Upgrade Advisor for upgrade to Synergy 7.2

David Honey

2011-03-02

1 Introduction

In releases prior to Rational Synergy 7.2, there was support for two legacy methodologies:

- Projects that updated by object status. This was sometimes called object status-based reconfigure.
- Projects that updated by tasks (TBCM, task based CM) but with manual update properties rather than using process rules.

The intention to retire and drop support for both of these legacy methodologies was announced in 2007 in the Synergy 6.5 release notes. Prior to 7.2, support was provided through the legacy classic client, but not in the newer clients such as the Synergy GUI and Synergy CLI.

Since the classic client has been retired in Synergy 7.2, both methodologies are now retired. In 7.2, all projects will be updated using tasks and process rules. If all the projects in your Synergy databases are configured to update by process rule, the retirement of these legacy features should have no effect. If you have projects that update by object status, it is likely that after upgrade, unless you take steps to create a suitable baseline and use tasks, *update members* will deliver different results. Similarly, if you have projects that use task-based manual update properties, *update members* after upgrade could deliver different results. The Synergy 7.2 release notes describes this and provides an overview of the steps that you should take before upgrading to prepare for and minimize the effects of upgrading to 7.2. The benefit of reviewing and making these changes before upgrading is that this can be done piecemeal over a period of time, and tested as you go. This allows for an orderly transition and helps to mitigate any risks in upgrading.

This document describes an upgrade advisor that can be executed in a Synergy 6.5SP2, 6.6a, 7.0, 7.1, or 7.1a database. Note that direct upgrade to 7.2 is only supported from 7.0, 7.1 or 7.1a. However, if you are using an earlier release, you may still want to use the upgrade advisor described in this document to plan and prepare for upgrade to 7.2 via an intermediate release such as 7.1.

The upgrade advisor analyses the data in a database and provides specific advice where possible about how to prepare for retirement of object status based update and manual update properties in 7.2 and produces a report. The specific details of the analysis performed are described in Analysis.

The upgrade advisor does *not* make the changes to prepare for the upgrade. It produces a report that describes each issue that it finds and reports it using one of 3 severity levels:

- `Info` means an issue that does not require user action but which may provide useful information. For example, old non-static projects that are candidates for clean-up are reported with this severity level.
- `Warning` means an issue that will require corrective action for which the upgrade advisor can show the specific commands needed to perform such preparation.
- `WARNING` means an issue that the upgrade advisor has detected but where there is no clear way of resolving the issue in a manner that delivers the same update behavior. For example, if projects for the same release and purpose have inconsistent update properties, this cannot be directly supported by process rules since they mandate consistent update properties.

The upgrade advisor therefore provides a tool to detect what issues might be apparent on upgrade to 7.2, and where possible shows the commands that can be executed. You can copy the relevant parts of the report into scripts to partially automate the transition to task based update based on process rules.

2 Using the upgrade advisor

2.1 Introduction

The upgrade advisor is shipped as a separate ACcent library named `ua72.a`. The library is platform independent and is compatible with Synergy 6.5sp2, 6.5a, 6.6a, 7.0, 7.1, 7.1.0.*, and 7.1a on all supported platforms.

2.2 Loading the ACcent library

To use the tool, you need to first start a Synergy Classic CLI session on the database to be analyzed or repaired. Then load the ACcent library:

```
ccm load –a path/ua72.a
```

This will have to be done each time a new session is started.

2.3 Defining the ua72 command

In order to use the tool, a new command needs to be defined. In the example below, the command verb `ua72` is used, but you could choose another command verb if you prefer.

```
ccm define ua72 ua72cmd upgrade_advisor_cmd
```

This will have to be done each time a new session is started.

2.4 Command syntax

Assuming that the verb `ua72` is used for the definition, the command syntax is:

```
ccm ua72 [-cs|-comment_string comment_string]
       [-inactive_days inactive_days]
       [-report report_file]
       [project_spec...]
```

`-cs|-comment_string comment_string`
Specifies a comment string that will precede each line in the report that is not an example of a command. If not specified, the default is "#", the comment character used on Unix shell scripts. On Windows, if you want to copy comments and commands into a Windows batch file, you might use a comment string of "`REM`".

`-inactive_days inactive_days`
Specifies the number of days that a project has not been modified for that will result in the project being shown as a candidate for clean-up. If not specified, the default is 365 days. That is, non-static projects that have not been modified for at least 365 days will be reported as candidates for clean-up.

