Before You Start

First, refer to the TypeScript Design Goals page and make sure your proposal fits within those guidelines.

Next, see if there are any related existing suggestions in the issue tracker to avoid creating duplicate work.

Defining a Scenario

A good design proposal starts with a problem statement. What are users having difficulty with? What common task do we need to make easier? Who is affected? Starting with several examples that show where the current language isn't meeting the needs of programmers will help make clear what problem is being fixed. Be sure to think about what the current workarounds are.

Outline the proposed change. Again, examples! You can think of TypeScript as three parts: a syntax, a type system, and a JavaScript emitter. Which parts are affected, and how? Are other language features impacted?

Finally, a formal description is useful in showing that a solution has been clearly defined. Provide an exact description of the rules you're proposing. List the steps in any new algorithms that the compiler would need to implement.

Language Feature Checklist

When writing your proposal, you'll need to keep in mind how it would interact with the rest of the TypeScript language. Specific things to consider (and document) are:

- Syntactic
  - What is the grammar of this feature?
  - Are there any implications for JavaScript back-compat? If so, are they sufficiently mitigated?
  - Does this syntax interfere with ES6 or plausible ES7 changes?
- Semantic
  - What is an error under the proposed feature? Show many examples of both errors and non-errors
  - How does the feature impact subtype, supertype, identity, and assignability relationships?
  - How does the feature interact with generics?
- Emit
  - What are the effects of this feature on JavaScript emit? Be specific; show examples
  - Does this emit correctly in the presence of variables of type 'any'? Features cannot rely on runtime type information
  - What are the impacts to declaration file (.d.ts) emit?
  - Does this feature play well with external modules?
- Compatibility
  - Is this a breaking change from the 1.0 compiler? Changes of this nature require strong justification
  - Is this a breaking change from JavaScript behavior? TypeScript does not alter JavaScript expression semantics, so the answer here must be "no"
  - Is this an incompatible implementation of a future JavaScript (i.e. ES6/ES7/later) feature?
- Other
  - Can the feature be implemented without negatively affecting compiler performance?
  - What impact does it have on tooling scenarios, such as member completion and signature help in editors?