Roundtable White Paper
Loading Your Application into Roundtable

1 Summary
This document provides helpful-hints on how to go about loading your application into RTB (Roundtable), above and beyond those in the Roundtable User's Guide. It is assumed that you already have defined your product and module structure, your subtypes, as well as your workspace flow.

The biggest helpful-hint for getting started with Roundtable is, "purchase consulting." The jump-start you will get from having an experienced consultant load your application and to train your developers is immeasurable. Only if you have a staff of very experienced Progress developers and already have an understanding of Roundtable should get started with Roundtable on your own.

2 Loading Your Application
Loading you application consists of 5 steps:
1. Loading the first version of your base application and schema.
2. Importing it through your workspaces
3. Checking in any custom variants.
4. Creating a release in the production workspace(s) to match your commercial version.
5. Load in the next completed commercial version(s).
6. Loading in the current release under development and checking out code currently WIP.

Backing up your repository in between each step may save you a headache later!!!!

2.1 Loading the First Version of Your Application
By loading in each commercial version of your application, you will be able to create deployment sites that will match the releases that your customers are currently on. This gains you the benefits of Roundtable's incremental deployments right away. Keep the following points in mind when using module load:
1. Module load is used to load in your base application, not custom variants. If module load finds a program in a second directory that has the same name as a program already used in Roundtable, it assumes it is a unique object that happens to share the same file name. It informs you that you must create an alias for the object. Checking in custom variants is covered later.
2. Module load looks in the module directories and the <module directory>/<subtype directory> only. If you have additional objects in directories below the module directory, they will not belong to the module unless they are in a subtype directory. If you have objects in subdirectories other than described above, then you need to seek assistance on re-defining the structure of your Product modules within Roundtable.
3. The "Module Load" utility loads all objects under a single task. If you have some objects that are WIP, do not load them at this time. By only loading completed objects, you will be able to label the task "load application version x" and complete it right away.

Module load is accessed using the Tools->Module load menu item in GUI RTB, and by choosing the LOAD menu item from the "Module Selection" screen in TTY RTB. Both the GUI and TTY manuals contain complete instructions on how to use the module load utility. Load it only into your primary development workspace. You will import it through to the other workspaces later.

To load your database, choose “Load Schema” by right clicking on the DBASE tab folder for the database in GUI RTB and by choosing the LOAD menu item after selecting the database object in TTY RTB. Keep the following in mind before using schema load for the first time:

1. The entire load is performed under a single task. You should create a task just for the schema load so that you can complete it immediately after loading the schema of your database.
2. The database loading utilities do not support field domains and will preface each field object name with the table name. This means that the “cust-num” field in the “customer” table will have an object name of “customer.cust-num” and the “cust-num” field in the “order” table will have an object name of “order.cust-num”, making them two unique objects.
3. Table domains are also not supported by the DB load. You may have a table name that is the same in two separate databases. If you have a table named “customer” in a database named “sports”, then try to load in the “demo” database that also has a “customer” table, module load will fail telling you that there is already a table object named “customer” in Roundtable. To handle this, the schema load utility gives you the opportunity to use a code-prefix for naming your objects. This tells module load to add the prefix to the object names. If you use a prefix of “demo”, then the table will be named “demo.customer” and the fields would be named like “demo.customer.cust-num”. If you know that you will have to use a prefix, then you should use it for all databases in order to keep consistency. It is easier to find your schema objects with consistent naming similar to “sp.customer.cust-num” and “demo.customer.cust-num” rather than making developers remember that a plain “customer.cust-num” belongs to the sports database.
4. When not using field domains, putting all objects that belong to a single database in separate modules simplifies finding and editing schema objects.

If you wish to use field and table domains:

1. You cannot use the database load utility – you must create you schema within Roundtable.
2. You should have all database objects in a single module since an object can only exist in a single module within a workspace. A field or table domain is a single object assigned to multiple database or table objects.

