Programming Language Standardization: Patterns for Participation

Allen Wirfs-Brock

Wirfs-Brock Associates, Inc

allen@wirfs-brock.com

Abstract. *A technical design team is a group that is chartered to collectively work to develop solutions to a set of related technical problems. A standards committee is a group is that chartered to produce the normative specification for some technology. A programming language standards committee is a standard committee that is also a technical design team and whose charter is both to develop technical enhancements to a programming language and to produce a normative specification for the extended language. Design teams and standards committees are social organizations and each has its own challenges for members who want to be effective participants. These challenges combine in programming language standards committees.*

This paper presents the first steps towards defining a pattern language that provides guidance on how an individual can be an effective and productive participant on a programming language standards committee.

Categories and Subject Descriptors

•Social and professional topics → Professional topics
•Software and its engineering → Software creation and management → Software post-development issues

General Terms

Patterns, Standards, Programming Languages

Keywords

Pattern Languages, Social Patterns, Standardization, Standards Committees, Programming Languages, JavaScript, ECMAScript

Preliminary versions of these papers were presented in a writers' workshop at the 5th Asian Conference on Pattern Languages of Programs (AsianPLoP). AsianPLoP'2016, February 24-26, Taipei, Taiwan. It was published in Proceedings of 5th Asian Conference on Pattern Languages of Programs. SEAT ISBN978-986-82494-3-1 (paper) and 978-986-824-944-8 (electronic).

1. Introduction

Typically programming languages are initially created by an individual, a small group of designers, or as a project within a business entity. However, if a programming language achieves very wide adoption and becomes important to many different stakeholders there is often a call to create a standard language specification that is not under the control of a single individual or organization. For legal reasons, such standards are typically developed under the auspices of recognized standards setting or standards development organizations (SSOs and SDOs) such as ISO, ANSI, Ecma International, JISC, IEEE etc. The actual work of developing and maintaining such a standard is delegated to a technical "standards committee" consisting of individuals and organizational delegates with an interest and expertise in that particular language.

Once a language is standardized in this manner the responsibility of managing the ongoing evolution of the language largely transfers from the language's original designers to the participants in its standards committee. Individual or organizations that want to directly influence the evolution of such a language need to join and effectively participate as members of the language's standards committee. But simply becoming a member is not enough. In order to exert influence member need to learn how to effectively work within the committee setting.

Each SSO/SDO and individual standards committee has its own culture, rules, processes, technical infrastructure, and idiosyncrasies. In this paper I have tried to identify patterns of effective participation that are likely to be applicable in many such situations. These patterns codify observations made by the author as a standards committee participant for two significant programming language: ANSI X3J20 (1993–1998) the committee that developed the first standard for the Smalltalk programming language; and, Ecma International TC39, the committee that maintains the specification for JavaScript[1]. The author was project editor for three editions of the JavaScript standard.

2. Why Are Programming Language Standardized?

Over the history of computing thousands of programming languages have been designed and implemented. However, very few of them are ever used by anybody other than a small circle of associates of the languages' original designers. The maintenance and evolution, if any, of such language designs is of little interest outside of its small circle of users. Occasionally, a programming language breaks out of its initial niche and begins to be used by a much larger community. Once this occurs, there are many forces that can impact the evolution of the language and have a significant impact on its community of users.

For example, multiple implementations of the language may be created. If implementations differ significantly in either small or large details, dialects of the language are created that may make it impossible to move programs between implementations.

As usage grows, issues with the original language design may be discovered. For example, there may be technical mistakes made in the original design that are hard to implement or which degrade the potential performance of programs written in the language. With wide usage, it may be discovered that the language has error-prone features that act as "bug farms" as programmers try to use those features.

---

[1] The JavaScript standard is formally known as ECMA-262: *The ECMAScript Language Specification*.

With broad usage, idiomatic usage patterns of the language's features may become apparent. Such patterns may suggest new language features that, if available, could accomplish the same result in a more concise manner.

Over time, the language may be used in ways or in support of application domains that were not contemplated by the original language design. Such usage may suggest the need to add new features to the language that would enhance its applicability in those situations.

As the community of users expands, there may be conflicting desires for how the language might be enhanced. For example, programmers with an object-oriented programming background might wish to add better support for object-oriented constructs and conventions. But programmers with a functional programming background may find such features uninteresting or even undesirable.