`project_spec...`
Specifies one or more projects to be analyzed. Any projects that are in a static state or in the `checkpoint` state are ignored. If not specified, the upgrade advisor queries for all projects that are in a non-static state and not in the `checkpoint` state. It is advisable that you do not query by purpose or member_status. The upgrade advisor needs to consider all non-static and non-checkpoint projects for a release in order to correctly detect that the release has all the required release-specific process rules. If you want to break down the analysis into several pieces, querying projects by release would be a good way to achieve that.

-report *report_file*
Specifies the path to the report file that is generated. Relative paths may be used. The parent directory must be writable by the user. If not specified, by default the report is named using the simple database name (the last part of the database path) plus an underscore and the current time formatted as YYYY_MM_DD__HH_MM_SS. If the specified file already exists, an error is reported before analysis starts and the command returns a non-zero error status code.

2.5 Strategy

Since the upgrade advisor allows you to prepare for the upgrade to 7.2 at your own pace, there are several strategies that you might consider:

1. Take a copy of each production database and experiment in that copy. This has the advantage that testing of any changes can be done without any direct impact on your day-to-day production environment. This is the safest approach, but has the disadvantage that you will have to note all the changes required that passed testing, and then reapply those changes separately to your production database.
2. In your production database, copy new versions of projects from the projects that update by object status or use manual update properties, and test changes on those new project versions. There is increased risk in this approach. If you have a release for which some projects use process rules and others use manual update properties, changes made to process rules could affect existing projects. The advantages of this approach are that you do not need to find the resources to make a copy of a production database, and some of the changes can be made directly in that database. It also allows you to compare the resultant members between the version of project previously used, and the new version of project using a process rule.
3. In your production database, apply the changes for each release, purpose and project that need them. This has the advantage that you only make changes once and do not have to reapply them. However, its chief disadvantage is that there is an increased risk that during testing and resolution of issues, existing projects might not update as expected or desired. You might mitigate the risks of these by limiting this to releases and projects that can suffer some disturbance.

It is important that you test the changes you make to projects to use process rules and to the process rules by those projects. One way of testing this is to compare the project members as delivered by update members using a process rule with the members as delivered with your current set-up. This is most easily accomplished if those two project versions are in the same database since you can then directly compare the members of the two projects. Alternatively, you could perform an *update members* on a project examine the changes as output in the operation, and then if incorrect perform an *undo update members* on that project. However, remember that Synergy only supports one level of "undo" for update members. If you performed *update members* after incorrect results, you could not undo that and the previous one. Hence this technique requires greater care if you want to minimize disruption to regular development.

If you using DCM, you should co-ordinate any changes made with the corresponding users and build managers of the other databases in the DCM cluster. You should consider whether for any specific release, you want it and its release-specific process rules to be managed and controlled in a single database, and for other databases to use copies from that master controlling database. See *Replication of generic and release-specific processes* in the *DCM User Guide* for more details. You should ensure that any new purposes you define are defined in each database in the cluster – this is a prerequisite for DCM as described in *Establishing common database* parameters in the *DCM User* Guide. In order to plan this for a DCM cluster, it would be beneficial to run the upgrade advisor on each database in the cluster that is at a Synergy release prior to 7.2. Collating and viewing all the reports from the database in the cluster will help you plan for what needs to be done in each database for each release and set of projects.

3 Analysis

3.1 Introduction

The first stage of analysis is to categorize the projects as follows:

a. Projects that are controlled in another database in the DCM cluster grouped by the DCM database identifier of the controlling database.
b. Projects in a static state or the `checkpoint` state
c. All other projects grouped by release-purpose pair.

Further analysis and reporting is performed in the following steps:

1. Projects that are controlled in another database in the DCM cluster are reported. See Non-local projects.
2. Projects in a static state or the `checkpoint` state are reported. See Projects in a static state or the checkpoint state.
3. Any project not in #1 or #2 that has not been modified in the last N days or uses an inactive release is reported as a candidate for clean-up. See Candidates for clean-up.
4. Projects with a missing release or purpose are reported. See Projects with a missing release or purpose.
5. Missing purpose definitions used by the projects in group #c are reported. See Missing purpose definitions.

If there are any warnings reported in steps 4 or 5, the upgrade advisor stops further analysis. You must correct these issues before you can re-run the upgrade advisor and get a report on subsequent issues.

