Rational. Synergy

IBM

ビルド マネージャ ガイド

IBM Rational Synergy

ビルド マネージャ ガイド

リリース 7.1a

本書をご使用になる前に、「付録 B：特記事項」に記載されている情報をお読みください。

本書は、Rational Synergy（製品番号 5724V66）バージョン 7.1a および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

目次

1 概要

Rational Synergy ビルド マネージャ ガイドは、ビルド マネージャを対象読者としています。ビルド マネージャとは、ソフトウェア製品がどのように組み合わされ、ビルドされているかを理解している人のことです。このドキュメントでは、ビルド管理の準備、実行、処理、トラブルシューティングの方法を説明しています。

このドキュメントの読者は IBM® Rational® Synergy に精通しているものとします。使用する用語や概念、取り上げる方法論、提示するシナリオは、読者が概念（タスクベース CM 方法論）の上でも実務（開発者レベルのタスクベース CM 操作方法）の上でも Rational Synergy を理解していることを前提としています。また、このドキュメントでは、タスクベース CM 方法論を使用してビルド管理タスクを実行する方法についても説明しています。

このドキュメントに記述された手順は、すべて Rational Synergy GUI を使用して操作したものです。

Rational Synergy を初めて使用する読者は、このドキュメントを読む前に以下の Rational ドキュメントを十分に理解してください。

- 用語、概念、方法論については、『Rational Synergy の紹介』を参照してください。
- 用意されたチュートリアル データベースを使用して短いチュートリアルを行う場合は、『Rational Synergy チュートリアル』を参照してください。
- 操作手順の説明については、Rational Synergy ヘルプを参照してください。
- コマンドとデフォルトの設定の説明については、Rational Synergy Classic CLI ヘルプを参照してください。

  このドキュメントでは、コマンドライン インターフェイスを使用する手順については、Rational Synergy Classic CLI を使用した手順を記載しています。この CLI 用のヘルプは、Rational Synergy Classic CLI ヘルプといいます。

  Rational Synergy の CLI もあり、ヘルプは Rational Synergy CLI ヘルプ、ウェブ モード といいます。ただし、このドキュメントではウェブ モードのヘルプは参照して**いません**。

Rational Synergy には精通しているが、**ビルド管理は初めて**という読者は、このドキュメントを参照してください。

ビルド管理に関する情報の参照

ビルド管理に関する情報は、すべてこのドキュメントに記載されています。このドキュメントはランダムな参照がしやすい HTML 形式ですが、最初にビルド管理の準備とビルド管理の基本を順に読んでから他の章やトピックを参照してください。

このドキュメントには、最も基本的な情報（ビルド管理の準備）からより複雑な内容（さまざまなビルド管理）へ、情報が体系的に順序立てて掲載されています。初めて読む場合は、目次の順序どおりに読み進めてください。ドキュメントを理解したら、よく使用するトピックを「お気に入り」フォルダに保存するとよいでしょう。これで、知りたい情報をすばやく見つけることができます。

また、IBM® Rational® Change を使用している場合は、定義済みのクエリを実行して Rational Synergy ビルドの内容についての情報を収集できます。Rational Change の定義済みビルド管理クエリの詳細については、Rational Change ユーザー ヘルプの「CM ビルド情報のクエリ」を参照してください。

ビルド管理のロード マップ

以下に、チームのビルド管理を準備、実施するために必要な各操作について簡単に説明します。これらのタスクは、新たなプロジェクトを開始するたびに実行します。

準備

- 環境設定
- 統合テスト プロジェクトの作成
- システム テスト プロジェクトの作成

継続的な統合テスト サイクル

- プロジェクトの更新
- コンフリクトの検出
- ビルドとテスト
- 開発者にテスト ベースラインを公開

システム テスト サイクル

- 特定のタスクを使用するビルド
- プロジェクトの更新
- コンフリクトの検出
- ビルドとテスト

ソフトウェアのリリース

- ソフトウェアのリリース
- 新しいリリースの準備

特殊な考慮点

後半の章では特殊なトピックについて説明しています。これらのトピックは基本の範囲外のものです。サイトで必要にならない場合もありますが、説明を読み、特殊な状況が発生した場合の対処方法を把握しておいてください。

表記規則

このドキュメントで使用されている表記規則について説明します。

書体	説明	例
太字	— ダイアログボックス名とオプション — メニューのコマンドとボタン — 強調 — アイコン名 — レジストリ キー (大文字) — ツールバーのボタン名	— **ログイン** ダイアログボックス、**パスワード** フィールド — **タスク** ボックス、**適用** ボタン — パスを変更**しないで**ください。 — **クエリ** アイコン — **HKEY_LOCAL_MACHINE** — **カレント タスク** ボタン
イタリック	— プレースホルダ — ロールと状態 — 入力する文字列	— *picture.gif* ファイルを追加します。 — *build_mgr*、*shared* — *Yes* と入力します。
`Courier`	— コマンドとオプション — コード サンプル — ファイル名 — パス	— `ccm start -h lego` — `void main ()` — `foo.c` ファイル — `/user/local/ccm_docs`
大文字の英字	— 環境変数とマクロ — レジストリ キー（太字）	— CCM_ADDR — **HKEY_LOCAL_MACHINE**

注記：Rational Synergy ソフトウェアに関する重要な情報です。

注意！注意しないとデータベース、データベース サーバー、Rational Synergy ソフトウェアの統合システムの一部、またはシステムに被害を及ぼす情報を提供します。

コマンドライン インターフェイス

Rational Synergy は、サポートされているすべてのプラットフォームで CLI をサポートします。すべての Rational Synergy コマンドを、UNIX® のシェルまたは Windows® のコマンド プロンプトから実行できます。

オプション区切り文字

デフォルトでは、オプション区切り文字として、Windows クライアントはスラッシュ（/）をサポートし、UNIX クライアントはダッシュ（-）をサポートします。本書の例では、両方の区切り文字を使用します。

標準

テキスト ファイルの編集方法の説明にはメモ帳（Windows）、または vi（UNIX）コマンドを使用しています。メモ帳および vi は Rational Synergy のデフォルトのテキスト エディタです。別のテキスト エディタを使用する場合は、適切なコマンドを代わりに使用してください。

Rational Synergy の変更点

リリース 7.1a では、Rational Synergy のグラフィカル インターフェイスおよびコマンドライン インターフェイスに、ビルド管理操作が組み込まれました。その他のグラフィカル インターフェイスは、本リリースでは Rational Synergy Classic と呼ばれています。

このドキュメントでは、CLI を使用する手順については、旧リリースの Rational Synergy Classic CLI のコマンドとデフォルト設定を使用した手順を記載しています。新しい Rational Synergy CLI（ウェブ モード）を使用する場合は、必ず使用するインターフェイスのデフォルト設定を確認してください。デフォルト設定については、Rational Synergy CLI ヘルプ、ウェブ モードで説明されています。

ウェブ モードとトラディショナル モード

Rational Synergy 7.1a では、Rational Synergy クライアントが HTTP プロトコルでウェブベースの Rational Synergy サーバーと通信する、新しいアーキテクチャを導入し、ワイドエリア ネットワーク（WAN）の機能を拡張しました。このアーキテクチャは、クライアントとサーバー間でパラレル非同期ネットワーク通信を使用することで、ネットワーク待機時間への依存を軽減します。

Rational Synergy 7.1a では、従来のネットワーク通信を置き換えるのではなく、ウェブ モードとしてこの新しいアーキテクチャを導入しています。従来の RFC アーキテクチャはトラディショナル モードとして継続しており、これを利用することも可能です。

ほとんどの開発者とビルド マネージャはウェブ モードを使用できます。ただし、管理機能や高度な機能を利用するユーザーは、トラディショナル モードを使用する必要があります。トラディショナル モードは、Synergy 6.5a と同じように動作します。

以下の操作を行う場合は、Classic Client の CLI または GUI を使用する必要があります。

- 大部分の管理操作
- データの移行

このドキュメントでは、CLI から実行するコマンドについて、Rational Synergy Classic CLI を使用した操作を記載しています。

Rational Synergy の変更点の詳細については、*Readme* を参照してください。

Rational 製品のドキュメント

Rational 製品のドキュメントは、ドキュメント DVD では HTML と PDF 形式で、IBM® Rational® Information Center（`http://publib.boulder.ibm.com/infocenter/rsdp/v1r0m0/index.jsp`）では PDF 形式で提供されています。DVD を共有ドライブにマウントすることで、すべてのユーザーがドキュメントを利用できるようになります。

Readme の内容は、ドキュメントおよびすべての Rational Synergy ヘルプ システムの内容に優先します。Rational Synergy ユーザーは、IBM Rational Information Center から最新版の *Readme* を入手できます。

Rational Synergy ヘルプ システム

ここでは、使用するさまざまなインターフェイスで利用可能なヘルプ システムについて説明します。ビルド マネージャ ガイドでは、いくつかのインターフェイスとヘルプ システムについても記述しています。このドキュメントでは、CLI を使用する手順については、Rational Synergy Classic CLI を使用した手順を記載しています。詳細については、Rational Synergy の変更点 を参照してください。

Rational Synergy リリース 7.1a 用の Synergy Classic コマンドライン インターフェイス（CLI）ヘルプには、サポートされるコマンドがすべて含まれています。ただし、いくつかのデータベース管理コマンドについては、『管理者ガイド』にのみ記載されています。

Rational Synergy Classic グラフィカル ユーザー インターフェイス（GUI）には、GUI から開くヘルプ システムと同様の情報が含まれます。Rational Synergy Classic のヘルプ システムには CLI のヘルプもありますが、最新のコマンド ライン情報ではなく、新規コマンドとオプションが含まれていません。このリリースで使用可能な最新のコマンドとオプションを確認するには、コマンドライン インターフェイスの使用中にヘルプを起動してください。

Rational Synergy Classic インターフェイスの最新のヘルプは、リリース 6.3 用のものです。Rational Synergy Classic GUI は、リリース 6.3 以降大きな変更はありません。具体的な変更点については、*Readme* を確認してください。

どのヘルプが表示されているのかが分からない場合は、各ヘルプ ページのフッターを見ればリリースが確認できます。また、Rational Synergy Classic（CLI、GUI とも）のヘルプの背景は青、Rational Synergy のヘルプの背景は白です。

Readme

Rational Synergy の *Readme* には Rational Synergy の新しい機能の説明、ドキュメントのアップデート、トラブルシューティング、IBM Rational ソフトウェア サポートのお問い合わせ方法、および既知のエラーに関する情報があります。インストール ガイドの最新のアップデートは、*Readme* を参照してください。

Readme は、製品 DVD および Rational Synergy Information Center に HTML 形式で用意されています。

Readme の内容は、マニュアルおよびヘルプの内容に優先します。

IBM Rational ソフトウェア サポートへの問い合わせ

お手持ちのリソースで、問題が解決されない場合は、IBM®Rational® ソフトウェア・サポートに連絡してください。IBM® Rational® ソフトウェア・サポートでは、製品の問題解決に関する支援を行っています。

前提条件

IBM Rational ソフトウェア・サポートに問題を送信するには、有効な Passport Advantage® ソフトウェア保守契約が必要です。パスポート・アドバンテージは、IBM の包括的ソフトウェア・ライセンスおよびソフトウェア保守 (製品のアップグレードおよび技術支援) オファリングです。次のサイトからオンラインでパスポート・アドバンテージに登録できます。 http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.htm

- パスポート・アドバンテージについて詳しくは、パスポート・アドバンテージ FAQ (http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html) にアクセスしてください。
- さらに支援が必要な場合は、IBM 担当員に連絡してください。

問題をオンラインで (IBM Web サイトから) IBM Rational ソフトウェア・サポートに送信するには、さらに以下が必要です。

- IBM Support Web サイトの登録ユーザーであること。登録について詳しくは、http://www-01.ibm.com/software/support/ を参照してください。
- 許可された呼び出し元としてサービス要求ツールにリストされていること。

問題報告について

次のようにして、IBM Rational ソフトウェア・サポートに問題を送信します。

1. お客さまの問題のビジネス・インパクトを判別します。IBM へ問題を報告する際は、重大度レベルを問われます。そのため、報告する問題とそのビジネス・インパクトを理解して、評価する必要があります。

重大度のレベルを決めるにあたっては、下表を参照してください。

重大度	説明
1	問題は**危機的な**ビジネス・インパクトを持ちます。プログラムを使用できず、業務に重大な影響が出ています。この状況には、即時に解決策が必要とされます。
2	問題は、**重大な**ビジネス・インパクトを持ちます。プログラムは使用可能ですが、非常に限定されています。
3	問題は**部分的な**ビジネス・インパクトを持ちます。プログラムは使用可能ですが、比較的重要でない（業務に大きな影響はない）機能が利用できません。
4	問題は**わずかな**ビジネス・インパクトを持ちます。問題による業務への影響がほとんどないか、問題に対する有効な回避策が実施済みです。

2. 問題を説明して、背景情報を収集します。IBM に問題を説明する際は、なるべく具体的に説明してください。IBM Rational ソフトウェア・サポートの専門家が、問題を解決するために効果的な支援をできるように、関連するすべての背景情報を含めてください。時間を節約するために、以下の質問の答えを用意してください。

- 問題の発生時に実行していたソフトウェア（複数可）のバージョンは何ですか？

  次のオプションを使用して、正確な製品名とバージョンを判別することができます。

  - IBM Installation Manager を始動して、「ファイル」>「インストール済みパッケージの表示」を選択します。パッケージ・グループを展開し、パッケージを選択して、パッケージ名およびバージョン番号を確認します。
  - 製品を始動して、「ヘルプ」>「製品情報」をクリックし、オファリング名とバージョン番号を確認します。
- オペレーティング・システムおよびバージョン番号 (サービス・パックまたはパッチを含む) は何ですか？
- 問題の症状に関連するログ、トレース、およびメッセージはありますか？
- 問題を再現できますか？再現できる場合は、問題を再現するための手順は何ですか？
- システムに変更を加えましたか ？ 例えば、ハードウェア、オペレーティング・システム、ネットワーキング・ソフトウェア、またはその他のシステム・コンポーネントに変更を加えましたか？

- 現在、問題に対する何らかの回避策を使用していますか？使用している場合は、問題の報告時にその回避策も説明する準備をお願いします。

3. IBM Rational ソフトウェア・サポートに問題を送信します。次の方法で、IBM ソフトウェア・サポートに問題の送信ができます。
   - **オンラインの場合：** IBM Rational ソフトウェア・サポートの Web サイト (https://www.ibm.com/software/rational/support/) にアクセスして、Rational サポート・タスク・ナビゲーターで「サービス要求を開く **(Open Service Request)**」をクリックします。エレクトロニック問題報告ツールを選択し、「問題管理レコード (PMR) (Problem Management Record (PMR))」を開き、問題についてご自身の言葉で正確に記述してください。

