Code Contribution Guidelines

Introduction

This short guide provides instructions for the developers that plan to contribute code to the midPoint project.

MidPoint project was started by small and very dedicated team of engineers that knew each other quite well. Therefore the early years of midPoint development went without any major organizational issue. There were development guidelines, but those in fact were not really necessary. In fact the first rule of midPoint development was that there are no rules of midPoint development.

As the code base matured and midPoint gained wider adoption first external contributions began to appear. Those were usually small and simple things. But later on bigger contributions appeared. And those were not entirelly without problems. This is quite an expected development. MidPoint core team communicates in a very natural and efficient way. And this is something we do not want to change as this is the pillar of midPoint development efficiency. But it is much harder to maintain good communication with a broader, geographically and culturally diverse community. Hence the need for contribution guidelines.

Communication Channels

There are two communication channels for developers:

- MidPoint mailing lists are used for general communication with midPoint community. Communication about midPoint development and contributions is acceptable and encouraged.
- Development chat is on-line chat room that is dedicated to interactive communication among midPoint developers. One of the primary purposes of this chat is to support communication about code contributions.

Please feel free to use any of those communication channels. Communication about midPoint contributions is more than welcome. Just please keep in mind that those communication channels are public. And please maintain appropriate levels of politeness and civility.

On the other hand, we strongly discourage private communication with midPoint developers. Please do **not** address midPoint developer privately, whether it is private mail or private chat message. Private communication is not visible to the public and therefore it limits propagation of knowledge about midPoint code. It is our **policy** not to respond to such private messages. Therefore do not be surprised if you won't get any reply. Private communication is reserved for midPoint subscribers. **Community communication is always kept open.** Including communication about code contributions.

Accepting Contributions

Generally speaking, midPoint project is open to contributions of any kind. However, there are three critical criteria for accepting a contribution:

1. **Quality**: Contribution quality should be appropriate - relative to existing midPoint code. We will not accept low-quality contributions. See below for details.
2. **Maintainability**: We are deeply concerned about midPoint maintainability and sustainability of the project. Your contribution may be great and it may improve midPoint today. But even big and substantial contributions may be refused if it is likely that a contribution becomes a maintenance burden in the future.
3. **Licensing**: MidPoint is dual-licensed under Apache License and EUPL. Therefore we have to require Contributor License Agreements (CLAs) from all midPoint contributors. We need that to re-license the contributions under both licenses.

Contribution Quality and Maintainability

This section provides guidelines for quality of midPoint contributions and explains our maintainability strategy. Please read those guidelines carefully before you start your work on midPoint contribution. A contribution may be refused if it violates any of those guidelines. We would hate to refuse contributions as that means that a developer's work is wasted. That is always a terrible thing. Therefore please, make sure you follow those guidelines when preparing a contribution.

Discuss The Contribution

First and most important guideline: **Discuss your contribution before starting the work.** Let us know that you plan to contribute **before** you do any coding. Use mailing list or developer chat. Just let us know.

MidPoint is big and complex system. Significant part of midPoint functionality is documented. And even architecture and important design principles and decisions are documented (with some exceptions). But as midPoint itself is big, the documentation is also big. And it may be confusing. It is easy to miss something. Especially if you are first-time contributor. Therefore **discuss your idea** with midPoint core team. We may point out a better way to implement a particular feature. It also often happens that we have designed similar feature in the past and that there is already some preparatory work done in midPoint. That may make your work much easier. Such preparatory work is almost never documented, as such work is almost never funded and documenting it would be too much overhead. Therefore do not be afraid to ask. We will be glad to respond.

However, make sure that you mention contribution in your question. Ideally include the word "contribution" in the subject of the mail message. There are too many questions on mailing list. Many of the questions are asked by people that do not contribute back to the community. Such questions are low priority for MidPoint core developers. However, if you plan to contribute to midPoint please let us know and make your question stand out. That will attract attention of midPoint developers.

