TypeScript 4.7

- `resolveTypeReferenceDirectives` (both the services and global ts version) now accept an array of `FileReference`s as a first argument. If you reimplement `resolveTypereferenceDirectives`, you need to handle both the `string[]` and `FileReference[]` cases now.

TypeScript 4.5

- `factory.createImportSpecifier` and `factory.updateImportSpecifier` now take an `isTypeOnly` parameter:

```
- createImportSpecifier(propertyName: Identifier | undefined, name: Identifier): Import
+ createImportSpecifier(isTypeOnly: boolean, propertyName: Identifier | undefined, name
- updateImportSpecifier(node: ImportSpecifier, propertyName: Identifier | undefined, na
+ updateImportSpecifier(node: ImportSpecifier, isTypeOnly: boolean, propertyName: Ident
```

  You can read more about this change at the implementing PR.

TypeScript 4.2

- `visitNode`'s `lift` Takes a `readonly Node[]` Instead of a `NodeArray<Node>`

  The `lift` function in the `visitNode` API now takes a `readonly Node[]`. You can see details of the change here.

TypeScript 4.1

- Type Arguments in JavaScript Are Not Parsed as Type Arguments

  Type arguments were already not allowed in JavaScript, but in TypeScript 4.1, the parser will parse them in a more spec-compliant way. So when writing the following code in a JavaScript file:

```
f<T>(100)
```

  TypeScript will parse it as the following JavaScript:

```
(f < T) > (100)
```

  This may impact you if you were leveraging TypeScript's API to parse type constructs in JavaScript files, which may have occurred when trying to parse Flow files.

  See more details here.

TypeScript 4.0

- TypeScript provides a set of "factory" functions for producing syntax tree nodes; however, TypeScript 4.0 provides a new node factory API. For TypeScript 4.0 we've made the decision to deprecate these older functions in favor of the new ones. For more details, read up on the relevant pull request for this change.
- `TupleTypeNode.elementTypes` renamed to `TupleTypeNode.elements`.
- `KeywordTypeNode` is no longer used to represent `this` and `null` types. `null` now gets a `LiteralTypeNode`, `this` now always gets a `ThisTypeNode`.
- `TypeChecker.typeToTypeNode` now correctly produces a `LiteralTypeNode` for `true` and `false` types, which matches the behavior in the parser. Prior to this the checker was incorrectly returning the `true` and `false` tokens themselves, which are indistinguishable from expressions when traversing a tree.

TypeScript 3.8

- The mutable property `disableIncrementalParsing` has been removed. It was untested and, at least on GitHub, unused by anyone. Incremental parsing can no longer be disabled.

TypeScript 3.7

- the `typeArguments` property has been removed from the `TypeReference` interface, and the `getTypeArguments` method on `TypeChecker` instances should be used instead. This change was necessary to defer resolution of type arguments in order to support recursive type references.

  As a workaround, you can define a helper function to support multiple versions of TypeScript.

  ```
  function getTypeArguments(checker: ts.TypeChecker, typeRef: ts.TypeReference) {
      return checker.getTypeArguments?.(typeRef) ?? (typeRef as any).typeArguments;
  }
  ```

TypeScript 3.1

- `SymbolFlags.JSContainer` has been renamed to `SymbolFlags.Assignment` to reflect that Typescript now supports expando assignments to functions.

TypeScript 3.0

- The deprecated internal method `LanguageService#getSourceFile` has been removed. See #24540.
- The deprecated function `TypeChecker#getSymbolDisplayBuilder` and associated interfaces have been removed. See #25331. The emitter and node builder should be used instead.
- The deprecated functions `escapeIdentifier` and `unescapeIdentifier` have been removed. Due to changing how the identifier name API worked in general, they have been identity functions for a few releases, so if you need your code to behave the same way, simply removing the calls should be sufficient. Alternatively, the typesafe `escapeLeadingUnderscores` and `unescapeLeadingUnderscores` should be used if the types indicate they are required (as they are used to convert to or from branded `__String` and `string` types).
- The `TypeChecker#getSuggestionForNonexistentProperty`, `TypeChecker#getSuggestionForNonexis` and `TypeChecker#getSuggestionForNonexistentModule` methods have been made internal, and are no longer part of our public API. See #25520.

TypeScript 2.8

- `getJsxIntrinsicTagNames` has been removed and replaced with `getJsxIntrinsicTagNamesAt`, which requires a node to use as the location to look up the valid intrinsic names at (to handle locally-scoped JSX namespaces).

TypeScript 2.6

- Some services methods (`getCompletionEntryDetails` and `getCompletionEntrySymbols`) have additional parameters. Plugins that wrap the language service must pass these parameters along to the original implementation. See #19507

TypeScript 2.5

- `Symbol.name`, `Symbol.getName()`, and `Identifier.text` are all now of type `__String`. This is a special branded string to help track where strings are appropriately escaped and prevent their misuse. `escapeIdentifier` and `unescapeIdentifier` has been renamed to `escapeLeadingUnderscores` and `unescapeLeadingUnderscores` and had their types updated accordingly. Deprecated versions of `escapeIdentifier` and `unescapeIdentifier` still exist with the old name and type signature, however they will be removed in a future version. See #16915.