     サービス要求を開く方法について詳しくは、http://www.ibm.com/software/support/help.html にアクセスしてください。

     IBM Support Assistant を使用してオンラインのサービス要求を開くこともできます。詳しくは、http://www-01.ibm.com/software/support/isa/faq.html を参照してください。
   - **電話の場合**：国または地域別の電話番号を調べるには、http://www.ibm.com/planetwide/ の「IBM directory of worldwide contacts」で、お住まいの国名または地域名をクリックします。
   - **IBM 担当員に依頼する場合**：オンラインまたは電話で IBM Rational ソフトウェア・サポートにアクセスできない場合は、IBM 担当員に連絡してください。必要な場合は、お客さまに代わって、IBM 担当員がサービス要求を開くことができます。http://www.ibm.com/planetwide/ で、各国への詳しい連絡先情報を検索できます。

送信した問題が、ソフトウェアの障害に関するものか、資料の欠落や不正確な記述によるものである場合は、IBM ソフトウェア・サポートはプログラム診断依頼書 (APAR) を作成します。APAR には、問題の詳細が記述されます。IBM ソフトウェア・サポートは可能な限り、APAR が解決されてフィックスが提供されるまでの間に実施できる回避策を提供します。IBM は、同一の問題を経験している他のユーザーが同じ解決方法を利用できるように、ソフトウェア・サポート Web サイトに解決済みの APAR を公開し、毎日更新しています。

2 ビルド管理の準備

ビルド管理とは、企業のソフトウェア製品をビルドし、管理するプロセスです。

企業が Rational Synergy を使用してコードを管理する場合、ソフトウェア製品のビルドと管理はビルド マネージャが担当します。

ビルド マネージャとは、以下のプロセスを管理する責任を負う人です。

- ソフトウェアの初期バージョンからベースラインを作成する
- ソフトウェアの構造の体系化および精緻化
- テストおよびステージング用ビルド管理プロジェクトの設定
- プロセス ルールやフォルダ テンプレートの設定とメンテナンス
- 開発者からのソフトウェア変更の収集とテスト エリアの構築
- Rational Change のレポートを生成し、ビルドに含まれる／含まれない機能とタスクを調べる
- 顧客リリースなどの重要なマイルストーンでのソフトウェア凍結
- チームで使用する構成情報（プラットフォーム値やリリース値など）の設定
- 開発者へ最新の構成を提供
- 不要となったベースラインの削除
- 問題の特定や修正タスク作成のための旧ソフトウェア リリースの再作成

この一覧は、ビルド管理における責任項目を簡単にまとめたものです。各項目にはそれぞれ特定の作業が含まれます。このドキュメントではそのすべてについて詳しく説明します。

作業を開始する前に

次のセクションに進む前に、以下の準備操作を実行します。

UNIX のビルド マネージャ

- *ccm_root* グループのメンバーになる。

  すべてのビルド マネージャとユーザー *ccm_root* は、UNIX ネットワークの *ccm_root* グループのメンバーである必要があります。システム アドミニストレータに確認してください。

- *build_workarea* ディレクトリのグループを *ccm_root* に設定する。*build_workarea* は、すべてのビルド管理ワークエリアを格納するディレクトリです。*build_workarea* ディレクトリの権限は、グループが読み書きできるように設定します。

  ```
  su ccm_root
  cd /shared_directory
  mkdir build_workarea
  chgrp ccm_root build_workarea
  chmod 775 build_workarea
  chmod g+s build_workarea
  ```

すべてのビルド マネージャ

- 更新中のパラレル通知を選択する。

  Rational Synergy CLI からビルドする場合は、ログファイルにパラレル通知が表示されるように環境を設定します。これで、開発者のパラレル バージョンがチェックイン時にマージされているか判断でき、構成に問題がある場合は警告が受けられます。GUI または CLI のいずれを使用してビルドする場合でも、**メンバーシップ コンフリクトの検出** 操作を行ってパラレル コンフリクト（およびその他のコンフリクト）を検出します。

  Rational Synergy Classic の初期設定ファイルの `[Options]` セクションで、次に示す `reconfigure_parallel_check` オプションを設定し、セッションを再開します。

  ```
  reconfigure_parallel_check = TRUE
  ```

reconfigure_parallel_check を設定すると、一連の更新候補が同じスコアを持って並列になっている場合、ccm_ui.log に以下のような警告メッセージが表示されます。

```
Warning: Parallel versions selected by selection
rules, latest create time will be used:
save.c-3
save.c-2.1.1
```

- このリリースで作業しているチームがどのようにパラレル バージョンを使用するかを決定する、リリース プロパティを設定する。選択肢は、パラレル チェックアウトとパラレル チェックインの両方を許可するか、パラレル チェックアウトのみ許可してパラレル チェックインを禁止するか、またはどちらも禁止するかです。
- ビルド管理ワークエリアを設定する。

  ビルド管理ワークエリアに使用するディレクトリを設定します。ディレクトリは、ユーザー全員がアクセスできる、共有ドライブまたは NFS マウントされたパーティション上に設定する必要があります。

Windows のビルド マネージャ：すべてのビルド マネージャが共有ドライブに対してすべての権限を持ち、同じドライブ文字を使用してドライブをマウントするようにします。

ワークエリア パス テンプレートを共有アクセスに設定

他のビルド マネージャがアクセスできるように、ビルド管理プロジェクトのワークエリアを共有ロケーションに作成するため、ワークエリア パスのテンプレートを設定してください。

ワークエリア パスは、デフォルトで C:¥Documents and Settings¥*username*¥My Documents¥Synergy¥ccm_wa（Windows ユーザー）または *home*/ccm_wa （UNIX ユーザー）です。home はホーム ディレクトリです。

1. **ワーク** ペインで、**ワークエリア** タブをクリックします。
2. **デフォルト パス変更**ボタンをクリックします。

   オプション ダイアログボックスが表示されます。
3. **ベース ディレクトリ** フィールドに、ビルド管理プロジェクトを置く共有ロケーションを指定します。

Windows ユーザーと UNIX ユーザーのどちらも、**+ データベース名** オプションは選択したままにします。これによって、ベース ディレクトリにデータベース名が付加されます。

既存のビルド管理プロジェクトのワークエリア パスを共有ロケーションに設定し直す必要があります。

また、ワークエリアをリリース固有またはプラットフォーム固有のワークエリアに分割できます。これは、ビルド マネージャが多数のプロジェクトをカスタム構造に組み入れるのに便利です。

たとえば、ビルドする **vista** と **winxp** の 2 つのプラットフォームそれぞれに **toolkit** プロジェクトがあるとします。ワークエリア テンプレートを `N:¥network_disk¥Synergy¥shared¥ccm_wa¥%release¥%platform`（Windows ユーザー）または *home*`/ccm_wa/%release/%platform`（UNIX ユーザー）に設定すると、ワークエリアは以下のようになります。

Windows ユーザー：

```
¥user¥local¥shared¥ccm_wa¥shared¥2.8¥vista¥toolkit
¥user¥local¥shared¥ccm_wa¥shared¥2.5¥winxp¥toolkit
```

UNIX ユーザー：

```
/user/local/shared/ccm_wa/2.8/solaris/toolkit
/user/local/shared/ccm_wa/2.5/linux/toolkit
```

環境設定

ビルドするアプリケーションのタイプに合わせて環境設定を行う必要があります。環境設定の方法については、以下のセクションで説明しています。

- 標準リリース
- 複数プラットフォームでのアプリケーション開発
- 複数リリースのためのアプリケーション開発

プラットフォーム ファイルについて

注記：アプリケーションを複数のプラットフォームで開発する必要がある場合、必ずこのセクションとパラレル リリースとパラレル プラットフォームについてを読んでください。

プラットフォームは更新操作でも使用します。更新では、プラットフォーム値を使用してプロジェクトを更新し、最適なオブジェクトに書き換えます。

プラットフォーム名を指定し、続けて 2 つ以上の空白またはタブ、およびセミコロンを入れる必要があります。例：`IBM-AIX`

プラットフォーム ファイルの値は、アプリケーションをビルドするプラットフォームに応じて定義できます。

変更するプラットフォーム ファイルは `om_hosts.cfg` というもので、*`CCM_HOME`*`¥etc`（Windows サーバー）または `$CCM_HOME/etc`（UNIX）にあります。このファイルは、Rational Synergy を使用するすべてのセッションに適用されます。

代わりに、データベースごとに異なるプラットフォーム リストを使用することもできます。このためには、`om_hosts.cfg` ファイルを、*`CCM_HOME`*`¥etc`（Windows サーバー）または `$CCM_HOME/etc`（UNIX）ディレクトリからデータベースの **etc** ディレクトリにコピーし、編集してそのデータベース固有のプラットフォームとホストを設定します。

Windows クライアント ユーザー：データベースごとにプラットフォーム ファイルを定義してください。これは、開発者は通常独自の Rational Synergy インストール エリアを持っているからです。

新しい値を表示するには、変更後、ビルド マネージャとそのデータベースのすべてのユーザーがセッションを再開する必要があります。**プロパティ** ダイアログボックスの **プラットフォーム** リストを使用して新しい値を表示します。

プラットフォーム値の設定は任意です。プラットフォーム値を使用する必要があるのは、複数のハードウェア プラットフォーム／オペレーティング システム上で、または複数のハードウェア プラットフォーム／オペレーティング システム用に、ソフトウェアの複数バージョンをビルドする場合のみです。

プラットフォーム設定

1. Rational Synergy 管理ユーザー（Windows）または *ccm_root*（UNIX）としてログ オンします。
2. *CCM_HOME*¥etc（Windows）、$CCM_HOME/etc（UNIX）、またはデータベースにある om_hosts.cfg om_hosts.cfg ファイルを編集します。
3. ファイルを保存して閉じます。
4. Rational Synergy 管理ユーザー（Windows）または *ccm_root*（UNIX）として終了します。

リリースについて

注記：複数のリリースが必要な場合、必ずこのセクションとパラレル リリースとパラレル プラットフォームについてをお読みください。

注記：Rational Synergy のすべてのプロジェクトとタスクに リリース値を設定してください。

Rational Synergy は、ソフトウェア アプリケーションのリリースを保存します。リリースにより、プロジェクト、タスク、フォルダを特定のリリースに当てることができます。また、リリースは、各リリース用にどのオブジェクト バージョンが開発されたかを把握するのに役立ちます。

ビルド マネージャのみがリリースの作成や変更を行うことができます。**リリース プロパティ** ダイアログボックスでリリースを表示できます。Rational Synergy データベースごとにリリースのセットがあります。Rational Synergy Distributed（DCM）を使用するとデータベース間でリリースを転送できます。

リリースにはチームのプロセスに影響を及ぼす以下のような設定があります。

- リリースでパラレル開発を許可するかどうかを定義できる。
- リリースは、そのリリースに使用できるプロセス ルールを識別することで、チームのプロセスを定義する。

たとえば、典型的なリリースとして以下のようなものがあります。この例では、ビルド マネージャが作成したリリースを示します。これは、コンポーネント名とコンポーネント リリースで構成されます。リリースはユーザーが見るものです。

リリース	コンポーネント名	コンポーネント リリース
1.0		1.0
2.0		2.0
2.0_patch		2.0_patch
Synergy/6.6	Synergy	6.6
editor/2.0	editor	2.0
editor/2.1	editor	2.1

リリース名はコンポーネント名（オプション）とリリース区切り文字（デフォルトはスラッシュ）、およびコンポーネント リリースで構成されます。コンポーネント名はアプリケーションまたはコンポーネントの名前を示します。たとえば、**Synergy** または **editor** などがあります。コンポーネント リリースは、アプリケーションまたはコンポーネントの特定のリリースを識別します。

コンポーネント名はリリースの必須部分ではありません。上記のテーブルの 1 行目で、**1.0** コンポーネント名にコンポーネントがないので、Rational Synergy では空白になります。

新しいリリースを作成する場合、既存のリリースをもとに作成すると、新しいリリースはそのリリースの属性を引き継ぎます。

リリースには任意の 117 文字までの文字列を使用できます （コンポーネント名には 85 文字まで、コンポーネント リリースには 32 文字までの文字列を使用できます）。たとえば、**Integrations/telecom_patch** のようなリリースもあります。

コンポーネント名およびコンポーネント リリースの先頭には、以下の文字を使用できません。

/ ¥ ’ ” : * ? [ ] @ % - + ～ 空白、タブ

2 番目以降の文字列には、以下の文字を使用できません。

/ ¥ ’ ” : * ? [ ] @ %

コンポーネント名とコンポーネント リリースには、バージョン区切り文字（デフォルトは –）を使用できますが、制限文字は使用できません。

オブジェクトがチェックアウトされると、Rational Synergy はカレント タスクから新規オブジェクトにリリースをコピーします。

リリース値の変更

Rational Synergy には、リリース値を変更する方法がいくつかあります。以下の手順については、Rational Synergy ヘルプを参照してください。

- リリースの作成またはコピー

  既存リリースをベースとしないリリースを定義するときは、リリースを作成します。既存リリースをベースに新規リリースを定義するときは、リリースをコピーします。

- オブジェクト（リリース）の削除

  この一般的な削除トピックには、データベースからリリースを削除する場合の説明もあります。

- リリース プロパティの修正

  既存リリースを変更するときは、この手順を行います。

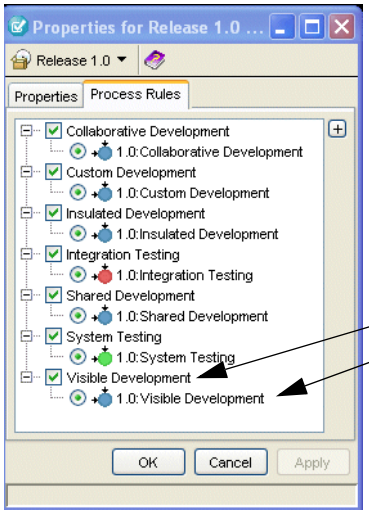
  リリース 7.1a は、ベースラインのコンポーネント タスク（**コンポーネント タスク**）の自動作成をサポートしています。**コンポーネント タスク** オプションは、**リリース プロパティ** ダイアログボックスのチェックボックスです。詳細については、Rational Synergy ヘルプを参照してください。