Make sure you discuss your contribution especially if it is new functionality. There is no need for discussion if you contribute just a simple and quite obvious bugfix. But if you extend midPoint functionality, then discussing the contribution before you do any work is almost a necessary condition for contribution acceptance. Therefore please avoid future disappointments and discuss your ideas very early. Discussing the contribution significantly lowers the risk of a contribution being refused.

Content of Contribution

Contribution should contain:

- **Code** (obviously): This is the "core" of the contribution. Probably self-explanatory.
- **Tests**: Whether your contribution is new functionality or a bugfix there needs to be appropriate automated tests for it. See below.
- **Developer documentation** (javadoc, comments): Provide inline documentation at an appropriate granularity. See below.
- **User documentation** (if needed): If you implement new feature or extend midPoint functionality there needs to be a appropriate documentation that allows users to actually use the functionality.

Code

Do what you have to do. Just make sure to read through following (those that are applicable):

- Development Guidelines
- GUI Development Guide
- GUI Look and Feel Style Guide
- Connector Development Guide

You should base your contribution on the most recent state of the **`master` branch**. If your contribution is based on older branches (e.g. maintenance branches) you will be very likely asked to rebase your contribution to a master branch. The master branch is the only long-term "living" branch for midPoint development. Support branches are "dead end" branches. They will never get merged back into the master. Those branches are there only for support (backport) purposes. Therefore contribution based on those branches does not really make sense.

The only exception from the above rule are security fixes. We will accept security fixes at any time and for any branch. In case of security issues we will take the responsibility of applying the fix to all appropriate (supported) midPoint versions and active branches.

Make sure that your code **does not break any existing functionality**. Run all the tests (`mvn clean install`). Make sure that they are all passing - to an applicable degree. If you are contributing to master branch (usual case), it may happen that some tests are failing. In that case make sure that the same tests are failing before your contribution and after your contribution. In other words: make sure that your contribution haven't made the situation any worse.

Tests

Almost all code contributions should contain appropriate automated tests. The basic mechanics of the tests is the same for all cases. But the scope and purpose of the tests differs whether your contribution is a new feature or bugfix:

- **New feature**: Provide tests for your specific use case. This is likely to be "story test". We expect quite a few positive test cases and maybe handful of negative test cases. Unless it is a security feature. For security features we expect few positive test cases and a lot of negative test cases. But for "normal" features you are not expected to write tests for all possible cases. In fact, for new features, the design (discussed beforehand) and readable, maintainable code is much more important than huge number of tests.
- **Bugfix**: Provide test for the issue you are fixing. If the test is feasible then at least one test is required. Bugfixes without a test are likely to be refused. And there is a good reason for this: Test-Driven Bugfixing. In fact, we recommend this approach:
    1. Write a test that exposes particular bug. See how the test fails.
    2. Fix the bug.
    3. See how the test passes.
    4. Contribute the fix together with test.

MidPoint has quite an elaborate environment for creating integration tests and UI tests. More than one third of midPoint code are tests. And most of the tests are integration tests. Therefore there are plenty of test examples in midPoint source code. One useful trick is to take JIRA issue identifier (e.g. MID-4321) and look for that string in midPoint source code. If that issue was reproduced by a test that the identifier should be included in test method javadoc. Therefore it should be easy to find examples for bugfix and feature tests. But there is also a documentation that is supposed to make writing tests easier:

- Integration Tests
- Model Integration Tests
- Story Tests
- GUI testing with Schrödinger
- Writing tests with Schrödinger

Developer Documentation

Provide inline documentation at an appropriate granularity. We are no overly strict about javadoc we do **not** require javadoc for every class or method. First priority is to make code readable. In that case no special comments are needed, not even a javadoc. We recommend using javadoc/comments in following cases:

- **Implementation classes** (class-level javadoc): It would be good idea to document purpose of your class in the class javadoc. Please document the purpose, not the implementation. Implementation is (or should be) obvious from the code. But the purpose is often less obvious. This is optional. If the purpose of the class is entirely obvious you do need to bother with javadoc or any other documentation.
- **Interfaces** (class-level javadoc): All interfaces should have at least short class-level javadoc documenting purpose of the interface. This guideline applies to all Java interfaces and public classes in *-api packages.
- **Interfaces** (method-level javadoc): It is recommended to include method-level javadoc for all methods of an interface. Remember, interface is not just the code. It is a contract. Such contract should be documented.
- **In-line comments** at various places in the code: Less is more. First priority is to avoid any need for comments by making the code readable. But comments may still be appropriate if the code is complicated or if the purpose of the code not obvious. In that case do **not** document how it is implemented. Document the purpose of the code. Document why the code is there. Document what the code is supposed to do.
- **Design decisions**: There are times in the life of every engineer when a decision has to be made. And those decisions may be difficult to do, e.g. choosing the lesser evil. Or choosing to make something work with limited resources ("done" is better than "perfect"). Those are all valid decisions and practical software might not be feasible without such decisions. However, we try to be maximally transparent in midPoint. If such decisions are made, they should be documented. We do not cheat ourselves. Do not lie, do not cover up, do not sweep the garbage under the carpet. If something is bad in midPoint code, it should be pointed out in the comments. It should explained why such a decisions was made. This is the only way how to improve the code later. Code with hard design decisions is likely to be accepted if those design decisions are justified and explained in the code. Code with unreadable design decisions that are not documented is very likely to be refused - even if those design decisions are good.

User Documentation

If your contribution contains a new feature, there usually needs to be at least some user documentation. MidPoint documentation is maintained in this wiki. Therefore it requires a separate contribution. If you discuss new feature beforehand (which you should) and if you keep communication (which you also should) you can get write access to wiki to contribute the documentation. New feature documentation usually contains two related, but slightly different wiki pages:

- Feature overview page (under Features parent page): This page describes the basic principles and motivation for the feature. It does not dive to configuration details. The purpose of this page is for the user to get basic understanding of the feature and to decide whether the feature is the thing what he needs or whether he needs something else.
- Feature configuration page (usually under Administration and Configuration Guide): Page that describes the details of feature mechanics, configuration, usage and so on. This page should contain configuration snippets, pointers to samples and so on.

No user documentation is needed for bugfixes. Smaller improvements are often OK with small updates to existing documentation.

Contribution Quality and Maintainability

Generally speaking, contributions should (at least) reach the average quality of midPoint code. But the quality requirement varies with the size and complexity of the contribution:

- **Small and simple** contributions: **average quality** is expected, but even lower quality contributions may be accepted.
- **Big and complex** contributions: **high quality** is required. We expect quality that is significantly above average quality of midPoint code.

This may look strange at the first sight. Contributor that submits big contribution has done a lot of work already. Why do we want him or her to do even more work? But, as always, we have very good reasons for this policy. It is all about maintainability of midPoint code and sustainability of midPoint project.

MidPoint core team is quite small. The team consists of professional, dedicated, full-time developers. MidPoint development is their day job. Even though the team is geographically distributed, good communication paths are established and maintained. Fluctuation is very low and most of the developers that started the project are still part of the team. Therefore if any of midPoint core developers discovers an issue with midPoint code, it is easy to track down the author, discuss the problem and find appropriate solution.This usually takes hours or even minutes. And this makes midPoint maintenance very efficient.

However, situation is very different for contributed code. Contributors are not part of midPoint core team. Communication with contributors is almost always slow and inefficient. Communication round-trip is very long: days or even months. Some contributors even disappear altogether. This means that we cannot rely on efficient communication with contributors.

When we accept a contribution to midPoint code base, we are also accepting responsibility for maintenance of the contributed code. If the contribution is small and simple, we are quite sure that maintenance overhead will be acceptable. Therefore we are willing to accept lower-quality contributions if they are small and their impact is limited. But for big and complex contributions we have to be more careful. We need to consider the effect of the contribution on overall maintenance effort. Also, big contributions are increasing risks, such as risk of instability, incompatibility risk, security risk or risk of leading that particular part of the system into a development dead end. Therefore we need to scrutinize big contributions much more carefully. And we have to insist on higher quality. Big contributions need to be perfectly readable, design decisions must be documented and the contribution must be covered with appropriate tests. Otherwise we risk that the contribution will become a maintenance burden and we will need to remove it. And then the whole effort of developing a contribution, accepting it, maintaining it and the finally removing it is completely wasted. MidPoint would be better off if we have refused the contribution at the beginning. Less work would be wasted - for everybody involved.