2.2 Import Through Your Workspaces

This is the easy part. Create a release in the primary development workspace, then go to the target workspace and import the release. Update your schema and create a release to be imported into the next workspace. Compile the workspace using selective compile to bring the .r code up-to-date. Complete directions on how to perform an import are in the Roundtable User’s Guide.

Be sure to import the entire base into your custom workspaces as well. Do not worry about custom variants at this time.

2.3 Checking in Any Custom Variants

After loading your base application and importing it through all your workspaces (including the custom workspaces), you may then manually create or use the global change finder to create your custom variants in your custom workspaces. The recommended setup is to keep the same Workspace module (and therefore directory) for the base and the custom product modules. Since you are keeping the customized configuration in a separate (custom) workspace, you are then able to create deployments that have the same Propath and other requirements as you main product (or other customized configurations). Alternatively, the customized procedures would have a product module mapped to a different workspace module (and therefore different directory), deleting the base procedure from its directory. In either situation, the object is given a new version number (010000) for its new life in the variant product module.

To manually create the custom variants, you create a new object using the same name as the base object but in the custom product module. Roundtable will warn you that it already exists and ask you if you want to move it to the new module. After creating all your variants, replace the physical objects with your customized procedures (on the disk drive). You can now complete the task to check in all the variants.

To use the global change finder, you replace the objects on the disk drive before creating the variants in Roundtable (it is recommended to run the global change finder first to make sure you don't accidentally have changed code in your workspace). In TTY RTB, you use WS-CONFIG->GLOBAL CHANGE FINDER to build the list of changed objects. In GUI RTB, you chose TOOLS-> GROUP CHECKOUT and choose the "select changed" button in the group checkout dialog to build the list of changed objects. If you are using a different subdirectory for the variants, you will be presented with the dialog asking if the object should be considered a deleted object. You must choose "no" to treat it as a changed object.

The global change finder has now built a list of the changed objects. In GUI RTB, under Check out, you must select to check the objects out "as group". This enables the "check out options". In the "check out options", you select the new module to move the object to. In TTY RTB, you enter the name for the new module to move the object to. You are now able to complete you task to check in the variants.

Note: Future imports into the workspace will not overwrite your variants with the modified base objects.

2.4 Create a Release in Each of Your Production Workspaces

Create a release in each of your final workspaces (both production and custom workspaces). Label this release to match the commercial version of your application that you loaded. Your Roundtable deployment sites will now be able to reference this release. When you create or load in the next version of your app, you can build an incremental deployment from one release to the next.

2.5 Load the Next Commercial Version of Your Application

Delete the physical programs in the directories in your primary development workspace that contain your application and replace them with the next version of your application (you delete first in case some files were removed in the next version). Use module load again to load any new programs. Module load is only loading objects that did not exist in the first release.

Use the global change finder to check out all objects that changed between the first and second release of your application. In TTY RTB, you use WS-CONFIG->GLOBAL CHANGE FINDER to build the list of changed objects. In GUI RTB, you chose TOOLS-> GROUP CHECKOUT and choose the "select changed" button in the group checkout dialog to build the list of changed objects. If a program has been deleted, the change finder will ask you if you wish to delete the object. After the global change finder checks out all objects that have changed, you can complete the task to check them in. The global change finder will also find objects that were removed from the physical directories and ask if they would be removed from the workspace.

Replace the workspace databases with the next version of the databases. Use the Schema load utility to load the changes (the same as you did when loading your initial commercial release). If you are using field domains, you will have to make the changes manually within Roundtable.

Follow the same steps to create a release and import through your workspaces. If you have any custom workspaces with custom variants, they will not be overwritten by the imports. Copy the directories with the next release of your custom variants on top of the custom workspace directories and use the global change finder to check out the ones that have changed (the same way you found the changed objects in your base application in this step). If there are new custom variants, use the module load utility to load them in (the same steps you followed during the load of new objects in your base application in this step).