目的とプロセス ルールについて

「目的」は、同じプロジェクトの複数の *prep*（準備）バージョン、*shared*（共有）バージョン、*working*（作業）バージョン、*visible*（可視）バージョンを、複数のテスト レベルなど、用途別に設定するために使用します。プロジェクトにはそれぞれ目的があります。プロジェクトの目的によってプロジェクトの状態が決まり、更新時に確実に正しいメンバーが選択されるようになります。

通常は、目的を変更する必要はありません。目的の作成が必要なときは、Rational Synergy ヘルプの目的の作成を参照してください。

各リリースにはそのリリースで有効なプロセス ルールのリストがあります。これにより、特定のリリースに対するチームのプロセスを管理し、異なるリリースの作業を行うチームに異なるプロセスを使用させることができます。

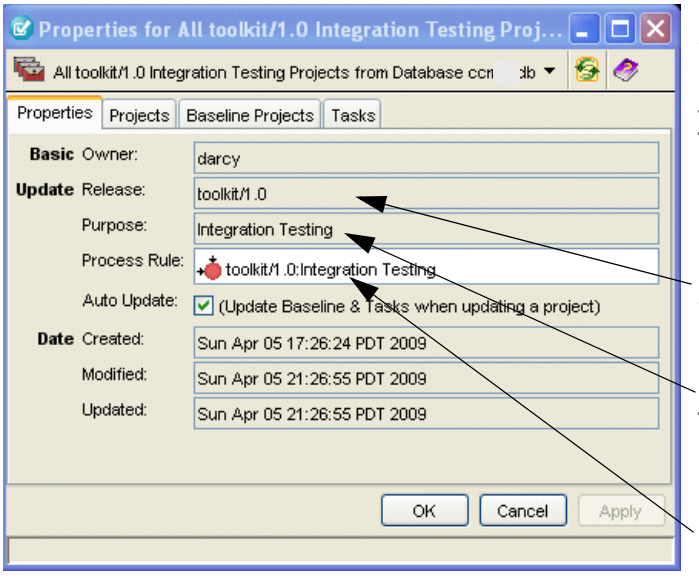


この図は、リリース **1.0** のプロセス ルールを示す **リリース プロパティ** ダイアログ ボックスです。

目的

プロセス ルール

プロセス ルールは、プロジェクトがどのように更新されるかを指定するものです。プロセス ルールを使用するには、プロジェクトにリリースと目的が必要です。プロジェクトのリリースと目的の組み合わせにより、プロジェクトが使用するプロセス ルールを決定します。リリースの目的ごとに 1 つのアクティブなプロセス ルールが存在します。



この図は、**プロジェクト グルーピング プロパティ** ダイアログボックス内のプロジェクト グルーピング「**All toolkit/1.0 Integration Testing Projects from Database ccm71db**」を示します。

プロジェクト グルーピングのリリースは **toolkit/1.0** に設定されています。

プロジェクト グルーピングの目的は **Integration Testing** に設定されています。

このプロセス ルールはベースラインとタスクがどのように選択されるかを定義しています。

Rational Synergy を初めて使用する場合は、デフォルトでデータベースがプロセス ルールを使用するよう設定されています。

Rational Synergy Classic で手動プロジェクトを使用している場合、プロジェクトに手動更新プロパティを使用するように設定された既存テンプレートがあるかもしれません。チームがリリースの途中でプロセス ルールへの変換を行う場合は、プロジェクトの変換の変換プロセスを参照して操作します。変換後、パラレル リリースとパラレル プラットフォームについてを読んで準備を続行します。

新規リリースのためのプロセス ルールの使用

標準目的の名前はリリース 6.5a で変更されました。元の標準目的のバージョンを 6.4a または 6.5a で修正している場合、7.1a へのアップグレードによって、目的のコピーが保存され新しい標準目的の名前に変更されます。あるいは新しい標準目的が作成されます。標準目的を修正した場合は、アップグレード後に、その目的がニーズに合っているかどうか確認してください。合っていない場合は、標準目的を修正せずに新しい目的を作成してください。

個別開発と共同開発

開発者が、統合テストを通過するまで他の開発者の変更を受け取りたくない場合、個別開発を行います。個別開発は、作業中の他の開発者の変更の影響を受けない、安定した環境です。

開発者が、統合テストを通過するのを待たずに他の開発者の最新の変更を受け取りたい場合、共同開発を行います。共同開発では、変更終了と同時に他の開発者の変更を共有できます。

開発者がプロジェクトをチェックアウトするとき、プロジェクトの目的として **Insulated Development**（個別開発）または **Collaborative Development**（共同開発）を選択できます。プロジェクトの目的は**プロパティ** ダイアログボックスで変更できます。プロジェクトの目的により、プロジェクトの更新時に追加されるタスクが決まります。**個別開発**ではプロジェクトに開発者のタスクと最新のテスト済みタスクが選択されます。**共同開発**では、プロジェクトに開発者のタスクと完了済みタスク（テスト済みかどうかに関わらず）が選択されます。

お互いの変更を取り込む時期をテスト後、またはチェックイン後のどちらにするか、チームで選ぶことができます。個別のレベルは、プロジェクトの更新時に選択したオブジェクトによって決まります。プロセス ルールは、目的ごとにプロジェクトの更新方法を定義するパターンです。これは、プロジェクトの更新プロパティを自動設定することにより行われます。たとえば、デフォルトで Rational Synergy には **Insulated Development**（個別開発）と **Collaborative Development**（共同開発）用のプロセス ルールがあります。

これらのプロセス ルールは **Insulated Development**（個別開発）と **Collaborative Development**（共同開発）の目的に対応しています。

コンポーネント開発

コンポーネント開発では、再利用できる部品からアプリケーションを作成します。この開発では、同じ技術上の問題を何度も解決する必要がないため、開発者はより規模の大きなソリューションの実装に注力できます。さらに、Rational Synergy を使用することで、コンポーネントを数多くの構成に取り込ませてそれぞれがユニークなソリューションとなるようにでき、拡張可能な製品を開発できるようになります。

コンポーネントは、個々のファイルやプロジェクトです。Rational Synergy ではファイル バージョンの再利用が可能なので、あるプロジェクトで作成またはビルドしたファイル バージョンを、別のプロジェクトでも使用できます。たとえば、`ccmscci.dll` ライブラリ ファイルは、そのソース コードがある `ccmscci` プロジェクトでビルドできますが、同じファイルを `visual_studio_integration` プロジェクトと `rhapsody_integration` プロジェクトの両方で、メンバーとしてインクルードすることもできます。各プロジェクトには、必要に応じて、異なるバージョンのファイルを入れることができます。

Rational Synergy 7.1a は、プロセスルールにコンポーネント開発用のフォルダまたはフォルダテンプレートを含ませる形でコンポーネント開発をサポートします。Rational Synergy は、コンポーネント開発をサポートするために自動タスクとコンポーネントタスクを作成します。ただし、これらのタスクは、コンポーネント開発の間のみ使用されるため、通常はユーザーからは見えません。

コンポーネント開発プロジェクトで使用される自動タスクとコンポーネントタスクを検索する、フォルダまたはフォルダテンプレートを作成して追加することができます。手順については、Rational Synergy ヘルプのコンポーネント開発用にフォルダまたはフォルダ テンプレートをプロセス ルールに追加を参照してください。

パラレル リリースとパラレル プラットフォームについて

パラレル リリースは、企業でアプリケーションの複数のリリースを同時に開発する必要がある場合に発生します。たとえば、あるチームで **toolkit** アプリケーションのリリース **toolkit/3.0** の新機能に取り組んでいる一方で、別のチームがリリース **toolkit/2.1** のバグ修正の作業をしているような場合です。同様に、複数のプラットフォームで 1 つのリリースを開発する場合があります。

パラレル プラットフォームやパラレル リリースをサポートするには、統合ビルド管理プロジェクト階層とシステム テスト ビルド管理プロジェクト階層をそれぞれ 1 つ、ビルドする一意のリリースとプラットフォームの組み合わせごとに作成します。

まだプラットフォーム ファイルについてとリリースについてを読んでいない場合は、先に進む前に読んでください。

ベースラインについて

ベースラインは、特定の時点におけるユーザーのデータを表現するために使われるプロジェクトとタスクのセットです。ベースラインにはさまざまな用途があります。更新を行うとき、Rational Synergy は新規変更を探す開始点としてベースラインを使用します。

統合テストと**システム テスト**ごとにビルドを作成しておくと、テスターおよび開発者はそのビルドに盛り込まれた一連の変更点を参照できます。通常は、同一のリリースおよび同一の目的を持つすべてのプロジェクトに対して 1 つのベースラインを作成します。たとえば、**統合テスト**用の各ビルドを作成する際には、そのリリースのすべての**統合テスト**プロジェクトを使うでしょう。

詳細については、ベースラインの機能を参照してください。

リリース プロジェクト階層の管理

ビルド管理プロジェクトの作業を実行する前に、開始点として使用するプロジェクト階層を管理する必要があります。通常はアプリケーションの最後に

リリースされたバージョンを使用します。すでに Rational Synergy を使用している場合は、最後に顧客にリリースしたプロジェクト階層からベースラインを作成することになるでしょう。

新規ユーザーでソース コードを Rational Synergy に移行していない場合は、必ず *CM Live!* のドキュメントを読んでください。このドキュメントには、プロジェクト構造についての解説と、ソース コードの移行方法が記載されています。

注記：新規プロジェクトには必ずベースラインを作成してください。

ビルド管理プロジェクトについて

ビルド管理プロジェクトは、テスト エリアやリリースをビルド、テストするためのステージング プロジェクトです。デフォルトでは、ビルド管理プロジェクトは統合テストとシステム テストの 2 つのテスト レベルをサポートします。したがって、アプリケーションのビルド管理プロジェクトを作成するとき、通常は各プロジェクトに少なくとも 2 つのバージョンを作成します。

注記：アプリケーションは通常、プロジェクト階層に分かれた多数のプロジェクトで構成されています。本ガイドでは、ビルド管理プロジェクトつまり統合ビルド管理プロジェクトとは、プロジェクト階層全体（プロジェクト階層がある場合）を意味します。

ビルド管理プロジェクトに目的とプロセス ルールを追加することにより、テスト レベルを追加できます。たとえば、パフォーマンス テスト レベルを追加するには、ビルド管理プロジェクト用の **Performance Testing** という名前の目的を作成し、新しい目的のプロセス ルールを作成し、それに対応するビルド管理プロジェクトのバージョンを作成します。

統合テスト プロジェクトという最初のビルド管理プロジェクトにより、開発者がチェックインした最新の完了タスクを取りまとめ、ビルドし、テストできます。このプロジェクトのメンバーは、すべての完了タスクにクエリを適用することによって取り込まれます。

統合テスト プロジェクトを設定するには、統合テスト プロジェクトの作成を参照してください。

第 2 のビルド管理プロジェクトは**システム テスト プロジェクト**というもので、ここでアプリケーションの取りまとめ、ビルド、テストをさらに細部にわたって行い、合意した品質基準を達成します。このプロジェクトのメンバーは、厳密に管理されたプロセスによって取り込まれます。

システム テスト プロジェクトを設定するには、システム テスト プロジェクトの作成を参照してください。

統合テスト プロジェクトまたはシステム テスト プロジェクトにビルド引数を設定するとよい場合があります。たとえば、統合テスト エリアにデバッグ フラグを設定したり、システム テスト エリアに最適化フラグを設定したりし

てビルドする場合です。ビルドの詳細については、ビルドのガイドラインを参照してください。

注記：統合テストを実行しないプラットフォームがある場合は、そのプラットフォーム向けの統合ビルド管理プロジェクト階層の作成は不要です。

統合テスト プロジェクトの作成

新しいプロジェクト階層をベースラインからコピーして統合テスト エリアを構築します。

1. **クエリ** ダイアログボックスを使用してベースラインを検索します。
2. ベースラインを右クリックし、**プロジェクトのコピー**を選択します。

   プロジェクトのコピー ダイアログボックスが表示されます。
3. **リリース**値を適切なリリースに設定します。

   適切なリリースが**リリース** リストに表示されていない場合は、**選択の再表示**を使用します。必要に応じて、リリースのコピーまたは作成（Rational Synergy ヘルプで説明）を参照してください。
4. **目的** を **Integration Testing**（統合テスト）に設定します。
5. **サブプロジェクトのコピー** リストに、ベースライン用のプロジェクトが表示されます。

   これらのプロジェクト用のビルド管理ワークエリアに使用するつもりの場所の一部が、Rational Synergy を実行しているシステムから見えないこともあります。その場合は見える場所にプロジェクトをコピーします。

   たとえば、Windows と UNIX など、複数のプラットフォームでアプリケーションを開発する場合、Windows プロジェクトと UNIX プロジェクトは、それぞれ適切な Windows マシンと UNIX マシンを使用して個別にコピーする必要があります。また、複数リリース用のアプリケーションを開発する場合は、リリースごとに個別の統合テストプロジェクト階層を作成する必要があります。
6. **バージョン** フィールドに名前を入力して、新しいプロジェクト バージョンをわかりやすい名前に変更します。コピーする複数のプロジェクトがそれぞれ別のバージョンをもつ必要がある場合は、» をクリックして各プロジェクトのバージョンがユニークになるように設定します。

   デフォルトを使用 をクリックし、各エントリに新しいバージョンを入力します。

   バージョンはプロジェクト階層のリリースとプラットフォームを示し、目的が統合テストであることを示すものとします。たとえば、Windows XP® プラットフォームでリリース名 3.0 を使用する統合プロジェクトには、**winxp_3.0_int** のバージョンが適切です。
7. ワークエリア パスを確認し、必要に応じて変更します。

8. デフォルトで、Rational Synergy はプロジェクトのコピー後にすべての新しいプロジェクトを更新します。新しいプロジェクトを更新したくない場合は、**新規プロジェクトの更新**チェックボックスのチェックを外します。
9. **OK** をクリックしてプロジェクトをコピーします。

   統合テスト プロジェクト階層が作成されます。

   注記：プロセス ルールを使用しているため、更新プロパティはすでに適切に設定されています。プロジェクトのリリースと目的により、プロジェクトで使用されるプロセス ルールが決まります。プロジェクトのリリースと目的を正しく設定していれば、更新プロパティ設定のために他に必要な操作はありません。