There may be business interests that potentially impact the evolution of a programming language. Some business entities may seek advantage by changing the language in ways that align with their current technologies or business strategies. Other business entities may feel threatened by broader usage of a language and try to discourage enhancements that would enable such usage.

Such issues and concerns may be outside of the scope of interest of the original language designers. Or the users and implementers of the language may not have sufficient confidence in the original designers to leave the resolution of such issues to them. When the popularity of a programming language and the backlog of pending language evolution issues reach such levels, a common solution is to form a standards committee to resolve the issues and to assume overall responsibility for the on-going evolution of the programming language.

A standards committee typically documents its decisions by publishing one or more specification documents that become the official normative definition of the programming language.

3. The Operation of Standards Committees

A programming language standards committee operates using the rules and processes of the SSO/SDO under whose auspices it is chartered. While there is variation among such organizations they are similar enough that a generic description is sufficient to understand how most language standards committees operate.

Commonly, the actual membership of a language standards committee consists of organizations: for profit businesses, non-profits, academic institutions, and government agencies that have an institutional interest in the language. In some cases membership may include individuals representing their own interests or "invited experts" who bring unique technical expertise or experience related to the language.

Each member organization is represented by people who are affiliated with that organization. We will use the term "delegate" for the actual people who attend committee meetings or otherwise contribute to the work of the committee. Delegates include both the representatives of member organizations and unaffiliated individuals such as invited experts. Typically delegates are technical experts with extensive experience using or implementing the programming language. However, occasionally delegates are chosen for some general expertise such as experience writing specifications or working within standard organization processes rather than because they are language experts.

It is common for some organizations to have multiple delegates simultaneously participating on a standards committee. Different delegates from the same organization may represent

different areas of technical expertise or different perspectives regarding the language. For example, one delegate might represent the users of the language within an organization while another delegate might represent a group within that organization that creates implementations of the language. When an organization sends multiple delegates to a language standard committee it is often an indication that the organization views the language as being highly important to it. The delegates from a single member organization may not always be in agreement. They may have differing technical opinions or represent divergent requirements.

Depending upon the rules it operates under, some decisions (or even all decisions) made by a language committee may be made by taking a formal vote. Typically member organizations are granted one vote per organization rather than one vote per delegates. This means that the delegates that represent a single organization must reach agreement on how the organization will vote.

While voting sometimes occurs, language standards committees generally prefer to make decisions via a consensus building process. Consensus is achieved when a super majority of the delegates support a particular decision and there is no major objection to making that decision. Consensus on a particular decision does not require that every delegate enthusiastically support the decision, but it does require that every delegate is willing to accept that decision.

Creating the standard for a programming language is a highly technical activity and most delegates are computer scientists or software engineers. Often such individuals prefer an unstructured informal work environment. Smaller language standards committees often operate with very little formal structure or ceremony. However, committees with a large number of delegates typically need to formalize their meetings with a prearranged agenda and other common practices for managing a large meeting.

Development of a language standard is not a green-field development activity. It typically starts after a programming language has already gained significant usage. A standards committee has to start with the language that already exists. They cannot simply throw away the existing language and design a different but better replacement language. A language standards committee exists to solve problems that arise from the current state of the language. Such problems might include the lack of a comprehensive specification, undesirable variation among different implementation of the language, problematic existing features, or the need for new features that will enhance the utility and applicability of the language.

Development of a new or significantly revised language standard is typically a multi-year activity and in some cases may take more than a decade. A typical work cycle of a language standards committee is to identify such problem areas, reach consensus that a particular problem needs a standardized solution that is within the scope of the language design; solicit from the delegates proposals for additions or changes to the language that will provide a solution to the problem; reach consensus on a specific proposal; and finally fully specify the complete specification of the proposed solution and integrate it into the normative language specification.

Many such cycles will typically be simultaneously in process within a standards committee. But, the nature of programming language design is that each new feature is likely to interact with other language features. So each new feature proposal has to consider its interactions with all pre-existing features and all other new features that are also under development.

