Versioning

Applicability

These policies apply to all documents, products, components or web services provided by DTI as part of TaxCore product offering.

Versioning

This topic explains versioning policy, which is based on the semantic versioning standard.

To learn more, read Semantic Versioning 2.0.0 (http://semver.org/)

What is semantic versioning?

In summary, the semantic versioning standard specifies that each version should be uniquely labeled by an identifier made up of three components:

- **major** version number
- **minor** version number
- **patch** version number

These components are separated by periods. For example: The version 1.2.3 has a **major** version of 1, a **minor** version of 2, and a **patch** version of 3.

For **pre-release versions** , we may suffix the version with an identifier indicating the version's pre-release status, like **-beta1** or **-alpha2**.

When we release a new version, we increment one of the major, minor, or patch components. Differentiating between new versions is based on the kinds of changes introduced in the new version.

As is conventional in the semantic versioning standard:

- The **major** version component increments when the version contains **breaking changes**.
- The **minor** version component increments when the version contains **new functionality that is backwards compatible**.
- The **patch** version component increments when the version contains **backwards compatible bug fixes**.

Understanding Versioning Policy

The semantic versioning standard is a set of guidelines, not rigid rules. Different products and companies interpret the standard in ways that make sense to them.

We version our APIs based on the following criteria:

What qualifies as "documented" behavior?

"Documented" behavior is behavior that is referenced or explained in technical and user documentation we provide about TaxCore, or behavior that exists in the public API.

Modifying unintentional behavior (patch version increment)

We release a patch version to modify a behavior if correcting that behavior does not change any documented types, properties, methods, or parameters.

We release a **patch** version to modify a behavior when:

- the behavior is unintended and does not work as documented (a "bug"), or
- the behavior works as intended at the time of release but is later found to cause problems

Adding new functionality (minor version increment)

"New functionality" is not a term that applies to all new behavior.

It means providing you the ability to do something with the SDK that you could not do before and that involves a new type, property, method, optional parameter, or supported parameter value.

New functionality qualifies as a **minor** version release.

Introducing breaking changes (major version increment)

A "breaking change" occurs when a type, property, method, parameter, or allowable parameter value is no longer defined or no longer produces the results or behavior you want when you use it according to the documentation we provide.

In cases like this, internal methods or properties can exist that DTI cannot prevent application code from accessing, including methods and properties which are excluded from all documentation and from explicit interface declarations. We consider methods and properties like that, such as TypeScript declarations, to be internal. We do not consider breaking these internal references to be breaking changes if the underlying behavior persists. We will not release a major version solely to resolve an internal breakage.

Examples of breaking changes that would qualify for a **major** version release include:

- the application code no longer compiles (in a compiled language)
- the application or SDK is unable to do the thing you want it to do, even when you use it correctly

Alpha and canary versions (alpha and prerelease suffixes)

We provide alpha versions of React as a way to test new features early, but we need the flexibility to make changes based on what we learn in the alpha period. If you use these versions, note that APIs may change before the stable release.

Commitment to Stability

Hundreds of developers are building, testing, accrediting, and maintaining E-SDC and POS solutions that depends on different TaxCore APIs. In addition to that, dozens of applications are depending on TaxCore.API.

Breaking changes are inconvenient for everyone, so we try to minimize the number of **major** releases. Instead, we release new features in minor versions. That means that minor releases are often more interesting and compelling than majors, despite their unassuming name.

As we change TaxCore over time, we try to minimize the effort required to take advantage of new features. When possible, we'll keep an older API working. That means we need to make it as easy as possible to upgrade to new versions; if we make large changes without a migration path, people will be stuck on old versions.

Supported versions

TaxCore APIs and SDKs follow End of Life (EOL) policy to determine each version's maintenance window. The end-of-life policy also defines when DTI can drop support for older versions of their underlying platforms.