システム テスト プロジェクトの作成

ベースラインから新しいプロジェクト階層をコピーします。このプロジェクトは、システム テスト エリアの構築に使用します。

1. **クエリ** ダイアログボックスを使用してベースラインを検索します。
2. ベースラインを右クリックし、**プロジェクトのコピー**を選択します。

   プロジェクトのコピー ダイアログボックスが表示されます。
3. **リリース**値を適切なリリースに設定します。

   適切なリリースが**リリース** リストに表示されていない場合は、**選択の再表示**を使用します。必要に応じて、リリースのコピーまたは作成（Rational Synergy ヘルプで説明）を参照してください。
4. **目的**を **System Testing**（システム テスト）に設定します。
5. **サブプロジェクトのコピー** リストに、ベースライン用のプロジェクトが表示されます。

   たとえば、Windows と UNIX など、複数のプラットフォームでアプリケーションを開発する場合、Windows プロジェクトと UNIX プロジェクトは、それぞれ適切な Windows マシンと UNIX マシンを使用して個別にコピーする必要があります。また、複数リリース用のアプリケーションを開発する場合は、リリースごとに個別の統合テスト プロジェクト階層を作成する必要があります。
6. **バージョン** フィールドに名前を入力して、新しいプロジェクト バージョンをわかりやすい名前に変更します。コピーする複数のプロジェクトがそれぞれ別のバージョンをもつ必要がある場合は、[>>] をクリックして各プロジェクトのバージョンがユニークになるように設定します。

   デフォルトを使用 をクリックし、各エントリに新しいバージョンを入力します。

   バージョンはプロジェクト階層のリリースとプラットフォームを示し、目的がシステム テストであることを示すものとします。たとえば、Windows

XP プラットフォームでリリース 3.0 を使用するシステム テスト プロジェクトには、**winxp_3.0_sys** のバージョンが適切です。

7. デフォルトで、Rational Synergy はプロジェクトのコピー後にすべての新しいプロジェクトを更新します。新しいプロジェクトを更新したくない場合は、**新規プロジェクトの更新**チェックボックスのチェックを外します。
8. ワークエリア パスを確認し、必要に応じて変更します。
9. **OK** をクリックしてプロジェクトをコピーします。

   システム テスト プロジェクト階層が作成されます。

3 ビルド管理の基本

サイトのビルド プロセスとは、サイトでテスト エリアを構築し、問題を発見し、質の高い製品をビルドする過程です。ビルド プロセスは、規模に関わらず、すべてのサイトで類似しています。

ビルド プロセスは以下の作業で構成されます。これらの作業は、製品リリースの過程で何度も行うものです。これらの作業にはそれぞれいくつかの手順があります。各作業で手順に従って操作していくと、ビルド管理プロセスが進行します。

- **更新**：ビルドするソフトウェアのセットを収集するため、ビルド管理プロジェクト階層を更新する。

  プロジェクト階層を更新すると、階層のメンバーが更新され、ビルド管理フォルダ内のタスクと関連付けられたオブジェクト バージョンになります。

  更新については、更新のガイドラインを参照してください。
- **コンフリクトを表示し解決**：構成の潜在的問題を特定して修正するため、コンフリクトを表示し解決する。

  コンフリクトとは、プロジェクトの潜在的問題です。ただし、すべてのコンフリクトが必ずしも悪いというわけではありません。コンフリクトを解決すると、プロジェクトの構成の問題を修正できます。
- **ビルド**：最新の完了タスクでアプリケーションをビルドする。

  ビルドについては、ビルドのガイドラインを参照してください。
- **アプリケーションの提供**：ユーザーがアプリケーションを実行してテストできるよう CD やインストール エリアなどの形態にする。

  アプリケーションの提供については、テストのためのアプリケーションの提供を参照してください。

作業を開始する前に

先に進む前に、必ず以下のことを行ったか確認してください。

- プロジェクトでプロセス ルールを使用する設定にした（新規リリースのためのプロセス ルールの使用を参照してください）。
- すべてのプラットフォームとリリースを含み、ビルドとテストを行うすべてのソフトウェアのある統合ビルド管理プロジェクト階層（統合テスト プロジェクトの作成を参照してください）を準備した。
- すべてのプラットフォームとリリースを含み、ビルドとテストを行うすべてのシステム テスト ソフトウェアのあるシステム テスト ビルド管理プロジェクト階層（システム テスト プロジェクトの作成を参照してください）を準備した。
- ビルド メカニズムがこれらのビルド管理プロジェクト階層で正しく機能することを（ビルドのガイドラインを参照してください）検証した。

ビルドについて

以下の 2 つのセクションで、ビルドを行う際に知っておく必要のあること、およびビルド自動化の利点について説明します。

ビルドのガイドライン

- ビルド出力を見直す。

  製品をビルドしたら、ログ ファイルにビルド プロセスの出力を取り込み、注意してログを見直して、ビルドにエラーや問題の徴候がないか調べます（ビルドがバッチ ファイルまたはスクリプトで自動化されている場合は、ログ ファイルに出力を取り込むことができます）。

- makefile の新しいバージョンを見直す。

  ビルド環境は慎重に管理してください。ビルド環境は部分的にマネージャの makefile で定義されているため、開発者がチェックインした makefile を見直し、ビルド環境に悪影響を及ぼすものでないことを確認する必要があります。

  たとえば、ある開発者が makefile をチェックアウトし、ライブラリ ファイルの自分のテスト バージョンを参照するようにカスタマイズし、誤ってその makefile をチェックインする場合などが考えられます（この結果、ビルド マネージャのビルドが開発者のテスト ライブラリを参照するようになり、いずれ問題を起こすことになります）。

- Java™ クラス ファイルを管理しない。

  コンパイルされると、Java ソース ファイルによってクラス ファイルがいくつか生成されます。そのファイルのいくつかは名前が予めわかりません。匿名内部クラスのクラス ファイルは、接尾辞として順番に番号が付けられます。Java クラス ファイルを管理すると、そのような名前を持つオブジェクトの履歴が意味を持たなくなります。別の時刻に作成された別の匿名内部クラスを参照する可能性があるからです。また、クラスを削除した場合、Java コンパイラはそのクラスのために存在していたファイルを削除しません。したがって、関連するオブジェクトをプロジェクトから手作業で削除する必要があります。コンパイル前にすべてのクラス ファイルを削除したとしても、ビルドのたびにそれら制御したクラス ファイル製品はすべて出現して変更されます。

  クラス ファイルを管理する代わりに、`jar` ファイル（必要に応じて `ear` ファイルまたは `war` ファイル）をビルドしてそれを制御します。

- Rational Synergy のスクリプトとツールを管理する。

  製品をビルドするためにスクリプトとツールが必要な場合、そのスクリプトとツール自体を Rational Synergy を使って管理することを考慮すべきです。ソフトウェアの各リリースをビルドするためのツールを正しくバージョン管理して保持することで、旧リリースの再ビルドやパッチビルドが

容易になります。ただし、古いマシンやオペレーティング システムへのアクセスも可能である必要があります。

- **UNIX ユーザー**：

  非管理製品ファイルは、必ず複数のビルド マネージャによって書き込み可能であるようにしてください。

  たいていの場合、チームで Rational Synergy 内ですべての製品を管理することはありません。ライブラリや実行形式ファイルなどの最上位製品を管理し、**.obj** ファイルなどの中間製品は、ファイルが非常に大きくなる場合があるので、管理しないのが一般的です。中間製品を管理すると、複数のユーザーによって製品ファイルのコピーが多数作成されるため、データベースがすぐに大きくなってしまいます。非管理製品は、マネージャのワークエリアにのみ存在するようにします。

  このような製品は管理されないため、Rational Synergy では所有者やアクセス権限を設定しません。したがって、複数のビルド マネージャがワークエリアで非管理製品を更新できるようにするために、いくつかのステップを実行する必要があります。ファイルを作成したビルド マネージャのみが変更できるという権限を付けて非管理製品が作成されると、別のビルド マネージャは、まず非管理製品を削除しなければプロジェクトを再ビルドすることができません。

  非管理製品が複数のビルド マネージャによって書き込み可能であるようにするには、以下のステップを実行します。

  - ビルド マネージャのプライマリ グループを *ccm_root* に設定し、新規ファイルが *ccm_root* グループに作成されるようにします。
  - 各ビルド マネージャの `umask` を、*owner*（所有者）と *group*（グループ）によって書き込み可能であるという権限を付けて新しい製品が作成されるように設定します。
  - プラットフォームやシェルの一部では、自分の `umask` を設定しても、ファイル権限が正しく設定されません。makefile を更新し、`umask` をサポートする Korn シェル（`ksh`）を使用するようにしてもよいでしょう。

    代わりに、ビルド スクリプトまたは makefile を更新し、ビルド前に中間製品を削除するか、そのファイルの権限を変更し、グループを設定してビルド後のグループ書き込みアクセスを許可するようにすることもできます。

ビルド管理プロセスの自動化

バッチ ファイルまたはスクリプトは、更新とビルドのプロセスを自動化する優れた方法です。

自動化には以下のような利点があります。

- ビルド プロセスが毎回同じ方法で実行されるので、再現性が高くなる。
- ビルドの度に詳細を覚えておく必要がないので、エラーが発生しにくくなる。
- 不在時や、夜間などシステムが頻繁に使用されないときにプロセスが実行されるようスケジューリングできる。
- マネージャ不在時は、他の人が代わってビルド プロセスを実施できる。
- バッチ ファイルまたはスクリプト プログラムを作成して、更新とビルドのログを自動的にチェックし、不具合や問題の可能性を示すワードやパターンが見つかったら通知を受けるようにできる（これを自動化しても、ログには目を通す必要があります）。

ビルドを自動化する場合は、問題発生時に診断ができるよう、必ず出力のログをとってください。

注記：ビルドを自動化した場合は、ビルド前にコンフリクトを解決することができません。コンフリクトがある場合、ビルド サイクル（またはその一部）を再度実行する必要が生じることがあります。

テストのためのアプリケーションの提供

アプリケーションを顧客に提供する一般的な方法には、ユーザーがアプリケーションを実行できる DVD、CD-ROM、インストール エリアがあります。

インストール エリアとは、ユーザーが実行できるよう、ソフトウェア アプリケーションのバージョンをインストールできるファイル システム内の場所です。インストール エリアは、テスト、問題の再現、社内での独自のアプリケーション使用に利用します。このエリアは通常、実行形式ファイル、ライブラリ、バッチ ファイルまたはスクリプト、データおよびアプリケーションの実行に必要な環境設定ファイルで構成されます。

ビルド マネージャは、さまざまな用途のために多数のインストール エリアを設定することになります。たとえば、チームが作業に当たっている一般リリースのインストール エリア **rel_int/3.2** などがあります。また、**rel_sqe/3.2** などという名前の、SQE チームが一般リリースをテストする他のインストール エリアもあるでしょう。さらに、サービスパック用のインストール エリア **rel_sp/3.1** などもあるでしょう。

テストのためにアプリケーションを提供する方法にはさまざまなものがあります。一般的な方法の 1 つについてはインストール プロジェクトの作成を参照してください。

ビルド作業フローの使用

サイトのビルド作業フローとは、サイトでテスト エリアを構築し、問題を発見し、質の高い製品をビルドする過程です。ビルド マネージャは、以下を実行する必要があります。

- 頻繁な統合テスト サイクルの完了

  各統合テスト サイクルには、ビルド プロセスのサイクル（更新、コンフリクトの特定、コンフリクトの解決、ビルド、ベースライン）が含まれます。
- いくつかのシステム テスト サイクルの完了

  各システム テスト サイクルには、ビルド プロセスのサイクル（更新、コンフリクトの特定、コンフリクトの解決、ビルド、ベースライン）が含まれます。
- ソフトウェアのリリース
- 新しいリリースの準備

各サイトで、以下の質問に対する答えを元に、テスト サイクルの頻度とテストのレベルを決定する必要があります。

- 製品の品質要件はどうなっているか。
- ソフトウェアの変更の頻度はどの程度か。
- 製品がどの程度リリースに近づいているか。
- 製品のテストにどれだけの時間がかかるか。

以上の質問に対する答えにより、統合テスト サイクルとシステム テスト サイクルを行う頻度が決まります。製品が頻繁に変更される場合は、統合テストを頻繁に行う必要があるでしょう。リリースが近づいていれば、システム テストが主になるでしょう。

この作業フローは実際の組織に合わせて変更することもできます。

統合テスト サイクル

統合テスト サイクルには以下のステップが含まれます。

- 更新
- コンフリクトの特定と解決
- ビルドとテスト
- ベースラインの作成

統合ビルドでは、開発者が新たに完了したタスクがすべてまとめられ、ビルドされます。タスクは、統合テスト プロジェクト階層で使用する統合テスト プロセス ルールに基づいてまとめられます。

この時点ではソフトウェアに問題があることが多いので、ビルドが成功しない場合もあります。目的は直ちに問題を発見することで、高品質なインストール エリアを構築することではありません。開発のこの時点では、ソフトウェアはたいてい不安定です。

統合レベルのビルドに発生する問題でよくあるのは以下のようなものです。

- パラレル ブランチがマージされていない（ある開発者の変更は取得しても、別の開発者の変更が得られない）。
- 開発者が、自分が担当した変更の一部のみをチェックインした（特定のタスクを完了するのに必要な全オブジェクトを関連付けるのを開発者が忘れている場合など）。
- 2 人の開発者が矛盾するような変更を加えた（2 人が同じ名前で定義を追加した場合など）。
- 構文エラーのためにプログラムのコンパイルが失敗した（開発者が単体テストを忘れた場合など）。

統合ビルド エリアは最近完了したタスクを含んでおり、安定した環境でないという点に注意してください。また、開発者が順次タスクを完了していくのに従って候補が頻繁に変わることも理由の 1 つです（これはごく普通のことです）。

統合テスト サイクルを短くし、頻繁に実行してください。開発サイクルのできるだけ早い段階に問題を発見するのに役立つからです。また、個別開発プロジェクトを行う開発者は、タスクが統合テストを通過するまで、お互いの変更を取り込まないでください。

統合サイクルは、ビルドとテストを毎日行い、タスクがテストを通過したらすぐに開発者が使用できるようになっている状態で最もよく機能します。

通常、統合レベルのビルド サイクルには以下のステップが含まれます。

1. 開発者は、変更を加え、タスクを完了することによってチェックインするという作業を継続的に行い、サイクルには注意を払いません （この利点は、チームの作業がテストによって中断されたり、混乱したりしないことです）。

