Enforcement Database's information is to use its search tool, which has very limited capabilities. Users may search by state, agency, zip code, and by industry code. (DOL deserves credit for including the industry codes in a link from the search page.) So, a user cannot narrow the search range to a county, or a congressional district, or by the owner of a facility. Compare this to the search tool used at transparencydata.com, a new initiative from Sunlight that allows users to search a database on campaign contributions, that allows searching, sorting, and downloading in a multiplicity of ways.

As mentioned before, the Online Enforcement Database itself is not available for download in bulk. There's no way to look at all of the information the Labor Department has painstakingly gathered. And despite the wealth of information, a clunky search tool adds to the frustration. Without access to the supporting data, researchers cannot answer many questions. In fairness, the Labor Department says that bulk access and improved search tools are "coming soon," but it would be very helpful to have a date to accompany this promise. Doing so would make the promise concrete and testable.

I do not mean to pick on the Department of Labor, which made an effort in its Open Government Plan [PDF] to identify datasets for online publication and to set deadlines. Indeed, they stated they plan to take all data they collect and make it publicly available online and in downloadable formats, with appropriate caveats. Many agencies fell far short of DOL's achievements. But DOL should go further.

Open Data Principles

Elsewhere I've pulled together resources (from Princeton and Sunlight Labs) on building good data sets, including drafting guidelines for government data catalogs. It's important focus, however, at the fundamental

level of what it means when we talk about how government should publish data online, a.k.a. “open data principles.” As an attorney, I’m hardly qualified to talk about this, so I am fortunate that much of the heavy lifting was done at a conference in 2007. Afterward, my colleagues Clay and John and I worked on revising the open data principles, nine in number, and fleshed out a rough evaluation of when they are satisfied.

When agencies think about how to make information available, they should look to these (draft) principles. They state, in short, that data should be: complete, primary, timely, accessible, machine processable, non discriminatory, non propriety, license free, and permanent. Resource to these principles by the agencies — and a better effort to comply with the directive’s requirement to identify all high-value data sets and set deadlines for online publication — would have turned the thus-far mixed results of the Open Government Directive into an unqualified success. There is still time to make that promise into a reality. Here are the 9 open data principles in a framework to evaluate the extent to which they are satisfied:

Open Data Principle	Awful execution	Poor execution	Satisfactory execution	Good execution	Great execution
Complete	selectively disclosed portions, complete scope of data unknown		bare-bones Excel spreadsheet	Source material provided with formulas for derivative data	Source Material Provided with Metadata, Aggregate data provided with formulas for their creation, data documentation available.
Primary	summary of aggregate statistics	Aggregate statistics	bare-bones Excel spreadsheet	data w/ collection methods documented	data with collection methods documented, source documents provided
Timely	information released only after it has become inert or irrelevant. e.g. released one year after collection	released one month after collection	released one week after collection	released one day after collection	information disclosed as it is collected. Given control over collection, info also collected at most effective frequent interval.
Accessible	Paper	FOIA-provided; behind search forms	data format supports analysis and reuse	Available through bulk access protocols such as FTP and Rsync with sufficient bandwidth to allow demand to be met, as well as available through a well-documented API with good performance.	Available through bulk access protocols with sufficient bandwidth and API functionality, alongside links and pointers to outside sources.
Machine Processable	Paper	PDF, Scanned Images	.csv, tab delimited data	Documented API coupled with .csv, tab delimited data	json or XML data dumps, well documented coupled with a well documented API
Non Discriminatory	In-person visit Neccessary to View Data, Data released selectively to specific parties	Registration required to view data			No registration required to view or download data
Non-proprietary	Undocumented, proprietary format.	Fairly well-known proprietary format, such as Microsoft Access.	Format based on an open standard but with limited independent implementations. For example PDF, Semantic Web.	Format based on Open Standard with multiple *different* implementations, for instance, CSV.	Format based on an open standard with multiple, independent implementations that use the format. For example: HTML, XML, JSON.
License Free	Pay-for-use, or most restrictive TOS or EULA, with possibly unlawful restrictions	Use with terms-of-service, citation requirements, non-commercial-use requirements	No license specified. Terms of use as given in law (e.g. FEC data).	Display of legislative terms of use in clear fashion alongside data	Clearly labeled public domain, work of the government.
Permanent	Subject to indiscriminate or malicious deletion. no guarantee of permanence, information fully open to manipulation and removal. Non-Digital	Current data available but no archive. (E.g., a webcast stream but no file is an example)	Archived for the term of the current Administration	Plan in place for indefinite archival	strong archival standards, frequent archiving, versioning, archives available on web