4. A Pattern Language for Language Standards Delegate

While some delegates remain active on a standards committee for years there is also a regular turnover of delegates as individuals change organizational roles and affiliations and as personal interests evolve. Because of this turnover, standards committees have the continuing challenge of integrating new delegates as effective collaborators and contributors. We envision a pattern language that provides guidance to delegates who want to be effective participants within a programming language standards committee. This pattern language includes groups of patterns that that are applicable to various phases of a delegate's participation lifecycle. The particular phases that have been identified include a delegate initially joining a standards committee, a delegate being an active participant, and situations where a delegate acts in a leadership role within a standards committee.

Participating in the development of a language standard is both a technical and social activity. So far, the patterns we have identified are focused on the non-technical aspects of standards committee participation. Standards committee participation is similar to other technical social activities and patterns that are applicable to such activities are likely also applicable to standards committee participation. For example, some activities within a language standards committee are focused upon "change" and many of Manns' and Rising's Fearless Change Patterns [MR2005, MR2015] are applicable to those situations. Our pattern language differs in that we describe our patterns using terminology and situations that are specific to programming language standards development.

In the next three sections we introduce the three major pattern groups using patlet[2] level descriptions of each pattern. Following that we provide a complete pattern definition of one pattern from each group. The complete patterns are presented using a simple Alexandrian format that is intended to be accessible to standards committee delegates who are not pattern language experts.

4.1 Patterns for Joining a Standards Committee

As a new delegate, you need to understand the inner workings of the language standards committee before you can effectively participate. The following patterns will help a new delegate learn how the committee works and integrate themselves as an effective participant in committee meetings.

Don't Talk Too Much

The first few meetings you attend should be devoted to listening and observing. Only speak up if you are sure you have something unique and important to add to a discussion.

Learn the History

When joining a committee you may already have many ideas about how the language could be improved. Similar ideas have probably already been considered and quite possibly rejected by the committee. Before presenting your ideas study the deliberation history of the committee to ensure your ideas have not already been considered.

[2] A *patlet* is a pattern name followed by a short description of the pattern.

Understand the Other Players

Learn the names and affiliations of the other delegates. What interests do they represent? What are they trying to accomplish within the committee? What is their vision for the evolution of the language?

Volunteer to Take Notes

Standards committees need to capture detailed technical meeting notes that record their deliberations. But good note takers are hard to find. By volunteering to take notes you force yourself to be a listener and demonstrate that you can be a valuable member of the committee.

4.2 Patterns for Participating

After you have attended a few meetings and learned how the committee functions you should be ready to become a more active participant in its ongoing technical work. The following patterns will help a delegate to be an active committee participant that is able to advance proposals through the committee process.

Have Goals and Plans

Know what you and the organization you represent want to accomplish relating to the language standard. Based upon your knowledge of the committee, create a plan for achieving those goals.

Be a Contributor

A standards committee needs delegates to actually do work. Be an active contributor rather than a passive participant. Write and promote proposals that are aligned with your goals.

Start Small

Start with a small and uncontroversial proposal. Use it to learn how to successfully get a proposal accepted and to build your credibility within the organization.

Claim a territory

Try to identify an area of the language design that has issues that are currently not being addressed by the committee. By making proposals to address those issues you can become the de facto expert within the committee on that area of the language.

Find Allies

Identify other delegates whose interests and goals align with yours and your organization. Work with them to develop joint proposals.

Pick Your Battle

There are many ways to solve most problems and you will find that some proposals take a different approach to a problem than you would have taken. Express your views. But don't get into fights about unimportant differences.

Back Pocket Alternative

A majority of the delegates may approach a problem in a manner that you believe won't work but you cannot convince them that they are going down the wrong path. Instead of wasting additional time trying to convince them, work on developing an alternative solution. If you are right, the majority will eventually recognize their mistake and your solution will be available to take its place.

Break Consensus as Your Last Resort

Maintaining consensus is important to the effective operation of a language standards committee. As a delegate you should avoid breaking consensus, as the majority of the committee will likely perceive that as a hostile act. But if you or your organization has a serious disagreement with a proposed action of the committee then you must make it clear that there is not a consensus to proceed with that action.

4.3 Patterns for Leading

Like any human organization, a standards committee requires effective leadership. The following patterns address leadership roles and situations within a language standards committee.

Become an Editor

A standards editor typically has responsibility for authoring the specification document that reflects the consensus decisions of the committee. An editor can be very influential within the committee because of the "fingers on the keyboard" nature of this job. Put yourself forward as a candidate for open editorship positions.

