Roundtable White Paper
Working With Purchased Applications

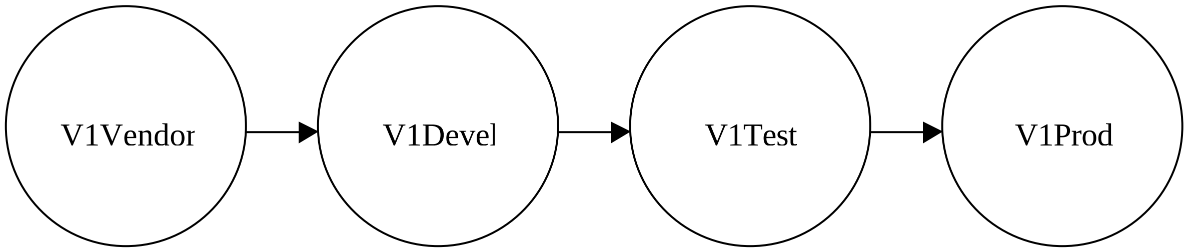
Introduction

It is fairly common for a company to purchase an application from a vendor then highly customize it. When the vender sends you a patch, you need to integrate it into your custom version of the application. This is facilitated by Roundtable's management of object variants (customized objects). You are able to generate a list of all objects changed by the vender, promote these changes forward through your development life cycle without overwriting your custom versions, and generate a list of objects changed by both you and the vendor.

Implementation

Loading the Vendor Application

In the most common scenario, you use four workspaces: a generic "vendor load" workspace, a development workspace, a QA workspace, and a production workspace. In the following diagram, the example workspace names are prefixed with a "V1" as the example application is version 1:



After defining the above workspaces (and sources shown by the arrows) the purchased application is loaded into "V1Vendor" (see the white paper entitled "Loading Your Application"). This release is then imported (promoted) forward to "V1Devel", to "V1Test, and then to "V1Prod".

Loading the Existing Customizations

All work performed at you site will be treated as a customization. All objects loaded in the "V1Vendor" workspace are assigned product modules from a set of vendor Pmods. You must also define a set of custom Pmods for your WSmods (remember that a WSmod is the group that defines the directory for objects and a Pmod is just an attribute of an object).

Example WSmod and Pmod definitions

WSMods	Directory	Vendor Pmods	Custom Pmods
code	inv/code	VenCode	CstCode
includes	inv/includes	VenIncludes	CstIncludes
trig	common/trig	VenTrig	CstTrig

The “V1Devel” workspace currently has only the generic vendor application in it (imported from “V1Vendor”). Either change the “V1Devel” workspace path to point to your customized version of the application or overlay your custom version in the current directory. Use Module load to load in all new objects (specifying only the “Cst” Pmods). Use the Group Check Out’s Global Change Finder to find and check out all objects that you have customized – specify that the Group Check Out should give all the objects “Cst” Pmod when checked out. When you complete the task, you now have all customizations loaded into the “V1Devel” workspace. You can cut a release and move it to “V1Prod”.

Future Development

You do not check out objects flagged with a Vendor Pmod. You first change the Pmod to a “Cst” Pmod (see the white Paper entitled “Managing Customizations” and the Roundtable User’s Guide for more information on how to create object variants - change the Pmod attribute). We recommend that you apply non-primary security to RTB to keep developers from accidentally checking out objects without giving them a Cst Pmod.

Loading a Vendor Patch

When you receive a patch from the vendor, apply it to the “V1Vendor” workspace directory. You then use “Module Load” to load in new objects and use the “Group Check Out’s Global Change finder” to find changed objects (leaving the Pmods as “Ven” Pmods when checking them out). After completing the task and creating a release, you can run the release report to generate a list of EVERY changed and new object (something even the vendor may not have).