GOVERNMENT DATASETS THAT FACILITATE INNOVATION

BY JOE CALANDRINO - POSTED ON MARCH 1ST, 2010 AT 4:41 PM

There's a growing consensus that the government can increase its openness and transparency by publishing its raw data in bulk online. As several Freedom to Tinker contributors argued in Government Data and the Invisible Hand, publishing data empowers third party software developers to produce innovative new technologies that engage citizens and illuminate government's inner workings. With the establishment of Data.gov and the federal Open Government Initiative, federal agencies are quickly embracing a culture of machine-readable data release, and many states and municipalities are now following their lead.

But how usable are these datasets for developers? The answer lies primarily in the structure and contents of the datasets themselves. While all data in digital form is technically machine-readable in some sense, the ease of use for machine-readable datasets can vary widely. In fact, machine-readability is just a baseline requirement: a developer can't start to work with a dataset until it's in this form. Once that minimum standard is met, the critical factor is how easy it is for developers to use the dataset in new, innovative ways.

In this series of posts, we'll draw on our experience building applications that use government data to offer some thoughts about best practices government could follow in releasing data. By taking a few straightforward steps in preparing its datasets, government can make the data much more useful to developers.

One key factor in determining ease of use for developers is the structure of the dataset, and that is the topic of our first post. Let's start with a trivial example:

<BOOK>A Tale of Two Cities by Charles Dickens. Chapter 1. The Period. It was the best of times, it was the worst of times [...] The end.</BOOK>

This is a "well-formed" XML version of Dicken's "A Tale of Two Cities" in its entirety. Though more usable than a PDF copy of the book, the XML document lacks basic structure and is not particularly helpful to a developer building tools to display or analyze the book. Compare that to:

```
<BOOK>
  <HEADER>
    <TITLE>A Tale of Two Cities</TITLE>
    <AUTHOR>Charles Dickens</AUTHOR>
  </HEADER>
  <BODY>
    <CHAPTER NUMBER="1">
      <TITLE>The Period</TITLE>
      <PARAGRAPH NUMBER="1">
        <SENTENCE NUMBER="1">It was the best of times [...]</SENTENCE>
      </PARAGRAPH>
      [...]
    </CHAPTER>
    [...]
  </BODY>
</BOOK>
```

This data is far more structured, and a developer can take it and immediately do lots of new things. If the developer plans to build an interface for a new e-book reader for instance, it's easy to extract the component

parts of the book for appropriate formatting. With the less-structured version, the developer needs to guess where chapters, titles, and paragraphs begin and end. Because manual analysis is infeasible for large, complex datasets, developers who have only minimally-structured data will need to build automated processing scripts to make these guesses. Developing these scripts can be difficult and time-consuming, and data quality will suffer because the scripts will inevitably make mistakes.

Whether a dataset facilitates innovative uses by developers is not a yes or no question but a matter of degree, and it depends largely on the quality of the data's structure and the needs of specific developers. In deciding what structure to add, agencies should consider who is in the best position to add various types of structure to the data. Sometimes, the agency is in the best position. Employees of an agency may amass specialized knowledge about the data, or the agency may already internally store the data with structural details like explicit database columns. In these cases, the agency can provide this structure with little effort, relieving developers from the potentially Herculean task of reconstructing these details. In other cases, the agency may have no significant advantage over private parties.

Agencies should get as close to this dividing line as is reasonably possible to broaden the range of creative possibilities for application developers. The goal is to minimize structural obstacles that might prevent developers from tinkering with the data. Better structure leads to more innovative tools, a more transparent government, and a greater appreciation for the work done by federal agencies.

Over our next several posts, we'll discuss choices that agencies make when releasing datasets and the ways these choices affect developers. Among other things, we'll explore basic data format lessons, data labeling, and correction/modification of datasets. Our goal is to turn this series into a best practices white paper for government use, and we'd appreciate any comments, suggestions, or insights from readers.