6. Missing release definitions used by the projects in group #c are reported. You must correct these issues before you can re-run the upgrade advisor and get a report on subsequent issues. See Missing release definitions.
7. Missing purposes for releases used by the projects in group #c are reported. You must correct these issues before you can re-run the upgrade advisor and get a report on subsequent issues. See Missing purposes for release definitions.
8. Projects that update by object status are analyzed and reported. See Projects that update by object status.
9. Projects that update using task-based manual update properties are analyzed and reported. See Projects that use task-based manual update properties.

3.2 Non-local projects

Projects that are controlled in another database in the DCM cluster cannot be updated in the current database. So any upgrade of the current database will not have any direct effect on them for update members. The upgrade advisor reports on each such controlling database and the projects in the current database that are controlled in that controlling database. These projects are excluded from further analysis. If you are upgrading a DCM cluster and any of those other databases are not at Synergy 7.2 or later, you should run the upgrade advisor in each of those other databases. See Strategy for details.

3.3 Projects in a static state or the checkpoint state

Projects in a static state or the `checkpoint` state cannot be updated. So any upgrade of the current database will not have any direct effect on them for update members. The upgrade advisor reports on such projects and they are excluded from further analysis.

3.4 Candidates for clean-up

Removing old and unwanted non-static projects before upgrading is beneficial for two reasons:

- It reduces the number of projects that the upgrade advisor has to analyze in detail.

- It reduces the work that `ccmdb upgrade` will have to perform on upgrading to 7.2.

The upgrade advisor considers a project as a candidate for clean-up if it satisfies either of the following conditions:

- It uses a release value that corresponds to an inactive release.
- It has not been modified in the last N days (by default 365 days).

Further analysis will be performed on these projects. However, you may wish to remove some or all of these old projects and rerun the upgrade advisor.

3.5 Projects with a missing release or purpose

In Synergy 7.2, a project must have a valid defined release and purpose before *update members* can be executed on it. The upgrade advisor reports the projects that are in a non-static state and not in the `checkpoint` state that are missing a `release` or `member_status` attribute or where those attributes have en empty string value. The most likely scenario for this are projects using object status-based update methodology. That legacy methodology did not mandate that projects used release values. However, it was considered best-practice to use release values since this was usually required to appropriately manage development on parallel releases or variant releases.

The report shows an example of the commands but cannot show the specific commands to fix each issue since it does not what release and/or purpose should be used. You will have to fix such errors manually using release and purposes that you feel are appropriate for that project.

3.6 Missing purpose definitions

Synergy has always required that a project has a valid purpose before *update members* can be executed on it. The upgrade advisor reports the projects that have a `member_status` attribute value that does not correspond to a valid purpose as defined in the *project purpose table*. The report shows example commands that would create each missing purpose. However, since it does not know the meaning of that purpose, it cannot give it a meaningful descriptive *purpose name*.

If your database is part of a DCM cluster, you should ensure that an equivalent set of purposes is defined in each database in the cluster. You should compare the project purpose table from each database in the cluster so that you can use a consistent `member_status` value and *purpose name* for any new purposes you define. See Strategy for details.

3.7 Missing release definitions

Synergy 6.5 and later has always required that the release value used for a project must be defined by a release definition before *update members* can be executed on it. This is especially the case when process rules are to be used since the process rules for a release are related to the release definition that controls them.

The upgrade advisor reports each such missing release definition. It determines the set of purposes used by non-static projects of that release and shows the command that will create such a release definition. The suggested baseline release for the new release definition is determined as follows:

1. The baseline projects of the non-static and non-checkpoint projects are examined. If any have a baseline project with a release value that is not the same as the release, that release value is used as the proposed baseline release. If no baseline release is found by the immediate baseline projects, the baseline project of those baseline projects are examined, and so on.
2. If no baseline release can be found from a baseline project, the predecessors of the non-static and non-checkpoint projects are examined. If any of those have a release that is not the same as the release to be created, that release value is used as the proposed baseline release. If no

predecessor provides such a baseline release, the predecessors of those predecessors are examined, and so on.

3. If no baseline release can be found from either #1 or #2, and the release has a component name, then the most recently created release using that component name is used for the proposed baseline release.

The proposed purposes for the new release are determined by including all the purposes of non-static projects for that release being examined.