2. ビルド マネージャは、更新を行い、コンフリクトを特定し、コンフリクトを解決し、階層を構築し、テストのための新しいインストール エリアやメディアを作成します （この一部は自動化可能で、毎晩の作業として実行できます)。
3. ビルド マネージャは、結果としてできた製品に対して小規模のテストを実施して、製品が正しくビルドされていて使用に耐えることを確認します。不具合が見つかった場合は、チームのメンバーが問題を修正するためのタスクを作成します。
4. 深刻な不具合が見つからなければ、アプリケーションは開発者のテスト エリアなどに使用できる状態になっています。いつもそうであるとかぎらないので注意してください。深刻な不具合が発見されることも、ビルド自体が成功しないこともあります。
5. 深刻な不具合を発見しなければ、ビルド マネージャは続けてベースラインの作成を行うことができます。これで、ベースラインのタスクに関連付けられたオブジェクトは、開発者が次に自分のプロジェクトを更新する際に使用できるようになります。

統合テスト サイクルでマネージャが何を行い、これらの作業をなぜ実行する必要があるかを理解できたら、実際の作業に移ります。

更新

ここで、ビルド管理プロジェクト階層を更新できます。手順については、プロジェクトの更新を参照してください。

コンフリクトの特定と解決

ビルド管理プロジェクト階層を更新しました。これで、アプリケーションをビルドする**前に**構成の潜在的問題を特定し、解決できます。手順については、コンフリクトの検出を参照してください。

ビルドとテスト

更新とコンフリクトの処理が完了したら、アプリケーションのビルドを行います。ビルドはサイトによって大きく異なります。ビルドの際は、以下の一般的なガイドラインを参照してください。

1. ビルドが完了したら、ビルド ログを確認します。
2. インストール エリアまたはテスト メディアを作成します。
3. 一連の短いテストを実行してテストします。
4. ビルドに失敗した場合は、選択したタスクを追加して再ビルドするか、修正タスクを追加して更新からビルド プロセスを開始します。詳細については、特定のタスクを使用するビルドを参照してください。

ベースラインの作成

統合テスト プロジェクトを更新し、コンフリクト処理、製品のビルド、アプリケーションのテストが完了しました。次に開発者が変更を使用できるようにします。ベースラインの作成を参照してください。

詳細については、ベースラインの機能を参照してください。

問題のあるベースラインの処理

ベースラインに問題があり、開発者が使用すべきではない場合があります。このような場合、以下のいずれかのステップを行います。

- 修正がある場合、それを含む新しいベースラインを作成する。

または

- 以前のベースラインに戻す。

  不要なベースラインの削除を参照してください。ベースラインに削除のマーク付けをする方法、およびオフライン保存と削除コマンドの使用方法が記載されています。ベースラインに削除のマークを付けると、開発者はそのベースラインが使用できなくなります （すべての開発者が自分のプロジェクトの更新を完了してから、問題のあるベースラインを削除できます)。

システム テスト サイクル

システム テスト サイクルには以下のステップが含まれます。

- 更新
- コンフリクトの特定と解決
- ビルドとテスト
- 特定のタスクを使用するビルド
- ベースラインの作成
- ソフトウェアのリリース（省略可）
- 新しいリリースの準備（省略可）

システム テスト サイクルにより、開発者が継続的に行う変更と独立して特定のタスク セットをさらに詳細にテストできます。目的は、インストール エリアを作成するか、品質基準を満たすリリース メディアを作成することです。

システム テスト プロジェクトに追加するタスクは選択できるので、継続して行われる変更とは独立したものになります。これで、合意した品質基準に達するまで、継続して行われる変更とは別にソフトウェアのビルド、修正、テストを行うことができます。

ソフトウェアはシステム テスト レベルに達するまでにほとんどの統合上の問題が解決されるため、システム テスト エリアはより安定しており、構築が容易です。

たいてい、システム テスト サイクルは、リリースなどのマイルストーンの準備に使用されます。システム テスト サイクルの頻度とテストのレベルは、以下のように状況によって異なります。

- リリース サイクルの開始時点では、多くの新機能が追加されると、システム テストがたまにしか行われない場合があります。これは、新しいテスト ケースの開発が必要となることが多く、テストに時間がかかり、困難になるからです。

  また、ソフトウェア開発はまだ進行中であるため、目的は、安定したインストールやリリースの作成ではなく、不具合の発見と新しいテストの開発になります。
- 開発フェーズが終了し、チームがソフトウェアの安定化と不具合の修正に取り組んでいる場合、システム テストを頻繁に（たとえば 1 週間に 1 度か 2 度）行うことが多いでしょう。
- 開発サイクルの最後では、ソフトウェアのイタレーションそれぞれがリリースされるものになり得るため、1 つ 1 つの変更をテストする必要があるでしょう。

以下に、システム テスト ビルド サイクルの流れの概要を説明します。

1. システム テスト ビルド管理プロジェクト階層を更新し、コンフリクトを特定、解決し、ビルドを行います。クリーンなシステム テスト エリアを

得る必要があるため、コンフリクトの特定と解決は慎重に行います。次に、テストする新しいシステム テスト インストール エリアまたはメディアを作成します。

2. できた製品をテストします。不具合が見つかったら、タスクを作成します。
3. システム テスト エリアの品質基準を満たすために必要な不具合修正のため、以下を実行します。
   - どの問題を修正するかをプロジェクト チームで決定する。
   - 問題を修正するため、開発者に新しいタスクが割り当てられる。
   - 開発者は、新しいタスクをカレント タスクに設定して問題を修正する。
   - 開発者がカレント タスクを完了し、タスクが完了したことをビルド マネージャに通知する。
   - ビルド マネージャが完了タスクをシステム テスト フォルダに追加する（ステップ 1 に戻る）。
4. 合意した品質基準がシステム テスト エリアによって達成されたら、一般に使用できるようにするか、顧客にリリースします。リリースのその時点でプロジェクトを含める方法については、ベースラインの作成を参照してください。

更新

ここで、ビルド管理プロジェクト階層を更新できます。手順については、プロジェクトの更新を参照してください。

コンフリクトの特定と解決

ビルド管理プロジェクト階層を更新しました。これで、アプリケーションをビルドする**前に**構成の潜在的問題を特定し、解決できます。手順については、コンフリクトの検出を参照してください。

ビルドとテスト

このトピックについては、ビルドとテストを参照してください。

特定のタスクを使用するビルド

テストが完了し、ソフトウェアの品質が承認されたら、ソフトウェアのリリースを参照してください。

不具合が見つかり、階層に組み入れる特定のタスクをプロジェクト チームが決定したら、その承認タスクを適切なプロジェクト グルーピングに追加する必要があります。

1. 適切なプロジェクト グルーピングを右クリックし、ベースラインとタスクの自動更新の選択を解除します。

   これにより、更新操作時にベースラインとタスクが変更されなくなります。ベースラインとタスクは、プロジェクト グルーピングの一部です。したがって、プロジェクトを更新するときに選択したベースラインと新しいタスクによって自動更新したくない場合、プロジェクトが含まれるプロジェクト グルーピングでこのオプションをオフにする必要があります。

2. 承認されたタスクを以下の方法でプロジェクト グルーピングに追加します。

   - 承認されたタスクをプロジェクト グルーピングにドラッグ アンド ドロップする。
   - 承認されたタスクを右クリックし、プロジェクト グルーピングへ追加する。
   - **プロジェクト グルーピング プロパティ** ダイアログボックスの**タスク** タブを使用する。手順については、Rational Synergy ヘルプの再ビルドの実行の「ビルドにタスクを追加」を参照してください。

3. 適切なプロジェクト グルーピングを右クリックし、更新をポイントし、**すべてのプロジェクト**を選択します。

   更新操作は、新しいタスクから変更を取り込みます。

4. コンフリクトを特定し、解決します（詳細については、コンフリクトの解決を参照してください）。

5. 製品を再ビルドします。

6. ベースラインの作成を参照してください。

   ビルドの品質に満足したら、テスト ベースラインを作成できます。このテスト ベースラインは、SQE が使用できるように、また開発者がビルドの変更を確認できるように、保存したビルドのコピーであり、チーム全員が使用できるように公開、リリースしたものではありません。

   適切なプロジェクト グルーピングに対して**更新**はまだ無効になっています。プロセス ルールで指定されたタスクを受け入れる準備ができたらそれを有効にできます。

次のリリースまたはマイルストーンに含めるすべてのタスクが、プロジェクト グルーピングに入りました。これで、承認済みの修正を使用して別のビルド サイクルを開始できます。

ソフトウェアのリリース

アプリケーションがシステム テストを通過したら、ベースラインとすべてのオブジェクトをリリースできます。

ベースラインを作成していれば、そのベースラインをリリースできます。ベースラインを作成する必要がある場合は、ベースラインの作成を参照してください。以下に、ベースラインをリリースする手順を説明します。

- システム テスト ベースラインを選択し、ベースラインを右クリックして**リリース**を選択します。

  この時点で、製品を顧客に納品する準備を行うことができます。

また、ベースラインを公開したときと同様に、ベースラインの名前の変更、ベースラインにあるプロジェクトと製品のバージョンを変更できます。これらの操作は CLI からのみ一緒に実行できます。

1. コマンド プロンプトから Rational Synergy を開始します。

   ```
   ccm start -h engine_host -d database_path -nogui
   ```

   セッションの開始後、コマンド ウィンドウ（Windows）またはセッションを開始したシェル（UNIX）に Rational Synergy アドレス（CCM_ADDR）が表示されます。

2. ロールを *build_mgr*（ビルド マネージャ）に設定します。

   ```
   ccm set role build_mgr
   ```

3. 状況に応じて、ベースラインの名前またはそのプロジェクトのバージョンを変更します。

   バージョンを分かりやすい名前に変更しておくと、名前を見たときにプロジェクトの目的を思い出しやすくなります（開発者にテスト ベースラインを公開のステップ 2 のバージョン テンプレートについての説明を参照してください）。

   また、リリースの目的が分かるようなベースライン名を使用できます。ベースライン名は作った後に変更できます。

   ```
   ccm baseline -modify "7.1 Turn 10" -name "7.1 General
   Release" -versions
   -vt "%{platform:-}%{platform:+_}%{release}_GR"
   ```

4. ベースラインをリリースします。

   ```
   ccm baseline -release_baseline "7.1 General Release"
   ```

新しいリリースの準備

アプリケーションをリリースしたら、次のリリースに着手できます。以下の操作を完了する必要があります。

- 新しいリリースを追加します。
- すべての未完了タスクのリリースを更新します。
- 新しい統合テスト プロジェクトをまだ作成していない場合は、統合テスト プロジェクトを再利用します。
- システム テスト プロジェクトの作成を行います。
- 開発者に開発プロジェクトを再利用するように伝えます。

1. 新しいリリースを追加します。

   新しいリリースを追加し（リリース値の変更を参照）、新しいリリースに使用するプロセスを選択し、新しいリリースのベースライン リリースを選択します。

2. 新しいリリースのプロセス ルールが正しく設定されていることを確認してください。

3. すべての未完了タスクのリリースを更新します。

   開発サイクル中、一部のタスクは現行リリースに組み込みません。組み込まないようにする方法の 1 つが、タスクを完了しないことです。新しいリリースでタスクを完了したときにそれを取り込めるようにするには、タスクのリリース値を更新して、更新でタスクがプロジェクト構成に含める候補と見なされるようにする必要があります。

4. すべての完了タスクのうち、リリースしたばかりの製品には含まれなかったもののリリース値を更新します（さらに、変更依頼のリリース値も必ず更新します。このためには、Rational Change を使用します）。

   Rational Change ユーザー

   - **このビルドに含まれない CR** というクエリを選択し、Rational Change でクエリを実行し、現行リリースに設定されているが、ビルドに含まれていない変更依頼を表示します。このクエリを実行する前に特定のリリース情報の入力が必要です。

   または

   Rational Synergy ユーザー

   a. **クエリ** ダイアログボックスを表示します。

      検索 > タスクをクリックします。

      クエリ ダイアログボックスで、以下のカスタム クエリを入力します。以下のクエリは、当該リリース（*baseline_release*）と目的（*baseline_purpose*）を持つ最新のベースラインに含まれない、当該リリース（*task_release*）のすべての完了タスクを返します。

`is_available_task_of_release(`*`task_release`*`,`
`baseline_release``,`*`baseline_purpose`*`)`

さらに、*test_baseline*、*published_baseline* または *released* 状態のすべてのベースラインから最新のベースラインが選択されます。ベースラインに含まれるべきタスクは、ベースラインを構成するすべてのプロジェクトの更新プロパティに含める必要があります。

クエリの結果、リリースの最終ビルド後に完成したが、そのリリースに組み込まれなかったタスクが表示されます (これらのタスクに新リリース用とラベルします)。

b. リリースに含むべきタスクを適切なプロジェクト グルーピングに追加する必要がある場合は、Rational Synergy ヘルプの プロジェクト グルーピングにタスクを追加 を参照してください。

5. 新しい統合テスト プロジェクトをまだ作成していない場合は、統合テスト プロジェクトを再利用します。

a. 最上位の統合テスト プロジェクトを右クリックし、**プロパティ**を選択します。

プロパティ ダイアログボックスが表示されます。

b. リリースにバージョンを付ける場合は、バージョンを変更してください。

c. **OK** をクリックして変更を保存します。

プロジェクトにサブプロジェクトがある場合、サブプロジェクトにもリリースがコピーされます。

6. 新しいシステム テスト プロジェクトをまだ作成していない場合は、システム テスト プロジェクトを再利用します。

7. 開発者に開発プロジェクトを再利用するように伝えます。

開発者はプロジェクト階層のリリース値を変更する必要があります。

ベースラインの削除のマーク付け

新しいリリースの準備が終わったら、不要となったベースラインを整理します。リリースの過程で、特に統合テスト フェーズでは、複数のベースラインが作成されます。データベースが煩雑となるのを避けるためにも、不要となったベースラインを削除します。

手順については、不要なベースラインの削除を参照してください。

4 更新とコンフリクト

更新操作

更新操作によりプロジェクトが更新され、更新プロパティを満たす最新の変更が反映されます。更新プロパティは 1 つのベースライン プロジェクトと一連のタスクから構成されます。プロジェクトを手動で更新するか、プロセス ルールを使用して更新するかによって、ベースライン プロジェクトと一連のタスクの算出方法が異なります。これらは別個のプロセスなので、個別に説明します。