BASIC DATA FORMAT LESSONS
BY JOE CALANDRINO - POSTED ON MARCH 2ND, 2010 AT 7:45 AM

When creating a dataset, the preferences of developers may not be obvious to those producing the dataset. Seemingly innocuous choices by data providers can lead to major headaches for developers. In this post, we discuss some of the more basic challenges that developers encounter when working with a dataset. These lessons may seem trivial to our more technical readers, but they're often learned through experience. Our hope is to reduce this learning curve by explaining how various practices affect developers. We'll focus on XML datasets, but many of the topics apply to CSV and other data formats.

One of the hardest parts of working with a dataset can be figuring out what's in it and how it's organized. What data comes inside an "<FL47>" tag? Can a "<TEXT>" element ever contain a "<PARAGRAPH>" element? Developers rely heavily on documentation to explain the structure and contents of a dataset. When working with XML, one particularly relevant item is known as a schema. An XML schema is a separate file with an extension such as ".dtd" or ".xsd," and it provides a blueprint of the permitted structure for corresponding XML files. XML schema files tell developers where they can recover the information that they need from a dataset. These schema files and other documentation are often a necessity for developers, and they should be treated as such by data providers. Any XML file supplied by an agency should contain a complete URL address at which its schema can be found. Further, any link to an XML document on a government site should have prominent links near it for the corresponding schema file and reasonable documentation describing the contents of the dataset.

XML schema files can be seen as an informal contract between data providers and developers, effectively promising that a dataset will match the specified structure. Unfortunately, sometimes datasets contain flaws causing them not to match that structure. Although experienced developers produce software that detects the existence of structural errors, these errors can be difficult or impossible for them to isolate and correct. The people in the best position to catch and fix structural errors are the people producing a dataset. Numerous validation tools exist for ensuring that an XML document is well-formed and valid—that is, the document is structurally sound and matches its XML schema. Prior to releasing a dataset, an agency should run a validator on it to check for structural flaws. This sanity check can take just a few moments for an agency but save hours of developer time.

When deciding on the structure of a dataset, an agency should strive for simplicity while logically representing the underlying data. The addition of elements, attributes, or children in a schema can improve the quality and clarity of the dataset, but it can also add unnecessary complexity. When designing schemas, there's a tendency to include elements or other structure that will almost certainly go unused in practice. Schema designers may assume that extraneous items do no harm, but developers must cautiously account for them if allowed by schema. The result can be wasted developer time and increased software complexity. The true cost of various structural choices is not just the time necessary to encode these choices in a schema but also the burden these choices impose on developers. Additional structural complexity must provide a justifiable benefit.

In some cases, however, the addition of elements or attributes is not only justifiable but highly desirable for developers: logically distinct pieces of data should appear in separate XML elements or attributes. Suppose that a developer wishes to access a piece of data in a dataset. If the data is combined with other information, the developer will need to figure out how to extract it from the combined field. This extraction can be difficult, time-consuming, and prone to errors. For example, assume that a data provider includes the following element:

<DOCINFO>Doc No. 2001345--Released 01-01-2001</DOCINFO>

To extract the document number, a developer might look for all characters following "No." but before a dash. While this is straightforward enough, other parts of the same or future datasets might instead use the document number format "2001-345" or separate the document number and release date with a space rather than a double-dash. Neither case would lead to invalid XML, but both would break the developer's extraction tool. Now consider this alternative:

```
<DOCINFO>
  <DOC_NO>2001345</DOC_NO>
  <RELEASE_DATE>01-01-2001</RELEASE_DATE>
</DOCINFO>
```

Using extra elements to separate logically distinct data can prevent extraction errors. This lesson often applies even when the combined data is related. For example, the version number 5.3.2 could be broken into major version 5, minor version 3, and revision 2. In general, agencies should separate such items themselves when they can do so more easily than developers.

Even when the basic structure of a dataset is ideal, choices about how to provide data inside this structure can affect developers. Developers thrive on consistency. Suppose that a dataset details various costs. Consider all possible ways of writing cost: $4,300, 5938.37, 74 dollars and 63 cents, etc. Unless an agency decides on, documents, and adheres to a standard format, developers' software must handle a large number of possibilities to avoid unexpected surprises. Consistency in a dataset can make a developer's life far easier, and it reduces the possibility that surprises will break an application. Note that a schema can be helpful for enforcing consistency for certain fields—for example, cost might be defined as a decimal field with a constraint on the number of fractional digits.