Therefore, if you plan to make big contribution please make sure that you understand the size and complexity of midPoint code and that you are not overestimating your abilities. In that case it is absolutely essential to **discuss the contribution** before you start any real work. And make sure that high quality standards are applied while developing the contribution. Otherwise the contribution may pose a risk for midPoint maintainability and we will have to refuse such contribution.

Tips and Best Practice

- **Do not submit each individual commit** unless the commit itself is a complete contribution. If your contribution is divided into several commits (which is perfectly fine) then send all the commits together so the maintainer can apply and test them together.
- If you have many commits but you want only to show them as one in the final midPoint history you might want to **squash** these commits to one. You can use git interactive rebasing to do this. (`git rebase -i`, the Git book provides more details)
- Provide a meaningful **commit message**. If the commit message is longer than a single line provide a **short summary of the message in the first line** and then provide more details in subsequent lines. Most git tools display just the first line of the commit message therefore the developers should be able to get an idea about the commit just from the first line.
- If there is an "issue" created in our bug-tracking system it is recommended to include issue identifier in the commit message.
- You may want to create a **topic branch** for larger contributions.
- There is **no code ownership** principle. Not in the midPoint development team and we do not provide that to the contributions as well. All code "belongs" to every developer and anyone has the "right" to modify any code. The only thing that we care about is the quality of the modification, not its origin. Therefore feel free to modify any code and fix bugs anywhere in the midPoint core or in any of the contributions. Just please make sure you know what you are doing. If you are not you are free to discuss that on `midpoint-dev` mailing list. If you contribute a code be prepared that others may modify it. If you do not want others to "ruin" you code then do not contribute it.
- MidPoint core team is trying to be quite careful about the state of the `master` branch in main midPoint repository. We try very hard not to break the build and we are also careful about passing tests and overall code quality. But this is software development and we are only human beings. Therefore it may happen that we break something occasionally. Therefore it is good idea to **check the state of the source code before pulling** from the main midPoint repository. The easiest way to do this is by looking at our continuous integration system. If it is mostly green then it is probably OK to pull changes. If it is too red it is better to postpone the pull for a while.
- Also write tests (e.g. integration tests) not just the main code. If you are fixing a bug try to write a test for the bug first and fix the bug second. For larger pieces of functionality try to create a fair amount of test code. Submit the tests as part of your contribution. You write the tests for your own good. As there is no code ownership anyone might (unintentionally) break your code. If you have good tests for the code the problem will be detected soon after the modification while it is still easy to fix. If you have no tests then you code will break without anyone noticing it for quite a long time. This will cause that your contribution might "corrode" over time and it may even be removed from the main code if its quality drops too low.
- For more details about contributing using git please see the "Distributed Git - Contributing to a Project" chapter of the Git book. In fact the whole book is more than worth reading.

Contributor License Agreements

MidPoint is dual licensed under Apache License 2.0 and European Union Public License 1.2. MidPoint users may choose any of those two licenses for their use of midPoint.

But the situation is more complicated for the contributors. While midPoint was single-licensed, the intent of a contributor to contribute under that license was quite clear. However, if users may to choose which license to use, contributors might be able to choose as well. And that may lead to confusion and uncertainty about midPoint licensing and other legal issues.

Therefore it is necessary to require contributor license agreements (CLA) from midPoint contributors. The purpose of the license agreements is to make licensing of midPoint code completely clear. Therefore if you submit a contribution to midPoint you will be asked to sign a CLA before the contribution can be accepted.

Credit

Git maintains the commit meta-data of the original commit. And this is what will be recorded in the history trail of main midPoint repository. Therefore the **original contributor will be recorded in each commit**. Apart from this the contributors are free to add their names to the appropriate place in the file header (e.g. Java `@author` annotation) if they feel their contribution is big enough to justify it.