いずれの場合も、ベースライン プロジェクトとタスクの算出後に以下のステップが実行されます。

1. ベースライン　プロジェクトとタスクに基づいて候補のリストを作成します。
   a. ベースライン プロジェクトの各メンバーが候補となります。
   b. 各タスクに関連づけられている各オブジェクトのバージョンが候補となります。

      このステップは**プロジェクト**ごとに 1 度実行されます。
2. 候補が収集されると、単純な選択ルールセットを使用し、各ディレクトリ エントリに最適なオブジェクト バージョンが選択されます。

   このステップはプロジェクトの**ディレクトリ エントリ**ごとに 1 度実行されます。
3. 更新の際に使用されるのは、ディレクトリ　エントリ内の選択されたオブジェクト バージョンです。

   選択を変更した場合、このステップはプロジェクトの**ディレクトリ エントリ**ごとに 1 度実行されます。

プロセス ルールの更新

プロジェクトでプロセス ルールを使用している場合、ベースライン プロジェクトとタスクはそのプロジェクトのプロジェクト グルーピングのベースラインとタスクから計算されます。プロジェクト グルーピングのベースラインはベースライン プロジェクトの集合です。プロジェクト グルーピングのタスクは、更新の際に使用されるタスクと完全に同じです。更新で使用されるベースライン プロジェクトは、更新対象のプロジェクトと一致する、プロジェクト グルーピングのベースライン内のプロジェクトです。あるベースライン プロジェクトが一致していると考えられるのは、それが更新されるプロジェクトと同じ名前、インスタンス、プラットフォームを持つ場合です。

プロジェクト グルーピングのベースラインとタスクは、以下のように計算されます。

1. プロジェクト グルーピングの自動更新がオフになっていると、プロジェクト グルーピングで以前に計算して保存されたベースラインとタスクが使用されます。

   しかし、プロセス ルールでベースライン プロジェクトを **最新プロジェクト** メソッドで選択するよう指定している場合、プロジェクト グルーピングはベースラインを持たず、ベースライン プロジェクトは自動更新がオンになっている場合と同じ方法で再計算されます。

2. プロジェクト グルーピングの自動更新がオンになっている場合、ベースラインとタスクは以下のように計算されます。

   a. ベースラインは、そのプロジェクト グルーピング用のプロセス ルールに基づいて計算されます。プロセス ルールは、使用するベースライン（**最新のベースライン**、**プロセス ルールに指定されたベースライン**、**最新プロジェクト**、または**プロジェクト グルーピングに指定されたベースライン**）を決定するためのルールを指定します。プロジェクトの更新時に、プロセス ルールのベースライン選択ルールで選択したベースラインから、ベースライン プロジェクトが特定されます。

   プロセス ルールでベースライン プロジェクトを **最新プロジェクト** メソッドで選択するよう指定している場合、プロジェクト グルーピングはベースラインを持たず、各プロジェクトが、プロセス ルールで指定したリリースと目的に合った最新の静的プロジェクトであるベースライン プロジェクトを持つことになります。

   b. タスクは、プロジェクト グルーピング用にインスタンス化されたプロセス ルールで指定したフォルダとタスクから計算されます。

   * 各クエリベースのフォルダについて、クエリを再評価してフォルダ内のタスクを更新し、すべてのフォルダにあるタスクをリストアップします。

   * プロジェクト グルーピングがベーラインを持っている場合、算出したタスクセットからベースラインにあるタスクが差し引かれます。

   * プロジェクト グルーピングに手動で追加したまたは削除したタスクがある場合、算出したタスク セットにこれらのタスクが追加または削除されます。

手動による更新

ベースラインとタスクを手動で選択する必要がある場合は、カスタム目的を使用してください。カスタム目的を使用すると、プロジェクト グルーピングで直接ベースラインを選択できます。その後、適切なタスクを選択して右クリックし、**プロジェクト グルーピングに追加**を選択してタスクを手動で追加できます。

更新のガイドライン

以下に、更新時にビルド マネージャが考慮すべきガイドラインを示します。

- ビルドするすべてのプロジェクトを更新する。

  1 回の操作でビルド管理プロジェクト階層全体を更新する必要があります。これで、コンフリクトのないバージョンをアプリケーションに組み込むことができます。更新されないプロジェクトがあると、そのためにビルドされた製品が、他のビルド製品と両立しない場合があります。

  プロジェクト階層を一度の操作でビルドすることが現実的ではない場合、段階的ビルドのためのビルド プロセスを参照して段階的なアプローチをとってください。

  注記：別バージョンの使用操作を更新の代わりに使用しないでください。特定のバージョンを簡単なテストに使用する必要がある場合もありますが、不完全な変更を取得する可能性をなくし、完全でコンフリクトのないプロジェクト構成を得るには、完全な更新を行う必要があります。

プロジェクトの更新

アプリケーション開発のこの時点で、ビルド管理プロジェクト階層の更新の準備ができています。

1. 適切なプロジェクトを右クリックし、**更新**をポイントし、**メンバーとサブプロジェクト**を選択します。
2. 更新が完了したら、必ず結果を確認します。

   更新操作中に問題が生じた場合は、より詳細な出力情報を読む必要があるでしょう。選択時の問題の診断を参照してください。

選択ルールの作業

プロジェクトまたはディレクトリの更新の際、更新操作はプロジェクトの各オブジェクトに使用可能な候補を対象とし、そのプロパティを「platform」などプロジェクトのプロパティと比較して、最適な候補を選択します。

選択ルールでは、各候補オブジェクト バージョンが分析されます。

- オブジェクト バージョンのプロパティがプロジェクトと矛盾すると、不適格と見なされ、選択**されません**。

  たとえば、ユーザー *joe* が *working*（作業）状態のオブジェクト バージョンを持っており、ユーザー *tim* が更新を実行した場合、ユーザー *joe* の *working*（作業）状態のオブジェクト バージョンはユーザー *tim* のプロジェクト構成に含まれません。
- オブジェクト バージョンには特徴によってそれぞれポイントに基づくスコアが付けられます。ポイントは累積的に付けられます。例：

  working（作業）オブジェクト バージョンには、5 ポイントの状態スコアが付けられます。

  プラットフォームがプロジェクトのプラットフォームと一致するオブジェクト バージョンには、8 ポイントが付けられます。

  すべての候補オブジェクト バージョンの分析後、更新によって、最も高いスコアを持つ候補が選択されます （不適格な候補は対象になりません)。

複数の候補に同一の最高スコアが付けられた場合は、最新のものが選択されます。

> **注記**：Rational Synergy GUI または CLI のいずれを使用してビルドする場合でも、**メンバーシップ コンフリクトの検出**操作を行ってパラレル コンフリクト（およびその他のコンフリクト）を検出します。複数の候補がパラレルの場合は、**メンバーシップ コンフリクト** ダイアログボックスにパラレル通知が表示されます。
>
> これは通常、開発者が変更のマージを忘れたか、パラレル

ブランチのいずれかで選択プロパティが正しい値に設定されていない場合に起こります。

コンフリクト検出については、コンフリクトの検出を参照してください。

更新とベースライン

プロセス ルールに従ってプロジェクトを更新する際、使用すべき**ベースライン プロジェクト**を再評価します。ベースラインと更新の詳細については、ベースラインと更新プロセスを参照してください。

特定リリースとすべてのプロジェクトと prep（準備）目的を含む完全なベースラインを作成することが不可能または現実的ではない場合があります。その場合は、増分ベースラインの作成を参照してください。

プラットフォーム値を使用した更新

プラットフォーム値を持つプロジェクトを更新する場合、一致するプラットフォーム値を持つ候補が優先されます。プラットフォーム値が**一致しない**候補は選択されませんが、プラットフォーム値が設定されていない候補は選択されることがあります。

更新時、候補のプロパティは、プロジェクトのプロパティと比較されます。プラットフォーム値は以下のように比較されます。

- プロジェクトと候補の両方にプラットフォーム値が設定されていてプラットフォーム値が一致しない場合、更新によって候補が選択されることは**ありません**。
- プラットフォーム値が一致すると、その候補は**優先**候補となり、8 ポイントが付けられます。
- プロジェクトと候補のどちらにもプラットフォームがない場合、その候補は**優先**候補となり、8 ポイントが付けられます。

  プラットフォーム プロパティは、主にプロジェクトと製品に使用されます。通常、ソースコードは、（`#ifdef` などを使用して）同じファイルを別のプラットフォームで構築できるように作成されます。したがって、個々のソース ファイルには、**プラットフォーム** プロパティを設定する必要はありません。

更新の結果の検討

- 更新の結果を検討し、問題がないか確認する。

  更新中、出力がセッション ログ ファイルと**メッセージ** ダイアログボックスに書き込まれます。しかし、このログ ファイルには、更新の結果だけでなく他のすべてのメッセージも書き込まれるため、更新の結果を読み出すのは面倒です。

  ccm_client.log（ユーザー インターフェイス ログ）ファイルの出力先を Windows プロファイル ディレクトリ（Windows ユーザー）または ccmlog ディレクトリ（UNIX ユーザー）以外に変更できます。このためには、ccm.user.properties ファイル内の ccm.user.properties キーを以下のように設定します。

  1. プロパティ ファイルを開きます。

     Windows ユーザーの場合、このファイルは ccm.user.properties という名前で、Windows プロファイル ディレクトリにあります。

     UNIX ユーザーの場合、このファイルは ccm.user.properties という名前で、ホーム ディレクトリにあります。

     * ログ ファイルの出力先を C:¥cmsynergy¥synint¥bob に変更するには、以下のように指定します。

     ```
     user.default.logfile=
     C:¥¥cmsynergy¥¥synint¥¥bob¥¥ccm_client.log
     ```

     * int という名のデータベースのログファイル（主に複数のデータベースを使用している場合）の名前を変更するには、以下のように指定します。

     ```
     user.default.logfile=
     C:¥¥cmsynergy¥¥bob¥¥ccm_client_int.log
     ```

  2. ファイルを保存して終了します。

     ccm.user.properties キーを使用する場合、上記の例のようにフル パスとファイル名を使用する必要があります。

     また、Windows のパスはダブル円記号を使用して入力する必要があります。

  更新／ビルド サイクルのたびに更新の結果を確認し、問題がないか確認します。更新メッセージの終わりには、Rational Synergy により**メッセージ** ダイアログボックスまたは出力ログに更新が成功したか失敗したかを示すサマリが書き込まれます。しかし、ログを参照して更新失敗に関する詳細レポートを読むようにしてください。

  また、ビルドが成功しても、ソフトウェアが正しく構築されているとはかぎりません。更新の結果を確認することは、オブジェクトまたはプロジェクトの誤バージョン、マージされていない変更、誤った選択プロパ

ティ設定などの構成エラーを見つけるよい方法です。以下のような点をチェックします。

- パラレル バージョンがないか確認する。

  Rational Synergy CLI からビルドしており、`reconfigure_parallel_check` オプションを設定すると、所定のセットの更新候補が同じスコアを持っている場合、Rational Synergy Classic の `ccm_ui.log` に以下のような警告メッセージが表示されます。

  ```
  Warning: Parallel versions selected by selection
  rules, latest create time will be used:
  save.c-3
  save.c-2.1.1
  ```

  警告メッセージに示されたオブジェクトの履歴を参照し、マージされるべきパラレル バージョンがマージされているかどうか確認します。マージされていない場合は、ビルドでは変更の一部が欠落しています。パラレル バージョンをマージする必要があることを関係開発者に知らせてください。

  Rational Synergy または Rational Synergy CLI からのビルド時にパラレル バージョンを調べる方法については、更新中のパラレル通知を選択する。 を参照してください。

- 置き換えられたサブプロジェクトがないか確認する。

  ビルド管理プロジェクト階層は、ユニットとしてまとまっている必要があります。更新プロパティ（リリース、プラットフォームなど）が誤って設定されている場合は、更新により、サブプロジェクトの別バージョンが選択される場合があります。以下のようなメッセージがないか確認してください。

  ```
  Subproject editor-int_3.0 replaces editor-int_2.1
  under toolkit-2:dir:1
  ```

  置き換えられたプロジェクトに関するメッセージが見つかった場合は、そのプロジェクト バージョンを調べます。更新プロパティを調べて正しいことを確認します。

- 空のディレクトリ エントリがないか確認する。

  デフォルトでは、候補がないと、ディレクトリ エントリは空のままになります。このようなディレクト リエントリが見つかったら、間違ったリリース値を持ったタスクなど、原因を調べます。以下のようなメッセージがないか確認してください。

  ```
  2 directory entries were left empty because they had no
  candidates.
  ```

  ディレクトリ エントリが空でも、エラーであるとはかぎりません。たとえば、1 つのディレクトリ内で複数プラットフォームの製品をビルドした場合、ディレクトリ エントリが空のままになることがあります。共有ラ

イブラリは Solaris™ では「mylibrary.so」、Windows では「mylibrary.dll」という名前になります。両方の製品を同じディレクトリで管理し、2 つのプラットフォームの 2 つのパラレル プロジェクトを使用する場合、Windows プロジェクトの Solaris ライブラリ、Solaris プロジェクトの Windows ライブラリは空になります。

- 置き換えられた makefile がないか確認する。

  開発者が自分の環境に固有の設定で makefile をカスタマイズし、誤ってそのカスタム makefile をチェックインしてしまったことが考えられます。更新プロセス中に makefile が置き換えられた場合は、makefile の新しいバージョンを調べ、変更がビルド環境に適切なものであるか確認します。以下のようなメッセージがないか確認してください。

  ```
  'makefile-6:makefile:3' replaces 'makefile-
  5:makefile:3' under 'editor-2:dir:1'
  ```

- ワークエリア コンフリクトがないか確認する。

  プロジェクトにコンフリクトがある場合、**ワークエリアの同期**を使用してワークエリア コンフリクトを解決し、その後で再度更新する必要があります。以下のようなメッセージがないか確認してください。

  ```
  Unable to update membership of project
  ccm_client,td_7.1 with
  InteractiveProcessCreator.java,21:java:J#1 due to work
  area conflicts.
  ```

- 新しいオブジェクト バージョンが古いオブジェクト バージョンに置き換えられていないか確認する。

  更新プロセス中に古いオブジェクト バージョンによって新しいオブジェクト バージョンが置き換えられる場合は、Show Conflicts 操作を実行して問題が発生しないようにします。以下のようなメッセージがないか確認してください。

  ```
  'foo.c-2:csrc:3' replaces 'foo.c-3:csrc:3' under
  'toolkit-4:dir:1'
  ```