Redundant information is another source of difficulty for developers. Redundancy can appear in numerous ways. Suppose that a dataset contains the element "<VERSION>Version 5</VERSION>." The word "Version" is unnecessary, and developers must go through additional trouble to extract the version number. In so doing, developers must consider the possibility that "Version" could be misspelled, abbreviated, or omitted. Supplying a version number alone ("<VERSION>5</VERSION>") would avoid this issue altogether. More subtly, suppose that a dataset contains all bills introduced in Congress on a certain date:

```
<INTRODUCED_BILLS>
  <DATE>11-12-2014</DATE>
  <HOUSE_BILLS DATE="NOV 12, 2014">
    [...]
  </HOUSE_BILLS>
  <SENATE_BILLS DATE="NOV 12, 2014">
    [...]
  </SENATE_BILLS>
</INTRODUCED_BILLS>
```

Date information appears three times even though it must be the same in all cases. The more often a piece of information appears in a dataset, the more likely that inconsistencies will occur. These inconsistencies can lead to software errors requiring manual resolution. While redundancy can serve as a sanity check for errors, agencies typically should perform this check themselves if possible before releasing the data. After all, the agency is in the best position to fix inconsistencies. Unless well-justified, agencies should avoid redundancy.

Processing datasets often requires a significant amount of developer time, so adherence to even basic rules can dramatically increase innovation. What other low-level recommendations do FTT readers have for non-developers producing datasets?

Tomorrow, we'll discuss how labeling elements in a dataset can help developers.

BASIC DATA FORMAT LESSONS

BY JOE CALANDRINO - POSTED ON MARCH 2ND, 2010 AT 7:45 AM

When creating a dataset, the preferences of developers may not be obvious to those producing the dataset. Seemingly innocuous choices by data providers can lead to major headaches for developers. In this post, we discuss some of the more basic challenges that developers encounter when working with a dataset. These lessons may seem trivial to our more technical readers, but they're often learned through experience. Our hope is to reduce this learning curve by explaining how various practices affect developers. We'll focus on XML datasets, but many of the topics apply to CSV and other data formats.

One of the hardest parts of working with a dataset can be figuring out what's in it and how it's organized. What data comes inside an "<FL47>" tag? Can a "<TEXT>" element ever contain a "<PARAGRAPH>" element? Developers rely heavily on documentation to explain the structure and contents of a dataset. When working with XML, one particularly relevant item is known as a schema. An XML schema is a separate file with an extension such as ".dtd" or ".xsd," and it provides a blueprint of the permitted structure for corresponding XML files. XML schema files tell developers where they can recover the information that they need from a dataset. These schema files and other documentation are often a necessity for developers, and they should be treated as such by data providers. Any XML file supplied by an agency should contain a complete URL address at which its schema can be found. Further, any link to an XML document on a government site should have prominent links near it for the corresponding schema file and reasonable documentation describing the contents of the dataset.

XML schema files can be seen as an informal contract between data providers and developers, effectively promising that a dataset will match the specified structure. Unfortunately, sometimes datasets contain flaws causing them not to match that structure. Although experienced developers produce software that detects the existence of structural errors, these errors can be difficult or impossible for them to isolate and correct. The people in the best position to catch and fix structural errors are the people producing a dataset. Numerous validation tools exist for ensuring that an XML document is well-formed and valid—that is, the document is structurally sound and matches its XML schema. Prior to releasing a dataset, an agency should run a validator on it to check for structural flaws. This sanity check can take just a few moments for an agency but save hours of developer time.

When deciding on the structure of a dataset, an agency should strive for simplicity while logically representing the underlying data. The addition of elements, attributes, or children in a schema can improve the quality and clarity of the dataset, but it can also add unnecessary complexity. When designing schemas, there's a tendency to include elements or other structure that will almost certainly go unused in practice. Schema designers may assume that extraneous items do no harm, but developers must cautiously account for them if allowed by schema. The result can be wasted developer time and increased software complexity. The true cost of various structural choices is not just the time necessary to encode these choices in a schema but also the burden these choices impose on developers. Additional structural complexity must provide a justifiable benefit.