Always Have a Draft

The job of a language standards committee is to produce a language specification. But an unfocused committee can talk for years without making actual progress on a specification. As an editor, you should always have a draft specification that reflects current decisions. Once an edition of the standard is completed, the editor should immediately make a copy and re-label it as the first draft of the next edition.

Release Goals and Schedule

A standards committee needs to agree on what it is trying to accomplish and in what timeframe. Ensure that for each edition of the language standards there is a written set of goals and a target completion date.

The Max-Min Solution

Even when there is consensus on the general solution to a language design problem it may be hard to get consensus on all the details of a complex set of interrelated features that implement that solution. This can delay the schedule or put at risk the ability to address an important issue. Try to get consensus on a "maximally minimal" solution to the problem that strips away all the bells and whistle features that are a source of disagreement.

Fork an Ancillary Standard

There may be major new language features that are only important to some members or only applicable in some environments. Such a feature may be better served by the development of an ancillary standard rather than making it part of the primary language standard. This enables the members that consider it to be important to focus on it and removes a distraction to the members who are uninterested.

5. Pattern: *Volunteer to Take Notes*

You have recently joined the standards committee as a delegate. You are being careful to *Don't Talk Too Much*.

You are trying to *Understand the Other Players*, who they represent, and their positions on various issues.

You might be concerned that other delegates don't yet see you as adding much value to the committee.

Committees often have trouble identifying delegates who are willing to serve as a note taker.

❖❖❖

It is hard to just listen. This may be particularly true if, like many delegates, you are a senior person in your organization that is used to leading discussions. You may find it difficult to hold back from getting into the discussion on some topics. You may think you have something to contribute but aren't sure that you have enough context to be confident that what you would like to say would add value to the discussion. You don't want to say something that is going to be dismissed because of your lack of understanding or your need to *Learn the History*.

Perhaps there are discussion topics that are new to you or that you find less interesting and you are finding it difficult to stay focused while just listening. If the committee is large enough you may be finding it difficult to keep track of all the other delegates and their positions. You might be taking some personal notes to help keep this all straight in your mind and to help you stay focused.

Detailed discussion notes are important to a standards committee as they help track the evolving positions that lead to a consensus that requires several meetings to emerge. Committees often have a hard time finding members willing to assume the job of note taker. The problem is that most delegates want to be active participants in the discussions and it is almost impossible to be an active participant and a good note taker at the same time.

❖❖❖

Therefore, volunteer to be the note taker for all or part of each meeting.

Meeting notes usually are not literal transcripts. Instead they are typically discussion summaries that try to clearly capture the essential points being made by each participant. In order to take good notes, you have to listen carefully. You have to pay attention. As a note taker you have to recognize the nuances of the discussion so you can correctly capture all of the essential points while filtering out inessential rhetorical flourishes. You must be able to accurately identify each speaker.

If you are being a good note taker you won't be able to talk too much, as all of your attention will be focused on listening and understanding.

If you are doing a good job of note taking you won't have to worry about whether or not other delegates see you as a valuable addition to the committee. Everybody loves good meeting notes and appreciates the work that is required to produce them.

❖❖❖

While note taker is a great way to initially contribute to the committee you need to be careful not to get locked into the position of permanent note taker. At some point you will want to be a more active discussion participant or to champion your own proposals. It is very difficult to do so while also serving as note taker. Over time, try to restrict your note taker duties to parts of the meeting where you don't need to be an active participant.

Some people just don't have very good note taking skills. Other have a hard time learning associations between faces and names. If you are such a person you probably don't want to volunteer as the official note taker. Instead, practice by taking personal notes.

6. Pattern: *Back Pocket Alternative*

There is a significant new language feature area, issue, or work item that the standards committee has decided to work on.

The majority of the delegates have focused on a particular solution or technical direction.

You are convinced that the majority direction has serious flaws that if not recognized will have serious detrimental consequences.

You think there is a better alternative but have not yet been able to convince the majority of the delegates that there is a problem with

❖❖❖

Sometimes the majority of the delegates will coalesce around a feature or idea that you disagree with. In many such cases, after expressing your concerns, it is better to *Pick Your Battles* and accept the majority preference. Minor differences of technical opinion or style are not important enough to justify breaking consensus.