Contribution Mechanics (Pull Requests)

Preferred way to make a contribution is to follow the pull *request procedure* on github. This method is quite simple, fast and straightforward. Old school developers may also use the traditional way and we will be perfectly happy to accept such contributions as well.

To start working on your contribution simply "fork" the project on github. Smaller contributions can be easily developed directly on `master` branch in your fork. For bigger contributions we recommend to create a new branch ("feature branch"). That's the same approach that we use for midPoint core development and it works quite well. When you are done with the contribution simply create new pull request on github. MidPoint core team will be notified, we will review the code, provide feedback, you will have the chance to improve the contribution and finally we will either accept or refuse the contribution. All of that can be done on github. For the old school developers the process is the same, but mailing list is used instead of github.

The code of midPoint core is currently in a single git repository. But there are other repositories that contain related code: connectors, clients, overlay projects and so on. Each repository has its maintainer (or maintainers). Maintainer is responsible for keeping the project in shape. Maintainer will make sure that the pull request is reviewed and that a decision is made at the end.

Please be patient when it comes to interactions with midPoint core team. Our day job is to develop midPoint. If you are planning a contribution then you are supposed to get higher priority than usual. But do not expect immediate response, especially at times when midPoint development is reaching crucial milestone and the time is tight. Therefore if you plan for your contribution to be included in a particular release then make sure the timing is appropriate. New feature contributions are accepted only during the development phase of midPoint (between start of new version development and feature freeze). New features submitted after feature freeze will need to wait until the development of a new version starts. Bugfix contributions can be accepted any time. But please do not leave the contribution to the last moment. It takes some time and effort to review the contribution. And, as you probably know very well, free time is a precious commodity especially before deadlines such as feature freeze or release. If you submit your contribution close to the deadline then the risk of postponing the contribution to the next release is very high.

Github, Gitlab and big evil corporations

Some people will certainly express concerns about our use of github. After all, github is a centralized platform. And as such, there are always concerns of abuse, monopolization and single point of failure. We are more than aware of such concerns. And we highly value project autonomy. Despite that we have decided that centralized platforms such as github are providing good value - if they are used in moderation.

We are using github to publish the code (git repositories) and to govern pull requests. We are **not** relying on github for anything else. We are not using github issues, wiki or any similar feature. And we do not plan to. Because we value our autonomy. Github was acquired by a certain corporation which has done questionable things in the past and there is no telling what will be done in the future. Therefore we need a freedom to evacuate our projects from github when things start to move in wrong direction. Git makes this easy, as migrating git repository is basically a question of a single *push*. Therefore we use github today, but it may be gitlab tomorrow and we may migrate to a completely self-hosted solution the day after. But the situation is quite different for github issues and wikis - those are not that easy to migrate. Therefore we do not use them at all. We still use github pull requests, though. Those provide very good value. And pull requests are temporary anyway - if handled correctly. Accepted pull requests are transformed into git history. And we do not rcare about refused pull requets that much. Therefore there is very little risk of losing pull request history. Everything we value is part of git and git is easy to migrate any time.

Issue Reports

Bug reports, improvement suggestions, feature requests and similar "issues" are appreciated, but they are not considered to be *contributions*. Except for one case: security vulnerability reports. Security vulnerability reports gets highest priority and they will be addressed immediately without any concern to who reports them. Overall, the issues will be processed using our priority system:

- Priority 0: security issues: always addressed immediately
- Priority 1: issues of platform subscribers: bug reports, improvements, new features (depending on development phase)
- Priority 2: issues of customers covered by midPoint support contracts: bugfixes only
- Priority 3: community issues

Please, follow our bug report guidelines when reporting a bug. But if you are not covered by a support agreement then do not expect that the bug will be fixed immediately - or that it will be fixed at all. There is no free lunch. Not even in open source. There are only two ways how to be sure that the bug will get fixed: purchase midPoint support or fix it yourself.

Do **not** report issues using github. If you do it anyway, then do not be surprised that we do not react. See the exaplanation above.

See Also

- Development Guidelines
- Development Participation