In some cases, however, the addition of elements or attributes is not only justifiable but highly desirable for developers: logically distinct pieces of data should appear in separate XML elements or attributes. Suppose that a developer wishes to access a piece of data in a dataset. If the data is combined with other information, the developer will need to figure out how to extract it from the combined field. This extraction can be difficult, time-consuming, and prone to errors. For example, assume that a data provider includes the following element:

<DOCINFO>Doc No. 2001345--Released 01-01-2001</DOCINFO>

To extract the document number, a developer might look for all characters following "No." but before a dash. While this is straightforward enough, other parts of the same or future datasets might instead use the document number format "2001-345" or separate the document number and release date with a space rather than a double-dash. Neither case would lead to invalid XML, but both would break the developer's extraction tool. Now consider this alternative:

```
<DOCINFO>
  <DOC_NO>2001345</DOC_NO>
  <RELEASE_DATE>01-01-2001</RELEASE_DATE>
</DOCINFO>
```

Using extra elements to separate logically distinct data can prevent extraction errors. This lesson often applies even when the combined data is related. For example, the version number 5.3.2 could be broken into major version 5, minor version 3, and revision 2. In general, agencies should separate such items themselves when they can do so more easily than developers.

Even when the basic structure of a dataset is ideal, choices about how to provide data inside this structure can affect developers. Developers thrive on consistency. Suppose that a dataset details various costs. Consider all possible ways of writing cost: $4,300, 5938.37, 74 dollars and 63 cents, etc. Unless an agency decides on, documents, and adheres to a standard format, developers' software must handle a large number of possibilities to avoid unexpected surprises. Consistency in a dataset can make a developer's life far easier, and it reduces the possibility that surprises will break an application. Note that a schema can be helpful for enforcing consistency for certain fields—for example, cost might be defined as a decimal field with a constraint on the number of fractional digits.

Redundant information is another source of difficulty for developers. Redundancy can appear in numerous ways. Suppose that a dataset contains the element "<VERSION>Version 5</VERSION>." The word "Version" is unnecessary, and developers must go through additional trouble to extract the version number. In so doing, developers must consider the possibility that "Version" could be misspelled, abbreviated, or omitted. Supplying a version number alone ("<VERSION>5</VERSION>") would avoid this issue altogether. More subtly, suppose that a dataset contains all bills introduced in Congress on a certain date:

```
<INTRODUCED_BILLS>
  <DATE>11-12-2014</DATE>
  <HOUSE_BILLS DATE="NOV 12, 2014">
    [...]
  </HOUSE_BILLS>
  <SENATE_BILLS DATE="NOV 12, 2014">
    [...]
  </SENATE_BILLS>
</INTRODUCED_BILLS>
```

Date information appears three times even though it must be the same in all cases. The more often a piece of information appears in a dataset, the more likely that inconsistencies will occur. These inconsistencies can lead to software errors requiring manual resolution. While redundancy can serve as a sanity check for errors, agencies typically should perform this check themselves if possible before releasing the data. After all, the agency is in the best position to fix inconsistencies. Unless well-justified, agencies should avoid redundancy.

Processing datasets often requires a significant amount of developer time, so adherence to even basic rules can dramatically increase innovation. What other low-level recommendations do FTT readers have for non-developers producing datasets?

Tomorrow, we'll discuss how labeling elements in a dataset can help developers.

CORRECTING ERRORS AND MAKING CHANGES

BY JOE CALANDRINO - POSTED ON MARCH 8TH, 2010 AT 8:45 AM

Even cautiously edited datasets sometimes contain errors, and even meticulously produced schemas require refinement as circumstances change. While errors or changes create inconvenience for developers, most developers appreciate and prepare for their inevitability. Agencies should strive to do the same. A well-developed strategy for fixes and changes can ease their burden on both developers and agencies.

When agencies release data, developers ideally will interact with it in creative new ways. Given datasets containing megabytes to gigabytes of data, novel uses will reveal previously unnoticed errors. Knowledge of these errors benefits the agency as well as other developers using the data, so agencies should take steps to encourage error reporting. Labels in a dataset allow developers to specify errors efficiently and unambiguously. An easy-to-find channel for reporting errors, such as a prominently provided email address or web form, is also critical. Tracking down the contact information of the person responsible for a dataset can be difficult, and a well-known channel reduces this barrier to feedback.

Upon learning of an issue in a dataset, an agency should correct the problem and release the corrected dataset in a timely manner. An important fact to keep in mind when correcting data is that numerous developers may have already downloaded and begun using the old flawed version. For these developers, even a minor modification can cause major issues if not done carefully. Agencies should think about two things: how they will make developers aware that the dataset has been modified and how they will change the dataset itself. The first point is sometimes ignored in spite of its importance. Not only should datasets contain version information, but agencies should also notify developers when the data that they rely on has changed. In particular, agencies should allow developers to subscribe to an email list or an RSS feed for specific datasets that details updates in a well-structured manner. These updates should clearly specify the dataset and version affected, a location where the updated dataset can be found, and a description of the changes to the dataset. When possible, these changes should be specified via a formal, structured description—for example, a diff output—as well as a brief prose explanation.