2.6 Load the Current Release and Check out Code that is Currently WIP

Finally, copy any COMPLETED source files that are part of your current uncompleted release on top of your primary development directory. Do not copy over programs that developers are currently modifying. Use global change finder to check out all objects that have changed, module load to load new objects, then complete the task to check them in (the same as you did in the step above).

You now have Roundtable up to date on everything except objects that are Work In Process (WIP). Have your developers create tasks and check out all objects that they are currently working on. They can then replace the objects with the modified versions. This way, you are able to immediately start entering tracking information such as task descriptions and object version notes. If used the global change finder to check them all out under a single task, you would not be able to complete that task until every object was completed and the developers would not have access to the checked out objects.

3 Hints on Using and Modifying Module Load

Module/code load is supplied in source; "rtb0282.p" in UNIX and "rtb_mdld.w" in Windows. In its generic form, Module Load scans a directory specified by a workspace module for files. Files that have not already been defined as PCODE objects in your workspace are stored in a temp-table (read the chapter in the Roundtable User's Guide on module load). By browsing through this table, you can choose the files needed to create PCODE objects.

3.1 Hint 1 - Understanding loading aliased objects

Two code objects cannot have the same logical name in RTB. Roundtable uses the file-name for the object name by default. If an object is found that has the same name as an object already loaded into Roundtable, you are presented with a dialog box asking what alias name you want to give the object (see the section in the user's guide on alias objects). This can get tedious if you have many objects with duplicate names. Consider this example:

Module	Directory	File names
Ap	Src/ap	Program.p Window.w Include.i
Ar	Src/ar	Program.p Window.w Include.i
Inv	Src/inv	Program.p Window.w Include.i

When you load the “ap” module, module load will create three objects; “program.p,” “window.w,” and “include.i.” When you load the “ar” module, processing stops and module load asks you what alias name to give the three objects since it cannot use the file names again.

Alias names always start with the “@” symbol. One way to name the objects would be “@program-ar.p,” “@window-ar.w,” and “@include-ar.i” (the logical alias name does not affect the physical name of the file). When you load the inv module, processing stops again on each object. You could name the objects “@program-inv.p,” “@window-inv.w,” and “@include-inv.i.”

Module	Directory	File Name	Object Name
Ap	Src/ap	Program.p Window.w Include.i	Program.p Window.w Include.i
Ar	Src/ar	Program.p Window.w Include.i	@program-ar.p @window-ar.w @include-ar.i
Inv	Src/inv	Program.p Window.w Include.i	@program-inv.p @window-inv.w @include-inv.i

NOTE: I built the alias-object name by placing the module-name after the file-name so that you can easily search for the object. If you searched for an object named “@prog,” Roundtable would present you with a list; “@proram-ar.p,” “@program-inv.p.” Also not that there are no leading or trailing slashes on the directory path.

Stopping hundreds of times to enter a name for each aliased object in every module-load can be very time consuming. Luckily, you can modify the module load source to automate the loading of aliased object. Choose a standard naming process (such as the one described above), take out the UI portion of the alias logic, then automatically build the unique alias object-name.

3.2 Hint 2 – Never, Never, Never

* Never Number One - Do not attempt to modify module load to load multiple modules at the same time, unless you want a headache. Leave it as a module load utility rather than a honking-huge app-load utility.
* Never Number Two – Do not attempt to load in objects at a higher version number than 010000. There is too much internal logic in RTB that is based upon objects being born as version 1.
* Never Number Three – Do not put leading or trailing slashes on the directory path to an aliased object. A leading slash will make Progress look off of the root for the file rather than the workspace path. A trailing slash will cause cross-referencing of objects to get confused.

3.3 Hint 3 – Code Subtypes with Multiple Parts

RTB allows you to define code subtype with multiple parts (read the section in the RTB User’s Guide on Code subtypes if you want to understand this section). For example:

You might define a subtype named “procedure” that has one part with an extension enforcement of “.p”. You might also have subtype named “formbld” that has multiple parts; a “.p” extension enforced for the main part, a “.i1” extension enforced for the second part, and a “.i3” extension enforced for the third part (ex. “main1.p,” “main1.i1,” “main1.i2” is all one object called

“main1.p” in RTB). You might then have a subtype named “include” that has no extension enforcement (the extension can be anything).