3.8 Missing purposes for release definitions

If a release definition exists, the upgrade advisor looks at the purposes of all the non-static and non-checkpoint projects that were categorized. The suggested changes to the release use those purposes. If the release definition is modified using the suggested command, the release specific process rules used will be based on the `standard` process. If you wish to use another process rule for that purpose, you will have to manually modify the release definition to use the desired generic process rule.

3.9 Projects that update by object status

3.9.1 Introduction

When task-based CM (TBCM) was introduced in Continuus/CM 4.5 over 10 years ago, it was recommended that tasks were used even if customers remained on the previous object status-based methodology. This is because tasks group together related sets of changed files into a single logical change. However, the use of tasks was optional in this case. It is possible that projects using object status based update do not use tasks consistently or perhaps do not use any tasks. In order for update on such projects to work as expected with task based process rules, you must establish:

- Either a baseline, or a set of baseline projects, and…
- One or more tasks representing the changes in projects that are not in the baseline or baseline projects.

The upgrade advisor assumes that projects that update by object status will be converted to use task-based process rules that use *baseline methodology* or that you have set up the process rules to use *latest projects.* It reports on three aspects:

1. The commands to modify the project to task-based using process rules.
2. The tasks required to associate changes.
3. The baseline or baseline projects required.

3.9.2 Determining tasks to associate changes

To determine the tasks required to associate changes, the upgrade advisor iterates over each release for which there was at least one project that updated by object status. It determines which of the following three cases apply:

1. There is at least one process rule for the release using baselines, but there is no baseline for that release. It queries for all source members of the projects for that release that are not associated with a completed task for that release.
2. There is at least one process rule for the release using baselines, and there is a latest baseline for that release. It queries for all source members of the projects for that release that are not associated with a completed task for that release and are not members of the projects in the baseline.

3. There are no process rules for the release using baselines. It queries for all source members of the projects for that release that are not associated with a completed task for that release and that are not members of the latest baseline project for each project.

The source members are divided into static changes, and non-static changes by owner. The upgrade advisor shows the commands required to create tasks for these changes, associate them with the required objects, and complete those tasks.

3.9.3 Analysis of baseline or baseline projects

3.9.3.1 Introduction

The upgrade advisor performs this analysis for projects with either object status base update, or task-based manual update properties. The purpose of this analysis is to:

- See whether the current baseline projects used by the projects being analyzed are compatible with and would be candidates of the corresponding process rule.
- Determine the baseline project compatibility if the process rule was changed from using baselines to latest baseline projects, or vice-versa.

The analysis performed depends on the selection mode of the process rule:

- Latest baseline
- Baseline specified on process rule
- Baseline specified on project grouping
- Latest baseline projects

3.9.3.2 Latest baseline

The upgrade advisor determines the latest baseline that matches the process rule and checks that:

- There is an eligible latest baseline.
- That all of the baseline projects of the projects being analyzed are members of that baseline.

If either condition is not met, it shows an analysis of whether the baseline projects would be candidates for Latest baseline projects. Based on this, the user can decide between any of the following actions:

- Creating a new baseline that contains appropriate baseline projects.
- Changing the process rule to use a selection mode of latest baseline projects.

3.9.3.3 Baseline specified on process rule

The upgrade advisor checks that:

- The process rule is associated with a specified baseline.
- If such a baseline exists, that all of the baseline projects of the projects being analyzed are members of that baseline.

If either condition is not met, it shows an analysis of whether the baseline projects would be candidates for either Latest baseline or Latest baseline projects. Based on this, the user can decide between any of the following actions:

- Creating a new baseline that contains appropriate baseline projects.
- Changing the process rule to use a selection mode of latest baseline.
- Changing the process rule to use a selection mode of latest baseline projects.

3.9.3.4 Baseline specified on project grouping

The upgrade advisor checks that:

- the project has a project grouping, and that project grouping is associated with a specified baseline.

- and if so, checks that all of the baseline projects of the projects being analyzed are members of that baseline.

If either condition is not met, it shows an analysis of whether the baseline projects would be candidates for either Latest baseline or Latest baseline projects. Based on this, the user can decide between any of the following actions:

- Creating a new baseline that contains appropriate baseline projects.
- Changing the process rule to use a selection mode of latest baseline.
- Changing the process rule to use a selection mode of latest baseline projects.