Correction of dataset contents should proceed cautiously. Suppose that an application allows user to comment on parts of a document. If labels are in a dataset are not maintained consistently across versions, the developer may need to painstakingly map comments from the old data to the corresponding parts of the new dataset. Issues like this can be mitigated through several practices. First, an agency should seek to preserve labels across versions of a dataset when possible (alternatively, in some cases an agency might wish to change the labels but provide a mapping to assist developers). For example, a dataset might aggregate numerous documents, and a minor change in one document should not necessarily change the labels for the other documents. Recall the side note from our previous post that labels should be separate from ordering information. Corrections to a dataset may add, remove, or reorder items. Detaching order from labels can help agencies ensure label consistency across dataset versions. In addition, the last post and its comments discussed whether agencies should provide a label that is separate from its internally used agency label. This separation allows labels to remain consistent even when Subsection X becomes Section Y based on the internal agency labels. Note that these points about consistent labeling can be useful whenever a dataset could have multiple versions: for example, consistent labeling might be beneficial across various versions of a bill.

Similarly, the structure that agencies use for datasets, the locations where the datasets are hosted, and other details of a dataset sometimes must change. Suppose that an agency releases various statistics each month. When the agency is asked to provide a new statistic, the new data may necessitate changes to the XML schema. Alternatively, the agency may decide to host data at the address "http://www.agency.gov/YEAR/MONTH/data.xml" rather than "http://www.agency.gov/MONTH-YEAR/data.xml," causing issues for automated tools that periodically check for and download new data. To

reduce the adverse impact of these changes on developers, agencies should provide detailed notice of the changes as early as possible. Early notice gives developers time to modify their tools. These notifications can occur via an email list or RSS feed providing details of the changes in a clear, consistent format.

The possibility of changes and their impact on developers should be taken into account at all stages of the data production process. Suppose an agency adds an element to a schema that specifies a unique individual, but the schema may someday need to specify a corporation instead. Although the agency should not speculatively add unnecessary elements to the schema, it should be mindful of possible changes when designing the rest of the schema. Various design choices may minimize the impact of a change if necessary later. Agencies should also avoid the urge to alter a schema dramatically each time it requires a minor change. A major overhaul—even when done to clean up the schema—may require equally dramatic changes in tools utilizing the data. To ensure that developers notice changes to XML schemas, both schema files and datasets should contain a prominent schema version number. If an agency changes the location where data is hosted, it should consider temporarily using aliases so that requests using old addresses automatically take you to the correct data. Once the old addresses are phased out, agencies should use a standard HTTP 404 status code to indicate that the requested data was not found at the specified location. Simply supplying a "Not Found" page without this standard code could make life harder for developers whose automated tools must instead parse this page.

When making changes, agencies should consider soliciting input directly from developers. Because the preferences of developers might not be obvious, this input can lead to choices that help developers without increasing the burden on agencies. In fact, developers may even come up with ideas that make life easier for an agency.

Our next and final post in this series will discuss a handful of additional issues for agencies to consider.

BEST PRACTICES FOR GOVERNMENT DATASETS: WRAP-UP

BY JOE CALANDRINO - POSTED ON MARCH 12TH, 2010 AT 9:26 AM

For our final post in this series, we'll discuss several issues not touched on by earlier posts, including data signing and the use of certain non-text file formats. The relatively brief discussions of these topics should not be interpreted as an indicator of their importance. The topics simply did not fit cleanly into earlier posts.

One significant omission from earlier posts is the issue of data signing with digital signatures. Before discussing this issue, let's briefly discuss what a digital signature is. Suppose that you want to email me an IOU for $100. Later, I may want to prove that the IOU came from you—it's of little value if you can claim that I made it up. Conversely, you may want the ability to prove whether the document has been altered. Otherwise, I could claim that you owe me $100,000.