  さらに、新しいオブジェクト バージョンが関連付けられているタスクを探し、プロジェクトのプロセス ルールを探します。これらを比較すると、新しいバージョンが古いバージョンに置き換えられた理由が分かる場合があります。タスクが古いオブジェクト バージョンを追加した原因となった、プロセス ルールの差異が分かります。

  このような問題が見つかり、詳細がわからない場合は、詳細メッセージを有効にしてもう 1 度更新を行い、出力ログに書き込まれる追加の更新結果を参照します（**オプション** ダイアログボックスの**アクション** タブの**更新**オプションで**詳細メッセージを表示**を設定します）。

以下のセクションで、更新についてさらに詳しく説明しています。

- プロジェクトの更新
- 選択ルールの作業
- プラットフォーム値を使用した更新
- 選択時の問題の診断
- 更新プロパティの検証

選択時の問題の診断

更新中にオブジェクト バージョンが選択された理由、または選択されなかった理由を確認する必要が生じる場合があります。ビルド管理プロジェクト階層で発生したことや、開発者の開発プロジェクトの問題解決に役立てるため、理由を特定する必要が生じる場合があります。

以下の項目を順序どおりに確認してください。

1. プロジェクトのプロセス ルールで以下のことを確認します。
   - フォルダ テンプレートのタスク クエリが正しいこと。
   - このプロセス ルールに正しいフォルダまたはフォルダ テンプレートが含まれていること（プロジェクトのフォルダまたはフォルダ テンプレートが正しくない場合、プロセス ルールを更新する必要があるかもしれません）。
   - ベースラインが設定されていること。
2. `verbose` オプションを使用して更新操作を実行します。`verbose` オプションにより、分析する候補の詳細情報が得られます。各候補のスコアとそのスコアの根拠が表示されます。得られた情報を使用してトラブルシューティングを行います。

   注記：不確かなオブジェクト バージョンのあるディレクトリ**のみ**に対する冗長更新実行が可能です。最上位プロジェクトからよりずっと速く冗長更新を行うことができます。

3. プロジェクト グルーピングのプロパティを確認します。
   - **自動更新**をオフにしていた場合は、オンに戻す。
   - タスクを一時的に削除していた場合は、それを元に戻したか確認する。
   - タスクを手動で追加した場合は、そのタスクを保持したいか確認する。
   - ベースライン プロジェクトが設定されているか確認する。
4. プロセス ルールを比較して正しく設定されていることを確認します。
   a. プロジェクトまたはプロジェクト グルーピングを右クリックし、**プロセス ルール プロパティ**を選択します。
   b. **プロセス ルール プロパティ** ダイアログボックスで、オブジェクト メニュー（左上の角）をクリックし、**プロセスのプロセス ルールと比較**を選択します。
5. 問題があるプロジェクトを右クリックし、**プロパティ**を選択します。ベースライン プロジェクトが適切であることを確認します。
6. **プロセス　ルールに指定されたベースライン**が設定された汎用プロセス ルールを使用しており、それをリリースに追加した場合、プロセス ルールのベースラインを指定する必要があります。指定しないと、そのプロセス ルールを使用するプロジェクト グルーピングにベースラインが設

定されず、正しく更新されません。**プロセス ルール プロパティ** ダイアログボックスでベースラインを指定します (Rational Synergy ヘルプのプロセス ルールのプロパティの修正を参照してください)。

以上を実行しても選択上の問題が解決されない場合は、以下のことを試します。

- 2 つのリリースのプロセス ルールを比較する。

  直前のリリースでプロジェクトが正しく更新されているのに新リリースで正しく更新されない場合は、2 つのリリース間でプロセス ルールを比較します。

  Rational Synergy ヘルプの 2 つの類似オブジェクトの比較を参照してください。
- 現在のリリースのプロセス ルールをプロセスのプロセス ルールと比較する。
- 2 つのプロジェクトの更新プロパティを比較する。

  目的とリリースが同じ 2 つのプロジェクトの更新が異なる場合、更新プロパティを比較できます。これは、手作業で更新プロパティを設定している開発者にとって便利です。開発者は、更新プロパティのフォルダが、テンプレートをベースにしたものと異なるかを調べることができます。

  Rational Synergy ヘルプのプロジェクトの更新プロパティの変更を参照してください。
- 2 つのフォルダ テンプレートまたは 2 つのフォルダを比較する。

  フォルダ テンプレートまたはフォルダを比較してクエリが正しいことを確認するか、2 つのフォルダ間で異なるメンバーを確認します。

  Rational Synergy ヘルプの 2 つの類似オブジェクトの比較を参照してください。

更新プロパティの検証

冗長更新を実行し、メッセージには問題が見つからないのに、プロジェクトが正しく構成されていない場合、以下のリストを使用して、更新プロパティがソフトウェア リリースに正しく設定されているか確認します。

1. 適切な**プロジェクト グルーピング プロパティ** ダイアログボックスを開きます。
   - リリースが正しく設定されているか確認する。
   - 目的が正しいことを確認する。
   - プロセス ルールが正しく設定されているか確認する。
   - 正しい[ベースライン プロジェクト](#)が設定されているか確認する (ベースラインに更新するプロジェクトのバージョンがあることを確認してください)。

     これは大変重要です。ベースライン プロジェクトがなく、プロジェクトが最新ではない場合、プロジェクトは不完全になります。
2. プロジェクト グルーピング プロパティのタスクを確認します。期待するタスクが入っていることを確認します。
   - フォルダ テンプレートをダブルクリックして、フォルダ プロパティを確認する。**フォルダ テンプレート プロパティ** ダイアログボックスで、**フォルダ プロパティ** タブをクリックします。
   - クエリベースのフォルダの場合は、フォルダのクエリでリリースが正しく設定されていることを確認する。
3. 更新プロパティが正しい場合、フォルダ テンプレートを確認します（プロジェクト グルーピングを右クリックし、**プロセス ルール プロパティ**を選択して、**タスク** タブをクリックします)。
   - フォルダ テンプレートをダブルクリックして、フォルダ プロパティを確認する。**フォルダ テンプレート プロパティ** ダイアログボックスで、**フォルダ プロパティ** タブをクリックします。
   - クエリベースのフォルダの場合は、フォルダのクエリでリリースが正しく設定されていることを確認する。

   あるいは、[2 つの類似オブジェクトの比較](#)を参照して、フォルダ、フォルダ テンプレート、プロセス ルール、プロジェクト グルーピングに関する情報を調べることもできます。

コンフリクト検出のしくみ

コンフリクトとは、構成上問題になり得る点です。コンフリクトの検出は、必要な構成がプロジェクトに含まれているかどうかを確認する方法です。構成に変更の一部しか含まれていない場合、コンフリクトの検出によってそれが発見されます。特定の変更（タスクによって定義されたもの）を含めれば、確実にその変更がすべて含まれます。

> **注記**：コンフリクトが、問題と**なり得る点**であることに注意してください。コンフリクトがすべて悪いわけではありません。特定のコンフリクトについて通知を受けるべきかどうかは、ソフトウェア開発チームの作業の方法によって異なります。

たとえば、プロジェクトを更新した後に**コンフリクトの検出**操作を実行して、複数のタスクに関連付けられているオブジェクトが見つかれば、コンフリクトの警告を受けます。プログラムに複数ある問題を修正するためにチームで頻繁にプログラムを書き換える場合は、複数のタスクに関連付けられているオブジェクトは問題とはなりません。このようなコンフリクトの通知を無効にして、解決する必要のあるコンフリクトのみが通知されるように設定できます。

コンフリクト検出によって、構成に変更の一部が欠落していたり、予期しない変更が含まれていたりする状況を検出してください。

以下のセクションでは、このようなコンフリクトの検出方法、このような問題が発生する理由、また必要に応じて解決する方法について説明します。さらに、通知を受けたいコンフリクトのタイプを Rational Synergy で指定する方法についても説明します。

コンフリクトの発生

コンフリクトは、プロジェクトの更新プロパティとプロジェクトのメンバーに差異がある場合に発生します。Rational Synergy がコンフリクトの検出に使用する関係には、以下のものがあります。

- ひとまとまりになっている（1 つのタスクに関連付けられている）変更
- 他の変更（先行する変更）を含む変更
- 更新プロパティで、プロジェクトに含めるよう指定したタスク

  たとえば、Rational Synergy では、オブジェクトがプロジェクトのメンバーであるのにプロジェクトの更新プロパティに関連付けられているタスクがない場合、コンフリクトを検出します。逆に、オブジェクトが、プロジェクトの更新プロパティで指定されているタスクに関連付けられているのに、そのオブジェクトがプロジェクト（ディレクトリまたは他のオブジェクトの先行バージョン）にない場合にも、コンフリクトを検出します。

プロジェクトのコンフリクトを特定する直前に、プロジェクトの更新を行ってください。更新の後、あるいはメンバーを手作業で更新した後にプロジェクトの更新プロパティが変更されると、プロジェクトの更新プロパティとプロジェクトのメンバーの間に不一致が生じます。したがって、プロジェクトの更新直後にプロジェクトのコンフリクトの特定を行うと、余分なコンフリクトが発生する可能性を最小限にできます。

コンフリクトの検出

コンフリクト検出操作では、プロジェクト階層ごとのコンフリクトが検出されます。さらに、深いコンフリクトの検出を行うこともできます。この操作では、ベースライン以前またはベースライン内のタスクやオブジェクトなどプロジェクト メンバーを考慮した分析を行います（Rational Synergy ヘルプの深いコンフリクト検出の実行を参照してください）。

- コンフリクトを検出したいプロジェクトを右クリックし、**メンバーシップ コンフリクトの検出**を選択して、**プロジェクトとサブプロジェクト**を選択します。

  Rational Synergy でプロジェクトを分析している間、進捗状況が表示されます。分析が完了したら、**メンバーシップ コンフリクト** ダイアログボックスにプロジェクトのコンフリクトが表示されます。コンフリクトが見つからなかった場合は、メイン ウィンドウのステータス バーにコンフリクトが検出されなかったことを示すメッセージが表示されます。

  これで、解決すべきコンフリクトがわかります。コンフリクトと依存関係について不明点がある場合は、後続の 2 つのセクションコンフリクトのカテゴリとコンフリクトと依存関係を参照してください。コンフリクトと依存関係についてすでに理解できていれば、プロジェクトのコンフリクトを解決できます。コンフリクトの解決を参照してください。

  メンバーシップ コンフリクト ダイアログボックスのオプションによって、コンフリクトを調べることができます。オプションの詳細については、Rational Synergy ヘルプのプロジェクトまたはプロジェクト グルーピング内のメンバーシップ コンフリクトの解決を参照してください。

コンフリクトのカテゴリ

コンフリクトには 2 つの主要カテゴリがあります。以下にカテゴリを示します。

- プロジェクトにあるが更新プロパティにない変更

  たとえば、新しいオブジェクト バージョンを、そのタスクを更新プロパティに追加せずに使用すると、オブジェクトにはコンフリクトが生じます。

- 更新プロパティにあるがプロジェクトにない変更

  たとえば、プロジェクトの更新プロパティに、同一オブジェクトのパラレル バージョンに関連付けられている 2 つのタスクがある場合、プロジェクトのメンバーでないバージョンにはコンフリクトが生じます。

  タスクと選択したオブジェクト バージョンの間に関連付けがあるため、Rational Synergy はタスクまたは個別オブジェクトとしてコンフリクトを示します。

コンフリクトと依存関係

Rational Synergy では、オブジェクト バージョンは独立したものではありません。オブジェクト バージョンには、先行バージョンの変更がすべて含まれます。後継バージョンは、先行バージョンから逐次チェックアウトされ、その先行バージョンの内容を基にしたものです。

依存関係は、コンフリクトを理解するための**重要概念**です。依存関係について見てみましょう。**bar.c-1** がタスク 12 と、**bar.c-2** がタスク 25 と、**bar.c-3** がタスク 37 と、**bar.c-4** がタスク 48 と関連付けられているとします。この場合、**bar.c-4** にはタスク 48 からの変更だけでなく、タスク 37、25、12 からの変更も含まれることになります。

次に、依存関係がプロジェクトの構造にどのような影響を与えるかを見ていきます。プロジェクトに **bar.c-3** が含まれている場合、タスク 37 も含まれているでしょうか。含まれています。ただし、**bar.c-3** には先行バージョンの変更も含まれているため、タスク 25、12 も含まれます。プロジェクトの更新プロパティにタスク 37 があり、タスク 25 がない場合はどうなるでしょうか。これは、コンフリクトの定義「プロジェクトにあるが更新プロパティにない変更」に当てはまります。

ここまでの **bar.c** の例は一次元的なものです。各タスクが他のオブジェクト バージョンにも関連付けられることがあることを考慮すると、依存関係はもっと複雑になります。たとえば、タスク 37（**bar.c-3** と関連付けられた）が **foo.c-6** とも関連付けられているとします。**bar.c-3** またはその後継バージョンの 1 つがプロジェクトに含まれていれば、**foo.c-6** またはその後継バージョンの 1 つもプロジェクトに含まれています。さらに、**foo.c-6** の先行バージョンに関連付けられているタスクがプロジェクトに含まれ、したがって他の関連オブジェクトも含まれることになります。Rational Synergy では、履歴とタスク関係をすべて分析し、依存関係に基づいて、どの変更が含まれているか、どの変更が含まれるべきかを判断します。

プロジェクトは、ベースライン プロジェクトと呼ばれる別のプロジェクトに基づくものです。ベースライン プロジェクトには、メンバー オブジェクトの先行バージョンの変更がすべて含まれています。Rational Synergy では、現行プロジェクトとベースライン プロジェクトの間の差異のみを対象としてコンフリクトを検出します。したがって、コンフリクト分析で調べる各プロジェクト メンバーは、ベースライン プロジェクトにあるバージョンまでのものです。

コンフリクトの用語

Rational Synergy によって示されるプロジェクトのコンフリクトのタイプにはそれぞれ名前があります。ここでは、コンフリクトの各タイプの用語について説明します。

プロジェクトに直接含まれる変更を、明示的変更と呼びます。間接的に含まれる変更を、黙示的変更と呼びます。

詳しく説明しましょう。更新プロパティにある変更は、プロジェクトに含まれるよう指定されたタスクであるため、明示的に指定されたものです。更新プロパティで明示的に指定されていなくても他の変更が依存しているか、含んでいるために必要な変更は、黙示的に必要なものです。

ソース コードがプロジェクトにある変更は、含まれている変更です。含まれている変更が明示的に指定されていない場合、黙示的に含まれています。黙示的に含まれている変更は、プロジェクトの他の変更がその変更に依存しているか、その変更を含んでいるため、プロジェクトに含まれています。