You then want to load a directory (workspace module) that contains the following files:
Mainscreen.p
Inform.p
Inform.i1
Inform.i2
Maininc.i

You would want to create RTB objects for “Mainscreen.p,” “Inform.p,” and “Maininc.i” only. When you check in “inform.p”, RTB will check in the entire logical object (including the “inform.p,” “inform.i1,” and “inform.i2”).

The generic version of module load looks at every file as an individual object. It would flag “inform.i1” and “inform.i2” for load as “include” type objects by default (because it does not pre-process the list of objects looking for subtype parts). Since you only want to load the main object, you would have to make sure that you toggle the parts “inform.i1” and “inform.i2” not to be loaded. This would be very time consuming for hundreds or thousands of objects. You would want to modify module load to preprocess the list so objects are only created for main object parts.

4 Using Multiple Version of Progress

4.1 Mixing V8 and V9 Progress

V9 Progress does NOT allow you to update the schema of a V8 database. This means that you cannot use a V8 database (even running a server) as a workspace database in a V9 workspace if you plan on changing the schema. You must be running a V8 (Progress and Roundtable) client in order to update the schema of a V8 application/workspace database.

If you wish to mix and match V8 and V9 workspaces in a single repository, you will have to leave your repository as V8 so that your V8 clients can connect to it.

This makes installing V9 Roundtable against a V8 repository a little more difficult. You must run the first half of the V9 _update.w (the schema update) using V8 progress. This is because there are schema changes required by V9 Roundtable that you must apply to your V8 repository. After applying the schema update, you then quit the install, restart it using V9 Progress, skip past the schema update (that you already completed), and perform the V9 compile of Roundtable against the V8 Repository.

V9 Progress has not known difficulty writing data to a V8 database.

Other Considerations:

1. V9 RTB uses the new "raw" data-type to store binary files in the repository (removing the previous 1 MB limit). The side affect is that a V8 RTB client cannot read or manipulate a binary file that was checked in using V9. V8 will expect the binary to have been uuencoded and stored in a rtb_repo record as text.

2. Another interesting side-affect is that you cannot add a user to a V8 repository using a V9 RTB client. You get the message, "A version 9 client may not modify schema on a version 8

database." It appears Progress considers modifying any "_" table to be a schema change, and since we store users in the "_user" table. Connect with your V8 client when you need to add users.

4.2 Mixing Three Versions of Progress

Progress only allows users to connect to a database of the same version or one version older than the client. Since the Roundtable repository is a Progress database, this makes developing in V6, V7, and V8 (or any three versions) a little tricky to set up.

You would wan to follow the standard implementation of a mixed V7 and V8 environment. Use a V7 database server for the repository, then set up a V7 and V8 workspace. You would only select the V7 workspace when using a V7 Progress/RTB client. RTB would connect to the V7 workspace databases for you. You would only select the V8 workspace using a V8 Progress/RTB client. RTB would connect you to the V8 workspace databases when you chose the workspace.

When you set up the V6 workspace, make sure that you have servers for each of the workspace databases. Use the V7 Progress/RTB client when selecting the workspace. The V7 client will be able to connect to the V6 databases. Do all of your development using the V7 Progress/RTB client. You will do your testing using a V6 Progress client (click on the icon or run the script to start the app outside of RTB). Since shops mixing three environments typically make very few changes to the oldest version, so this system works out nicely.

Issues:
The r-code generated for the workspace is V7 r-code. You will either need to turn of r-code for the workspace so that you hit source when you test, or set up the external compiler using a V6 Client to do the compile. WARNING: Roundtable has not been tested using the external compiler in this manner, but it should work ok. When RTB populates the external compile (V6) database with a record to compile, the V6 session will read the record and perform the compile.