Digital signatures help in proving the origin and authenticity of data. These signatures require that you create two related big numbers, known as keys: a private signing key (known only by you) and a public verification key. To generate a digital signature, you plug the data and your signing key into a complicated formula. The formula spits out another big number known a digital signature. Given the signature and your data, I can use the verification key to prove that the data came unmodified from you. Similarly, nobody can credibly sign modified data without your signing key—so you should be very careful to keep this key a secret.

Developers may want to ensure the authenticity of government data and to prove that authenticity to users. At first glance, the solution seems to be a simple application of digital signatures: agencies sign their data, and anyone can use the signatures to authenticate an agency's data. In spite of their initially steep learning curve, tools like GnuPG provide straightforward file signing. In practice, the situation is more complicated. First, an agency must decide what data to sign. Perhaps a dataset contains numerous documents. Developers and other users may want signatures not only for the full dataset but also for individual documents in it.

Once an agency knows what to sign, it must decide who will perform the signing. Ideally, the employee producing the dataset would sign it immediately. Unfortunately, this solution requires all such employees to understand the signature tools and to know the agency's signing key. Widespread distribution of the signing key increases the risk that it will be accidentally revealed. Therefore, a central party is likely to sign most data. Once data is signed, an agency must have a secure channel for delivering the verification key to consumers of the data—users cannot confirm the authenticity of signed data without this key. While signing a given file with a given key may not be hard, surrounding issues are more tricky. We offer no simple solution here, but further discussion of this topic between government agencies, developers, and the public could be useful for all parties.

Another issue that earlier posts did not address is the use of non-text spreadsheet formats, including Microsoft Excel's XLS format. These formats can sometimes be useful because they allow the embedding of formulas and other rich information along with the data. Unfortunately, these formats are far more complex than raw text formats, so they present a greater challenge for automated processing tools. A comma-separated value (CSV) file is a straightforward text format that contains values separated by line breaks and commas. It provides an alternative to complicated spreadsheet formats. For example, the medal count from the 2010 Winter Olympics in CSV would be:

```
Country,Gold,Silver,Bronze,Total
USA,9,15,13,37
Germany,10,13,7,30
Canada,14,7,5,26
Norway,9,8,6,23
```

...

Fortunately, the release of data in one format does not preclude its release in another format. Most spreadsheet programs provide an option to save data in CSV form. Agencies should release spreadsheet data in a textual format like CSV by default, but an agency should feel free to also release the data in XLS or other formats.

Similarly, agencies will sometimes release large files or groups of files in a compressed or bundled format (for example, ZIP, TAR, GZ, BZ). In these cases, agencies should prominently specify where users can freely obtain software and instructions for extracting the data. Because so many means of compressing and bundling files exist, agencies should not presume that the necessary tools and steps are obvious from the data files themselves.

The rules suggested throughout this series should be seen as best practices rather than hard-and-fast rules. We are still in the process of fleshing out several of these ideas ourselves, and exceptional cases sometimes justify exceptional treatment. In unusual cases, an agency may need to deviate from traditional best practices, but it should carefully consider (and perhaps document) its rationale for doing so. Rules are made to be broken, but they should not be broken for mere expedience.

Our hope is that this series will provide agencies with some points to consider prior to releasing data. Because of Data.gov and the increasing traction of openness and transparency initiatives, we expect to see many more datasets enter the public domain in the coming years. Some agencies will approach the release of bulk data with minimal previous experience. While this poses a challenge, it also present an opportunity for committed agencies to institute good practices early, before bad habits and poor-quality legacy datasets can accumulate. When releasing new datasets, agencies will make numerous conscious and unconscious choices that impact developers. We hope to help agencies understand developers' challenges when making these choices.

After gathering input from the community, we plan to create a technical report based on this series of posts. Thanks to numerous readers for insightful feedback; your comments have influenced and clarified our thoughts. If any FTT readers inside or outside of government have additional comments about this post or others, please do pass them along.

DRAFTING GUIDELINES FOR GOVERNMENT DATA CATALOGS

WRITTEN BY LUIGI MONTANEZ; 03/29/2010 12:49 P.M.;
WRITTEN WITH DAVID JAMES.

A major focus of the Sunlight Labs is to push government to publish its data online. In recent months, we've gained in-depth familiarity with government data catalogs through our work on the National Data Catalog. The most prominent example of a data catalog is data.gov. Since its launch last year, a handful of states and cities have followed suit with their own efforts. As more data catalogs come online, we want to make sure their contents are open and exchangeable. We want to determine how to best structure the data catalog itself, and we want to ensure that the metadata it contains -- the data about the data -- exists in the most accessible way possible.