But what if the disagreement is about a major feature or issue that will have serious consequences? What if you believe that the majority direction is not only wrong but is doomed to failure? You will probably want to continue to point out (in a non-disruptive manner) you concerns. But there probably isn't anything else you can contribute to the work of the majority.

You might *Break Consensus as Your Last Resort*, but because you believe that the majority is on a path to failure there is no reason you should suffer the negative consequence of being a delegate who breaks consensus. It's better to ready yourself to help the committee move forward when they recognize their mistake.

❖❖❖

Therefore, work on an alternative solution and develop it to the state where it could replace the currently favored solution.

The most useful thing you can do in this situation is to develop your alternative solution and have it ready go if/when the majority approach is recognized as a failure. If you are wrong and the majority approach works out then you will have only wasted your own time and haven't hampered the majority's work. If you are correct, you will be in a position to "save the day" when the failure is recognized.

❖❖❖

It may take a long time for the committee to recognize that they are going down a dead-end path. But if it really is a dead-end, it will ultimately be recognize it as such.

There are probably other delegates that have similar concerns to yours. But they may not know of a viable alternative and hence are just going along with the majority. Try to *Find Allies* and get them involved in the development of your alternative.

Continue to make it known that you have concerns about the majority solution, but don't do so in a way that is disruptive to the work of the majority. You are not trying to make them fail; instead you are providing a safety net in case they do fail.

You don't need to hide the fact that you are working on an alternative. Knowledge that an alternative is available may make it easier to the majority to recognize that they are on the wrong path.

This pattern should be reserved for disagreements about major work items. But even so, its application can vary in scale from work on a single major feature or language subsystem up through the plan for a complete revision of the language.

Ecma TC39 spent several years working on a major revision of the JavaScript language that was known as ECMAScript 4. This revision included major changes to the core semantics of the language and hoped to incorporate some features that were still topics of active academic research. A majority of the committee delegates supported the ECMAScript 4 plan while a minority had strong reservation about both the approach and its likelihood of success. Ultimately, the majority never produced a draft specification for ECMAScript 4 (see *Always Have a Draft*) while the minority produced and had waiting in their "back pocket" a nearly complete draft of a smaller incremental update to the then current ECMAScript 3 standard. When work on ECMAScript 4 was finally abandoned the alternative draft was already available and within a year it was published as the ECMAScript 5 specification.

Choosing to develop a *Back Pocket Alternative* is a major commitment of time and resources. Development of the ECMAScript 5 alternative proceeded with multiple contributors for over a year prior to it being accepted in place of ECMAScript 4 (which had been under development for over two and a half years by a larger set of contributors).

7. Pattern: *Max-Min Solution*

There is agreement among the delegates that there is a new requirement or a shortcoming of the language that requires introduction of a major new language feature.

There is agreement about the purpose and core semantics of the feature.

The overall feature is complex. Beyond the core semantics of the feature there are many design details and secondary semantics characteristics that the committee is having trouble agreeing upon.

Because of lack of consensus about all of the feature details there is a risk that the entire feature may have to be abandoned and hence the motivating language problem would be left unaddressed.

❖❖❖

In some cases, it is easier to get agreement about a general approach to solving a problem then it is to get agreement about all the details of the solution. In the case of language design this might manifest as agreement about the need for a statement that performs some specific

function, but disagreement about the syntax of the statement or about subtle semantic details. For example, there might be agreement that a class declaration is a need in the language but there are significant differences of opinion about the exact syntax of a class declaration, the semantics of a class and its objects, the subparts of a class definition, etc.

The more complex the feature, the more details there are to disagree about. A committee that is overly focused on too many details can get into a situation where even though there is consensus on the overall shape of a solution it cannot reach consensus on all the details.

If the committee cannot reach consensus on all the details of a complete feature design, the feature cannot become part of the language standard.

❖❖❖

Therefore, propose a maximally-minimal design that only includes the essential details that all the delegates can agree on.

A maximally-minimal solutions is the most comprehensive design of the feature that adequately addresses the problem that originally motivated the feature, is complete (has no loose end), and that is acceptable to all delegates as a consensus solution. A maximally-minimal solution will not have all (or perhaps even none) of the "bell-and-whistle" feature details that some delegates might like to see, but it has the essential functionality and feature details that allow it to achieve consensus as a solution to the base problem.