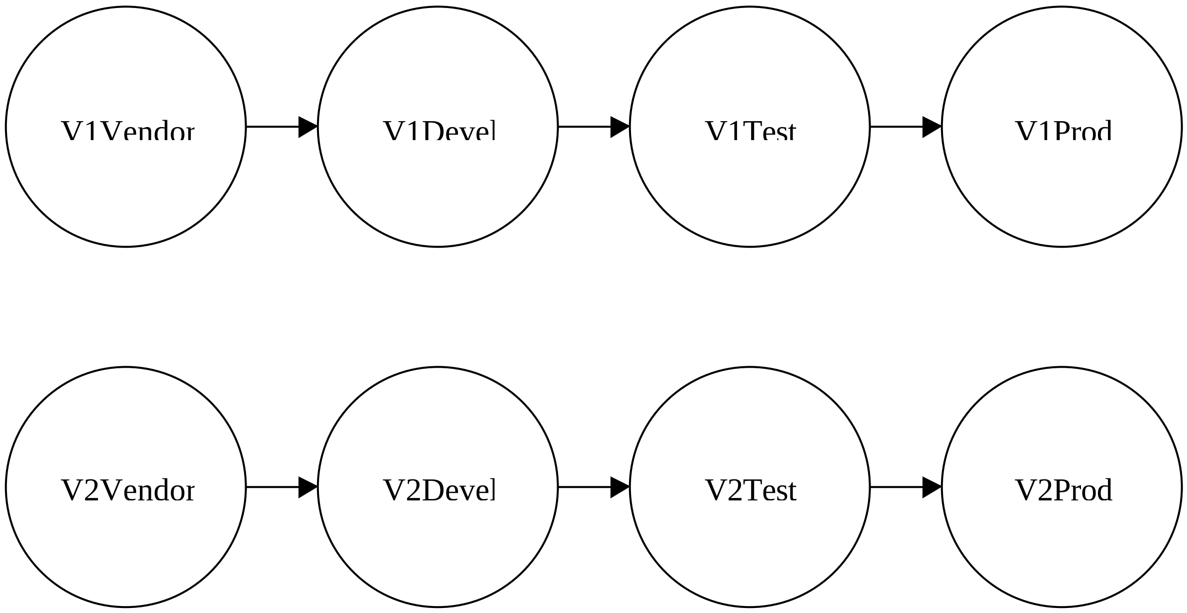
When you import the release into “V1Devel”, only the objects that you have not customized will be imported. Objects that have changed in both “V1Devel” and “V1Vendor” will be marked as “EXC” in the import list (excluded) and will not be imported. You should then use the release report generated in “V1Vendor” to see all objects that have been changed both you and the vendor so that you know what changes nee to be merged (see the white paper entitled “Managing Customizations” for directions on using the release report to show objects that need changes merged).

Parallel Development

Multiple Versions of the Vendor App

You may also get into a situation where the Vender sends you a new version of their app that either has too many changes to implement at once or runs on the next Progress version. In this situation, you may want to have a new environment to work on the new version of the application while still maintaining the original environment for fixing bugs, etc.

Roundtable allows the implementation of parallel development cycles. This means that you can set up a work environment for working on separate “version branching” of your application. The complete environment is shown in the following diagram:



Implementation

The desired goals are:

* To obtain a list of every object that has changed between V1 and the new V2 of the Vendor App.
* To find all objects in that list that you have also customized.

After defining the new workspaces (and sources shown by the arrows above), the first step is to bring the V2 vendor environment up to the V1 state (we’ll need the V1 base-line for the release report later). To do this, create a temporary source from “V1Vendor” to “V2Vendor” and import the last release. You can then remove the source and cut a release in “V2Vendore” to mark the V1 base line.

Next, define a temporary source from “V1Devel” to “V2Devel” and import the latest customized version of the Application. You can then remove the source and cut a release. You now have the latest “customized” version of your customized application in “V2Devel”. Import it to “V2Test” and “V2Prod”. We’ll pretend (for the moment) that there wasn’t any half completed work in “V1DEVEL” or “V1Test” waiting to be moved into production.