Last week, Clay posted three challenges for the community to tackle, and this is challenge #3. We're looking to start this conversation now and move towards consensus within a few months. I was at Transparency Camp, digging deeper into this topic, putting us on the path to make recommendations that governments can adopt quickly.

Resource-Oriented Architecture

A data catalog lives on the Web, so it makes sense that it embrace the architecture of the Web. A simple Web site with a few pages is good for citizen use, but doesn't lend itself to being interoperable. As such, we strongly recommend following the Resource Oriented Architecture (ROA) guidelines. When applied to the concept of a government data catalog, ROA means:

* Resources that represent the data sources, agencies, and jurisdictions (if there are more than one) should exist.
* Each resource should have a unique URI, so that each one can be addressed individually.
* Resources should be made available in both human-friendly (HTML) and machine-readable formats (such as XML or JSON).
* Content negotiation should be used to serve the resource correctly depending on the user agent.

Defining a Vocabulary

Let's turn to the metadata that describes a particular government data source. Existing government data catalogs like data.gov have already established some best practices. But going back through the Internet's history, it's helpful to contemplate the work of the Dublin Core, which over a decade ago published a set of fifteen metadata elements for describing online resources. The Dublin Core spec applies to a wide range of things published online, from videos to academic papers to datasets. The fifteen elements forming the spec give us a good starting point, and remind us how similar online government data is to any other kind of resource published online. Some elements are not fully applicable (Language and Contributor, for example). Others can be broken up into several elements. Coverage can mean a physical geographic area, a political jurisdiction, or a period of time.

Going from theoretical to practical, let's look at some sample data pages for existing data catalogs:

* data.gov
* data.dc.gov
* data.gov.uk

* utah.gov/data

Looking at those four examples, we begin to see a lot of similarities among the catalogs and with Dublin Core. From these, we propose a preliminary vocabulary:

Element Name	Description
Title	Title of data source.
Description	Short description of data source.
URL	Permanent, unique URL that contains this metadata. Can be self-referential.
Type	Is the data source a dataset, API, or online database?
Downloads	File format/URL pairs that point to data files.
Created	Creation date of the data source.
Released	Release date of the data source.
Last Updated	Update date of the data source.
Update Frequency	How often the data is updated
Creator	Entity (agency, department, or organization) that created the data.
Publisher	Entity that published the data.
Maintainer	Entity that maintains the data.
Jurisdiction	Political jurisdiction of the data.
Time Period	The time period the data refers to.
Grouping	Can the data be grouped with a larger set of similar data? Recommended for data sets scoped to a time period or jurisdiction.
License	The license under which the data set is released.
Documentation	Any documentation, such as a data dictionary, or a reference (URL) to that documentation.

This vocabulary forms a base-level set of metadata. Individual data catalogs can and should publish more elements as appropriate, but serious efforts should be put into ensuring that the fifteen elements above are present.

The Formats

With a vocabulary defined, we can move to the actual data formats to represent an entry in a government data catalog. We don't need to just choose one. Instead, we can map our general vocabulary to several existing data format standards:

* XML: Maps easily to the vocabulary defined above. Loved by the enterprise.
* JSON: XML's lighter-weight alternative, loved by modern Web developers (and the Sunlight Labs).
* CSV: A good compromise between machine-friendliness and human-readability.
* Microformats: Can act as an easy-to-implement solution since these can be placed on plain ol' Web sites with only a little bit of effort.

Lastly, and possibly most importantly, we need a data format to represent updates to entries in a data catalog. For that, we recommend the Atom Syndication Format. An individual entry in the Atom feed would contain the unique URL identifier and any updated elements. The entry should use any one of the four formats above, with a bias towards XML, since Atom itself is XML.

So in summary, the preliminary proposal is:

* Ensure that you have enough metadata to closely follow the vocabulary.
* If publishing the catalog as a Web site, use the Microformat on the data detail pages.
* If publishing an API, use XML or JSON in conjunction with Resource-Oriented principles.
* If publishing the data catalog as a bulk download, use XML, JSON, or CSV.
* To publish updates of entries in the catalog, use Atom.

We'll be fleshing out examples of these data formats in the Sunlight Labs Wiki over the coming weeks. I'll be working with the good people of Socrata, who announced their SODA API at TransparencyCamp on Saturday, and any other folks interested.