The *Eclipse Quality* (http://www.eclipse.org/projects/dev_process/eclipse-quality.php) document details a range of API types:

	Specification	**Test Suite**	**Implementation**	**Clients**	**Support Promise**	**Package**
Platform API	yes	yes	yes	yes	yes	public
Provisional	yes	yes	yes	yes	*not quite*	public
Incomplete	*incomplete*	*incomplete*	yes	yes	yes	public
Experimental	yes	yes	yes	yes	?	internal. provisional
Non-API	-	-	yes	-	none	internal

For DTP 0.7, a number of these choices are not possible:

1. *Platform API:* Because, by the Eclipse process, a project first has to build a community of extenders (clients) outside the project to validate API. We might be able to get *some* API to this stage by DTP 1.0, but acceptance by the Eclipse community is hard to attain for these.
2. *Incomplete:* These can only exist during project implementation, and cannot be propagated to a release.
3. *Non-API:* Obviously not a choice.

This leaves two API levels: *Provisional* and *Experimental*. (Why the package name for *Experimental* is "internal.provisional" is not clear to me, but that doesn't really matter at the moment.) The difference between the two is only the support promise and the package names. I believe that the goal of exposing some form of API in DTP 0.7 is to give the community direction about where we think API *might* go, so they can focus their review efforts in the correct areas. In a sense, we should make a suggestion to the community about where we see DTP API moving toward, as a method of involving a wide audience in their development, adoption, and evolution. For these reasons, I suggest that we aim for *Provisional* API statements in DTP 0.7.

Declaring *Provisional* API is not automatic based on meeting the conditions in the table above (specification, test suite, etc.). Likewise, not completely meeting a condition or failing to meet a condition at all is not immediate cause to invalidate a *Provisional* API statement. Rather, the statement is subject to community review during the release presentation(s). We need to state what has been done, and what has not been done, to meet the *Provisional* API criteria. We also need to state how we plan to deal with API going forward. If the community is satisfied with our presentation, then the *Provisional* API declaration stands.

Based on the current DTP code line and supporting collateral, I believe the status for each condition is:

1. *Specification:* We have some extension point and code API specification, in varying degrees of completeness. We'll need to have a fairly complete set of specifications for the release review.
2. *Test Suite:* We have little, if anything, for this yet. I understand that the community will emphasize this requirement, so it is likely to represent the most work for our API declaration. It is likely that the work associated with implementing test suites will be the limiting factor in the number of API declared in DTP 0.7.
3. *Implementation:* Required for all API statements, and we should be able to meet this easily, since we are inductively defining API.[1]
4. *Clients:* Ideally we would have multiple clients using DTP only at the API level. We will not have such clients in DTP 0.7. We will have DTP-internal clients, however, and this is the minimal starting point. I understand that the community likely will be flexible, given that we are trying to engage them (i.e. create more clients) by the very act of the *Provisional* API declaration.
5. *Support Promise:* We make not claims around this, other than the vague idea that this is the general direction DTP might evolve toward, given community feedback.
6. *Package: Provisional* API need to appear in "public" packages. We can keep non-API "public" packages, but we should mark them as "Non-API" in the JavaDoc.

[1] I'll classify API development as either *inductive* (code first, API emerge) or *deductive* (API first, code based on specification). The Eclipse preference is for deductive API (see Jim des Rivieres' presentation at EclipseCon 2005 for details: http://www.eclipsecon.org/2005/presentations/EclipseCon2005_12.2APIFirst.pdf). In a perfect world, we'd have a purely deductive API process in DTP. Clearly, we are not going to achieve this in DTP 0.7 or 1.0, but perhaps it is a goal that we can strive for in later versions.