TypeScript 2.4

- The following types/namespaces are now string enums: `ts.Extension`, `ts.ScriptElementKind`, `ts.HighlightSpanKind`, `ts.ClassificationTypeNames`, `protocol.CommandTypes`, `protocol.IndentStyle`, `protocol.JsxEmit`, `protocol.ModuleKind`, `protocol.ModuleResolutionKind`, `protocol.NewLineKind`, and `protocol.ScriptTarget`. Also, `ts.CommandNames` is now an alias for `protocol.CommandTypes`. See #15966 and #16425.
- The type `EnumLiteralType` was removed and `LiteralType` is used instead. `LiteralType` also replaces `.text` with a `.value` which may be either a number or string. See String valued members in enums.
- `Declaration` does not have a `name` property. TypeScript now recognize assignments in .js files as declarations in certain contexts, e.g. `func.prototype.method = function() {..}` will be a declaration of member `method` on `func`. As a result `Declaration` is not guaranteed to have a `name` property as before. A new type was introduced `NamedDeclaration` to take the place of `Declaration`, and `Declaration` moved to be the base type of both `NamedDeclaration` and `BinaryExpression`. Casting to `NamedDeclaration` should be safe for non .js declarations. See #15594 for more details.

TypeScript 2.2

- `ts.Map<T>` is now a native `Map<string, T>` or a shim. This affects the `SymbolTable` type, exposed by `Symbol.members`, `Symbol.exports`, and `Symbol.globalExports`.

TypeScript 2.1

- `ParseConfigHost` now requires a new member `readFile` to support configuration inheritance.

TypeScript 1.9

- `LanguageService.getSourceFile` has been removed; `LanguageService.getProgram().getSourceFile` should be used instead.

TypeScript 1.7

- `ts.parseConfigFile` has been renamed to `ts.parseJsonConfigFileContent`

TypeScript 1.6

CompilerHost interface change (comparing to TypeScript 1.6 beta)

- return type of `CompilerHost.resolveModuleNames` was changed from `string[]` to `ResolvedModule[]`. Extra optional property `isExternalLibraryImport` in ResolvedModule interface denotes if `Program` should apply some particular set of policies to the resolved file. For example if Node resolver has resolved non-relative module name to the file in 'node_modules', then this file:
  - should be a 'd.ts' file
  - should be an external module
  - should not contain tripleslash references.

  Rationale: files containing external typings should not pollute global scope (to avoid conflicts between different versions of the same package). Also such files should never be added to the list of compiled files (otherwise compiled .ts file might overwrite actual .js file with implementation of the package)

TypeScript 1.5

Program interface changes

- `TypeChecker.emitFiles` is no longer available; use `Program.emit` instead.
- Getting diagnostics are now all centralized on Program,
  - for Syntactic diagnostics for a single file use: `Program.getSyntacticDiagnostics(sourceFile)`
  - for Syntactic diagnostics for all files use: `Program.getSyntacticDiagnostics()`
  - for Semantic diagnostics for a single file use: `Program.getSemanticDiagnostics(sourceFile)`
  - for Semantic diagnostics for all files use: `Program.getSemanticDiagnostics()`
  - for compiler options and global diagnostics use: `Program.getGlobalDiagnostics()`
    > Tip: use ts.getPreEmitDiagnostics(program) to get syntactic, semantic, and global diagnostics for all files

All usages of 'filename' and 'Filename' changed to 'fileName' and 'FileName'

Here are the details: - `CompilerHost.getDefaultLibFilename` => `CompilerHost.getDefaultLibFileName` - `SourceFile.filename` => `SourceFile.fileName` - `FileReference.filename` => `FileReference.fileName`
- `LanguageServiceHost.getDefaultLibFilename` => `LanguageServiceHost.getDefaultLibFileName`
- `LanguageServiceShimHost.getDefaultLibFilename` => `LanguageServiceShimHost.getDefaultLibFileN`

The full list of APIs can be found in this commit

The `syntacticClassifierAbsent` parameter for the Classifier.getClassificationsForLine is now required

See Pull Request #2051 for more details.

Changes to TextChange

`TextChange.start` and `TextChange.length` became properties instead of methods.

SourceFile.getLineAndCharacterFromPosition

`SourceFile.getLineAndCharacterFromPosition` became `SourceFile.getLineAndCharacterOfPosition`

APIs made internal as they are not intended for use outside of the compiler

We did some cleanup to the public interfaces, here is the full list of changes: - Commit 2ee134c6b3c0ec
- Commit 35dde28d44122c
- Commit c9ef4db99ac93bb1c166a
* Commit b4e5d5b0b460cc88a10db

`typescript_internal.d.ts` and `typescriptServices_internal.d.ts` have been removed

The two files exposed helpers in the past that were not part of the supported TypeScript API. If you were using any of these APIs please file an issue to re-expose them; requests for exposing helper APIs will be triaged on a case-by-case basis.

For more information please see the full change.