A maximally-minimal solution is a tool for breaking decision deadlocks within a standards committee. If consensus can be reached on a minimally-maximal solution, that design can be incorporated into the draft language standard. It becomes a fixed foundation that provides the context for discussing additional feature details.

A maximally-minimal solution must be complete and fully usable without adding any additional feature details as there is no guarantee that the committee will ever reach consensus on making additions to the design.

❖❖❖

In order for *Max-Min Solution* to be used there must already be a consensus that the general feature is needed in the language. In addition there must already be agreement (or at least a strong possibility of agreement) on core syntax and semantics of the feature.

By defining a maximally-minimal solution we are limiting the scope of discourse in order to make it easier for the committee to agree upon important details. We are also getting a base-line feature into the draft language specification (see *Always Have a Draft*). This bounds future feature detail discussions by constraining feature additions to be compatible extensions of the maximally-minimal design.

For example, early in the work on ECMAScript 2015, TC39 reached consensus that some sort of explicit class declaration syntax was needed to augment the ad hoc object definition techniques provided by previous editions of the language. TC39 subsequently spent almost three years discussing alternative proposals for the syntax and semantics of class declarations without achieving consensus. The stalemate was only broken when a mailing list participant [LEG] suggested that TC39 focused on developing a proposal that addressed only the "absolute minimal requirement". Within a few days a "Maximally Minimal Classes" proposal [WIR] was available to committee members and within a few months the committee was able to reach full consensus on adopting that proposal.

Work on extensions to a maximally-minimal design can continue immediately after it is accepted or it might be deferred until a latter edition of the language specification. The

experience in Ecma TC39 was that once there is agreement on a maximally-minimal design, delegates will generally want to move on to dealing with other unrelated pending issues and defer most work on extending the maximally-minimal design for a future specification revision.

The concept of a maximally-minimal design is similar to the concept of a Minimum Viable Product as used by the Lean Startup Movement.

8. Conclusions

The patlets and three complete patterns presented in this paper are a first step to developing a pattern language for language standards committee participation. Completion of this pattern language will require the authoring of complete patterns for the remaining patlets. So far, this pattern language reflects the experiences of a single expert. Iba suggests [IBA] that pattern languages that address human interactions should be developed via collaborative mining, writing, and improvement. Our pattern language would likely be improved by applying such a collaborative process.

A pattern language for language standards committee participation is targeted to a very specific audience. A programming language standards committee is just one of a vast number of specialized situations in which humans collaborate or otherwise interact. Most of these situations can be viewed as instances of a few generalized styles of collaboration. Participation in a programming language standards committee is a specialization of general standards committee participation, technical design committee participation, generic committee participation, collaborative technical design, and human collaboration in general. Pattern languages either have or could be developed for any or all of these activities and it is reasonable to expect that there would be significant overlap between such pattern languages. Is this a feature or a bug? What are the advantages and disadvantages of a very specialized human interaction pattern language versus a pattern language for a more general form of the same basic activity? Do we need a catalog of common human interaction patterns that can be used to construct pattern languages for specialized situations? Further research into these and other pattern language meta-issues would be helpful to practitioners trying to develop human interaction pattern languages.

Acknowledgements

Emiliano Tramontana provided valuable comments and feedback as the AsianPLoP 2016 shepherd for this paper. The members of our AsianPLoP 2016 Writers Workshop Group also provided many valuable comments.

References

[IBA] Iba, T. 2011. "Pattern Language 3.0 Methodological Advances in Sharing Design Knowledge," International Conference on Collaborative Innovation Networks 2011 (COINs2011).

[LEG] Leggatt, Russell, “Finding a ‘safety syntax’ for classes”. Posting to es-discuss mailing list. March 19, 2012. https://mail.mozilla.org/pipermail/es-discuss/2012-March/021430.html

[MR2005] Manns, Mary Lynn and Rising, Linda, *Fearless Change: Patterns for Introducing New Ideas,* Addison-Wesley, 2005.

[MR2015] Manns, Mary Lynn and Rising, Linda, *More Fearless Change: Strategies for Making Your Ideas Happen.* Addison-Wesley, 2015.

[WIR] Wirfs-Brock, Allen, "Maximally Minimal Classes". Ecma TC39 Stawman Proposal. March 25, 2012. http://wiki.ecmascript.org/doku.php?id=strawman:maximally_minimal_classes&rev=1332652575