Loading the New Vendor Version

You can now load V2 of the vendor application. Either change the “V2Vendor” directory to point to the new V2 of the vendor app or completely delete and replace the app in the current workspace directory (do not overlay V2 on top of V1 as you want the global change finder to find objects that the vendor removed from the application). You can now use module load to find all new objects and use the Group Check Out’s Global Change Finder to find all changed and deleted objects (leave them in the Ven Pmods when being checked out). After completing the task and creating a release, you can run the release report to see all objects changed between V1 and V2 (something the Vendor may not even have). Remember to use the load schema to pull in any new schema changes from the vender database.

When you import the release into “V2Devel”, only the objects that you have not customized will be imported. Objects that are object variants in “V2Devel” and have changed in “V2Vendor” will be marked as “EXC” in the import list (excluded). You should then use the release report generated in “V2Vendor” to see all objects that have been changed both you and the vendor so that you know what changes nee to be merged (see the white paper entitled “Managing Customizations” for directions on using the release report to show objects that need changes merged). This list may be quite large if you have changed many objects and the vendor has changed many of the same objects.

Version Branching

You now have two separate workspace flows (or two development life-cycle flows, or two promotional models, or whatever terms you wish to use) defined. You can patch the old version in “V1Devel” without interfering with the merging of changes in “V2Devel” from “V2Vendor”.

Since you will want patches from “V1Devel” to get into “V2Devel” as well, I recommend that you use version branching (described in the white paper “Managing Customizations”). In a nutshell, you only create “patches” in the

"V1Devel" workspace. A patch is the smallest versioning unit. It is the "01" if you have version "031601" (read 03.16.01 or Version 03 Revision 16 Patch 01). If the same object has not been touched in "V2Devel", you can just assign the object patch you created in "V1Devel" to "V2Devel".

If the object has been changed in "V2Devel" before you made the change to the same object in "V1Devel", then you need to merge the changes in. Always create either revisions or versions in "V2Devel" to keep the branches simple. For example:

V2Devel	V1Devel	Work Performed
02.12.00	02.12.00	Same version in both to start
02.12.00	02.12.01	Patch made in V1Devel
02.12.01	02.12.01	Patch can just be assigned to V2Devel
03.00.00	02.12.01	V2 specific change made in V2Devel – you are starting different branches in the workspaces now
03.01.00	02.12.01	Another V2 specific change made in V2Devel
03.01.00	02.12.02	Fix made in V1Devel
03.02.00	02.12.02	Same fix also made in V2Devel

You do not have to only patch in V1 and not use patch in V2, but this policy makes the changes' origins very obvious just by glancing at the version.

Don't forget about the changes that were in "V1Devel" and "V1Test" that we ignored while you were loading the V2 application into RTB. Either assign them to "V2Devel" or also make the change in the "V2Devel" version. From that point on, all changes done in "V1Devel" have to be assigned to "V2Devel" or merged into "V2Devel" objects as described above.

Two Versions of Progress

If the two versions of the vendor's app also require two different versions of Progress (V8 and V9 for example), you follow all the same steps. You leave the repository as V8 so that both V8 and V9 Progress clients can connect to it. When you are working in the V1 workspaces, you use a V8 Progress/RTB client. When you are working in the V2 workspaces, you use a V9 Progress/RTB client (just as you would without RTB).

The schema changes applied to the repository for V9 RTB do not harm V8 RTB's ability to operate against the repository

Conclusion

At some point in time, your V2 environment will be ready for production and you can retire the V1 workspaces. As your development effort slowly changes to focus on mostly the V2 application, you will find that it is easier to make all changes in “V2Devel” and either assign the version to “V1Devel” (if possible) or check out and patch the object in “V1Devel” with the same change.

This white paper described a process that must be followed with or without RTB. RTB adds the accountability to the process to allow you to easily generate information that may be impossible to acquire otherwise. You easily see what the vendor changes and what changes you must merge, reducing the analysis time to a fraction of what it would take without RTB.