3.9.3.5 Latest baseline projects

The upgrade advisor determines the latest baseline project that matches the process rule for each project, and checks that:

- There is an eligible latest baseline project.
- That the latest baseline project is the baseline project.

If either condition is not met, it shows an analysis of whether the baseline projects would be candidates for Latest baseline. Based on this, the user can decide between any of the following actions:

- Changing the update properties to use the latest baseline project.
- Changing the process rule to use a selection mode of latest baseline.

3.10 Projects that use task-based manual update properties

3.10.1 Introduction

The upgrade advisor compares the manual update properties of the non-static manual update projects for the same release-purpose pair. It determines which one of the following cases applies:

1. The project's update properties are inconsistent. You cannot have inconsistent update properties between projects for the same release and purpose when using a process rule. Since 7.2 requires the use of a process rule, on upgrade, all of these projects will match the process rule. It is likely that for some or all projects, different candidates will be considered by *update members*. The upgrade advisor cannot determine what needs to be done in this case. You will have to determine an appropriate change of update properties to achieve the desired result with a process rule.
2. The project's update properties are consistent and equivalent to the corresponding process rule. Such projects will consider the same candidate changes as the process rule. These projects can be changed to use the process rule. See Equivalence of update properties for details of how equivalence is determined. The upgrade advisor goes on to consider baselines and baseline projects. See Analysis of baseline or baseline projects.
3. The project's update properties are consistent but not equivalent to the corresponding process rule. If those projects are changed to use the current process rule, it is likely that different candidates will be considered by *update members*. See Equivalence of update properties for details of how equivalence is determined. You have the choice between two approaches in such cases:
    a. Use the default methodology of using latest baselines. The upgrade advisor performs an analysis of baselines and baseline projects. See Analysis of baseline or baseline projects. You may have to create a suitable baseline to achieve the same update behavior. The advantage of this choice is that you will be using the standard out-of-the-box methodology. The use of baselines has a number of advantages including the ability to see which changes are partially or fully included in a baseline, and the use

of *component tasks* associated with the projects and products in baselines. These can be particular beneficial in *component-based development*.

b. Change the process rule to use latest baseline projects. The upgrade advisor shows the commands needed to make the process rule equivalent to the tasks and folders in the manual update properties of the projects. See Determining process rule changes to match manual update properties. The upgrade advisor performs an analysis of baselines and baseline projects. See Analysis of baseline or baseline projects. The advantage of this choice is that this will preserve the current update behavior. However, you will also not be able to take advantage of the benefits of using baselines or component tasks.

3.10.2 Equivalence of update properties

When comparing the manual update properties of projects, the upgrade advisor examine the tasks and folders in each project. Two projects are considered equivalent if they satisfy all of the following conditions:

- They have the same set of tasks directly in their update properties.
- They have the same set of manual folders.
- They have the same set of query-based folders that are writable by "none".
- They have the same set of normalized query-writableBy pairs. For example, if project A uses a query-based folder with query Q and writableBy W, and project B uses a different a query-based folder with the same query Q and the same writableBy W, both have the same set of query-writableBy pairs.

Query expressions are normalized by replacing:

- query clauses for `release` values that match the project to use a "`%release`" keyword.
- query clauses for `resolver` or `owner` values that match the project owner to use a "`%owner`" keyword.
- query clauses for `completed_in` or `created_in` values that match the local DCM database identifier to use a "`%database`" keyword

Note that this equivalence comparison does not test the baselines or baseline projects. The upgrade advisor performs a separate analysis of this – see Analysis of baseline or baseline projects.

3.10.3 Determining process rule changes to match manual update properties

The upgrade advisor works out the changes needed to make a process rule equivalent to the manual update properties on a project. This is determined as follows:

- Non automatic tasks that are directly in the update properties have to be represented as a manual folder containing those tasks, and then that folder needs to be added to the release-specific process rule. This is because process rules cannot use tasks directly.
- A folder in the update properties that is controlled by a folder template and which matches the project's release keeps that folder template in the process rule if it used. Otherwise, the folder needs to be added to the process rule.
- A manual folder, or a query-based folder that is writable by none, must be a folder in the process rule for it to be equivalent.
- A query-based folder that is not writable by none, must be represented by either a folder template in the process rule whose query expression and writeable-by match. If not, a query based folder with the query expression and writeable-by all is created and added.

- Any surplus folder templates or folders in the process rule not required from above are to be removed.