"Tags" implement a feature which is a new type called "NameTags".  The concept is that you create a new "NameTags" instance like so:  
  
new NameTags(TestNameTags);  
  
You can then perform the following actions:

* createTag - Create a tag given a name.  Returns you a tag Id which will never change for this name. (tag names can have spaces)
* deleteTag - Delete a tag by Id.
* renameTag - Rename a tag giving the new name.  The tag Id stays the same.
* getTagCount - Gets the total tag count.
* getTagName - Gets a tag name by tag Id.
* getTagId - Gets a tag Id by tag name.
* getAllTags - Gets a formatted list of tag Id / tag name for all tags.  This is separated with "\n" as tag names can have spaces.

The concept is that you can create a set of tags by creating a "NameTags" instance.  You can then add/delete/rename as many tags as you like.

The most important part is that other objects can now refer (any way you like) by tag Id which always stays the same, even if the tag name changes.  This is extremely important as it means you can rename a tag and not have to change potentially hundreds of references.

I add a "NameTagsTest.cs" file to the unit tests folder to show how it works.  It also persists a "NameTags" instance to a file named "savedNameTags.cs".  There's also a section at the top that, if you un-comment, it will load the persisted form of a "NameTags" type which, in the example, looks like this:

//--- OBJECT WRITE BEGIN ---  
new NameTags(TestNameTags) {  
      TagCount = "4";  
      Tag0 = "1\nEnemy";  
      Tag1 = "2\nPlayer";  
      Tag2 = "3\nScenery";  
      Tag3 = "5\nSpecial FX";  
};  
//--- OBJECT WRITE END ---

The above list is comprised of 4 tags:

Tag Id #1 = "Enemy"  
Tag Id #2 = "Player"  
Tag Id #3 = "Scenery"  
Tag Id #5 = "Special FX"

Note that there is a gap (no #4) because in the test-bed, I delete that entry.

---

So, assuming that the project has one of these name-tags instances, the project can add as many tags as it likes.  Then, other objects can refer to the tags by tag Id.  For example, let's say an image-map refers to "Scenery & Special FX" it would be:

new t2dImageMapDatablock( test ) {  
    ...      
    NameTags = "3 5";      
    ...      
};

Single object helpers:  
> tag  
> untag  
> hasTag  
  
Multi-object query:  
> queryTags  
  
Simple added any objects you wish to the NameTags (it's a SimSet now) and when using "quertTags", it'll search all those objects for you which is much faster than iterating using TS.  
  
Use the single object helpers to add/remove and check for the presence of tags on individual objects.  This uses a hard-wired field name of "NameTags" to perform the operations.  
  
All the above methods allow the specification of multiple tags if required.