コンフリクト メッセージの定義

コンフリクト メッセージは、コンフリクト検出実行時に表示されるメッセージです。デフォルトで表示されるメッセージと、表示されないメッセージがあります。これらのメッセージについては、Rational Synergy ヘルプのプロジェクトまたはプロジェクト グルーピング内のメンバーシップ コンフリクトの解決を参照してください。

チームで別のデフォルト表示設定が必要な場合は、ロールが *ccm_admin* のユーザーが設定を変更できます。詳細については、Rational Synergy CLI ヘルプの conflict_parameters を参照してください。

コンフリクトの解決

解決すべきコンフリクトが特定できたら、以下の事項を考慮して解決してください。

- 更新を実行してプロジェクト　メンバーを更新プロパティと同期させます。オブジェクト バージョンが選択された理由がわからない場合は、冗長更新を実行します。
- コンフリクトに関する情報を収集します。

  — プロジェクトを調べ、オブジェクトのどのバージョンが使用されているか確認します。

  — オブジェクトの履歴を調べ、コンフリクトが発生しているオブジェクトと、使用されているオブジェクトの関係を確認します。

  — 対象とするオブジェクト バージョンと関連付けられているタスクがどれか確認します。
- 黙示的に含まれているオブジェクトや要求されているオブジェクトについて、以下を調べます。

  — タスクをプロジェクトの更新プロパティに追加すべきか検討します。追加する必要がある場合は、なぜ含まれていないのか調べます。タスクのリリース値が誤っていないか確認します。

  — オブジェクトのタスクをプロジェクトの更新プロパティに追加すべきではない場合、後継バージョンを調べ、そのタスクをオブジェクトの更新プロパティから削除する必要があるか検討します。削除する必要がある場合は、なぜ含まれたのか調べます。リリース値の設定が誤っていないか確認します。
- プロジェクトの更新プロパティまたはタスク リリース値を更新した場合は、必ず再度更新してください。
- 各パラレル　バージョンについて、マージする必要があるか検討します (マージの必要がある場合は、新しいタスクを作成し、割り当てます)。
- 可能なかぎりコンフリクトを解決したら、コンフリクト検出を再度実行します。依存関係により 1 つのコンフリクトが連鎖的に影響を及ぼすため、コンフリクトを 1 つ解決すると他にも多くのコンフリクトが解決されることはよくあります。

  変更を追跡するプロセスと方法がチームによって異なる点に注意してください。何をコンフリクトと見なすかは、チームによって異なります。特定のコンフリクトを方法論の一部と考えるチームでは、そのコンフリクトはコンフリクトと見なさないようにする場合があります。同じコンフリクトが、別のチームでは、直ちに修正する必要のある問題と見なされることもあります。どのようなものを開発プロセスのできるかぎり早期に対処するコンフリクトとするかについては、チームで合意しておくことが必要です。

5 ベースラインの機能

ベースラインの作業

ベースラインは、特定の時点におけるユーザーのデータを表現するために使われるプロジェクトとタスクのセットです。ベースラインにはさまざまな用途があります。更新を行うとき、Rational Synergy は新規変更を探す開始点としてベースラインを使用します。また、2 つのベースラインを比較して、特定のビルドを基準にどのような変更が行われたかを確認できます。Rational Change を使用していれば、ベースラインを使用して変更依頼レポートを作成できます。

通常はビルド マネージャがベースラインを作成します。開発者は自分のビルドを他の開発者にも使用できるようにはしないので、ベースラインを作成する必要がありません。

ビルドを行ったら直ちにベースラインを作成すると便利です。ベースラインを作成し、すべての開発者に公開することなくテスト グループに公開できます。これをテスト ベースラインと呼びます。ビルド完了後にすぐにテスト ベースラインを作成すると、Rational Synergy は、後で特定のビルドに対して修正コードを作成する際に必要となるビルド情報を保存します。

統合テストと**システム テスト**ごとにビルドを作成しておくと、テスターおよび開発者はそのビルドに盛り込まれた一連の変更点を参照できます。通常は、同一のリリースおよび同一の目的を持つすべてのプロジェクトに対して 1 つのベースラインを作成します。たとえば、各**統合テスト** ビルド用には、そのリリースのすべての**統合テスト** プロジェクトを使用してベースラインを作成します。

ベースラインの存在は、更新操作のパフォーマンスを向上させます。ベースラインを使用した更新操作は、そのリリース全体のタスクではなく、最新のベースライン以降に追加されたタスクだけを分析すれば済むからです。

ベースラインの使用方法

ベースラインを作成する場合、ベースラインに入れるプロジェクトのリストを選択します。変更を参照するための完全なセットとなるように、必ずベースラインに関連する**すべて**のプロジェクトを含めてください。

静的状態のプロジェクトなら、特に何も変更せず、どれでも入れることができます。静的ではないプロジェクト（ビルド管理プロジェクト）を入れた場合、ベースラインを作成すると以下のようなことが起こります。

- オブジェクトの新バージョンがコピーされる。このバージョンにはワークエリアがないので、操作がとても速くなります。
- 階層内で複数のプロジェクトをコピーすると、もとの階層のコピーである単一の階層で使用される。
- コピーされたビルド管理プロジェクトより**前に**挿入されたベースラインを示すため、プロジェクトの履歴が更新される。
- 新しいプロジェクトが静的状態にチェックインされる。

もとのビルド管理プロジェクトとそのワークエリアは変更されません。この利点は、これらが継続的に増分再ビルドされることです。ビルド管理プロジェクトをチェックインして新しい prep（準備）バージョンをそこからチェックアウトすると、完全に再ビルドされます。これは、管理されていない中間製品が、新たにチェックアウトされたプロジェクトのワークエリアに入っていないからです。

ベースラインの作成後、ベースライン プロジェクトの選択されたプロジェクトに対してワークエリア メンテナンスを有効にして、他のユーザーにもワークエリアを使用可能にできます。これはワークエリアに書き出されます。通常、開発者が独自のバージョンをチェックアウトせずに静的サブプロジェクトを再利用する場合、絶対サブプロジェクトに対してこれを行います。以下のコマンドは、データベースの検索を行い、ワークエリアを有効にする例です。

```
ccm query "is_project_in_baseline_of
(baseline('20070203')) and name match '*_ext_x' and
platform='NIX'"
ccm wa -wa @
```

ベースラインの使用方法の検討

Rational Synergy は、ベースラインをある時点におけるプロジェクトおよびタスクのスナップショットとして使用します。ベースラインを作成する前に、それをどのように使用するかを考える必要があります。更新操作は、「ここから開始する」という意味でベースラインを使用します。したがって、標準のプロセス ルールを使用して、ベースラインに複数のコンポーネントのプロジェクトを入れたり、リリースのすべてのプロジェクトを入れなかったりした場合、Rational Synergy はベースラインを使用して適切に更新できなくなります。この方法論はサポートされていますが、使用するためにはプロセス ルールをカスタマイズする必要があります。

以下の 2 つの例はベースラインを正しく設定する方法を示し、最後の例は誤って設定されたベースラインを示します。ベースラインを正しく使用すれば、プロジェクトの更新を効率よく行うことができます。

ベースラインは、そのリリースによって選択されます。したがって、ベースラインには一貫したリリースのプロジェクトを含むことが重要です。ベースライン内のプロジェクトは、プロジェクト グルーピング内のプロジェクトのベースライン プロジェクトとして使用されます。下表では、プロジェクト グルーピング内のすべてのプロジェクトがベースライン内のプロジェクトに対応していることに注意してください。これは、完全なベースラインの例です。

CM/7.0 ビルド 1234 のベースライン	**< ベースライン**	**CM/7.1 統合テストのプロジェクト グルーピング**
cm_top-CM/7.0	← ベースライン プロジェクト	cm_top-CM/7.1
cm_gui-CM/7.0	← ベースライン プロジェクト	cm_gui-CM/7.1
cm_api-CM/7.0	← ベースライン プロジェクト	cm_api-CM/7.1
cm_platform-CM/7.0	← ベースライン プロジェクト	cm_platform-CM/7.1

下表では、複合コンポーネントの階層に 2 つの個別のベースラインが必要なことに注意してください。これは、正しいベースラインの例です。

CM/7.0 ビルド 1234 のベースライン	**< ベースライン**	**CM/7.1 統合テストのプロジェクト グルーピング**
cm_top-CM/7.0	← ベースライン プロジェクト	cm_top-CM/7.1
cm_gui-CM/7.0	← ベースライン プロジェクト	cm_gui-CM/7.1
cm_api-CM/7.0	← ベースライン プロジェクト	cm_api-CM/7.1
cm_platform-CM/7.0	← ベースライン プロジェクト	cm_platform-CM/7.1
TC/5.1 ビルド 5678 のベースライン	**< ベースライン**	**TC 5.2 統合テストのプロジェクト グルーピング**
change_api-TC/5.1	← ベースライン プロジェクト	change_api-TC/5.2

複合コンポーネントのプロジェクトに 1 つのベースラインを作成すると、標準のプロセス ルールを使用している場合、プロジェクトは正しくベースライン プロジェクトを見つけることができないことがあります。以下の例では、TC/5.1 リリース（表中イタリック）からのサブプロジェクトは CM/7.0 リリースのベースラインに含まれています。

Rational Synergy は、このプロジェクトをベースライン プロジェクトとして使用できません。なぜなら、このプロジェクトがメンバーとなっているベースラインが（たとえば **CM/7.0 ビルド 1234 のベースライン**）、これをベースライン プロジェクトとして使用するどのプロジェクト グルーピング（たとえば、**TC 5.2 統合テストのプロジェクト グルーピング**）からも選択されないからです。また、両方のリリースのタスクがベースラインに含まれることになります。

ベースライン内のすべてのタスクがベースライン内のすべてのプロジェクトから使用されない場合、それらのタスクは更新時にプロジェクト グルーピングに差し引かれることはありません。理想的には、更新操作で選択される新しいプロジェクトメンバーは以下のように集められます。

- Rational Synergy が、プロセス ルールで指定されたすべてのタスクを集める。
- Rational Synergy が、ベースラインからすべてのタスクを差し引く。

  ベースライン内の 1 つまたは複数のプロジェクトが使用していないタスクがベースライン内にある場合、Rational Synergy はこれを検出して、更新に使用するタスクを計算するときにこれを差し引きません。

  下表では、最初の行の **change_api-TC/5.1** は別のコンポーネントのものであり、CM/7.0 ベースラインに入れるべきではありません。この行があるため、それに関連するすべてのタスクもベースラインの一部となり、CM/7.0 または TC/5.1 のタスクはベースライン内のすべてのプロジェクトによって使用されないため、どのタスクも差し引かれません。これはパフォーマンスに悪影響を及ぼします。
- 更新は、ベースライン プロジェクト内の対応するオブジェクトとプロジェクト グルーピング プロパティ内のタスクをもとに、プロジェクトの各メンバーの候補を選択します。対象となる候補は、ベースライン プロジェクト内のオブジェクトと上記のように集められたタスクに関連するそのオブジェクトの任意の新バージョンです。

  下表で最初の列が**欠落**しているのは、**TC 5.2 統合テストのプロジェクト グルーピング**のベースラインが見つからないことを意味します。ベースラインがないため、**change_api-TC/5.1** のベースライン プロジェクトはありません。これは、プロジェクトの更新時に得られると期待していたメンバーを得られないことを意味します。

これは、標準のプロセス ルールでは使用できないベースラインの例です。

CM/7.0 ビルド 1234 のベースライン	**< ベースライン**	**CM/7.1 のプロジェクト グルーピング**
cm_top-CM/7.0	← ベースライン プロジェクト	cm_top-CM/7.1
cm_gui-CM/7.0	← ベースライン プロジェクト	cm_gui-CM/7.1
cm_api-CM/7.0	← ベースライン プロジェクト	cm_api-CM/7.1
cm_platform-CM/7.0	← ベースライン プロジェクト	cm_platform-CM/7.1
change_api-TC/5.1		
欠落	**< ベースライン**	**TC 5.2 統合テストのプロジェクト グルーピング**
なし	← ベースライン プロジェクト	change_api-TC/5.2

ベースラインの作成

この時点で、通常、統合テスト プロジェクトまたはシステム テスト プロジェクトを更新し、コンフリクト処理および製品のビルドが完了しました。これで、ベースラインを作成し、今後の参考用にこのビルドのコピーを保存できます。ベースラインの作成時に、ベースラインを公開して開発者が変更を利用できるようにするか、さらにビルドのテストを行うまで公開しないかを選択できます。

1. プロジェクト階層またはプロジェクト グルーピングを右クリックして**ベースラインの作成**を選択します。

   ビルド管理プロジェクト グルーピング、または静的またはビルド管理プロジェクトであるプロジェクトから、ベースラインを作成できます。

   ベースラインの作成ダイアログボックスが表示されます。

   注記：ベースラインを作成するとき、ベースラインに含めるプロジェクトのリストを選択します。変更を参照するための完全なセットとなるように、必ずベースラインに関連する**すべて**のプロジェクトを含めてください。

2. 必要に応じてベースラインのプロパティを修正します。

   a. **名前** フィールドに名前を入力します。

      ベースラインの名前を指定します。これは、このデータベース内でベースラインを一意に識別します。デフォルトで、Rational Synergy は、作成日を使用してベースラインに名前を付けます。たとえば、20090309 は、2009 年 3 月 9 日を意味します。ただし、これは変更できます。**/ ¥ ' " : ? * [ ] @ - #** は、禁止されており、名前には使用できません。

      複数のデータベースにベースラインを作成し、Rational Change を使用してこれらのビルドのレポートを作成する場合、各データベースで同じベースライン名を使用します。これにより、複数のデータベースで関連するベースラインを持つビルド レポートを作成できます。

   b. リリースが正しいことを確認します。

      リリースは、特定のリリース固有のベースラインを識別するプロパティです。

   c. 目的が正しいことを確認します。

      プロジェクトの目的は、たとえば、**統合テスト**など、それが何に使われるかを定義します。プロセス ルールはベースラインの目的を使用して、更新操作時に適切なベースラインを選択します。

d. **ビルド** フィールドに入力して、ビルドの識別子を設定できます（オプション）。

ビルド プロパティにはベースラインに関連するビルドの識別子（文字、数字、またはその組み合わせ）が表示されます。ビルド識別子は、最大 64 文字まで含むことができます。

複数のデータベースにベースラインを作成し、Rational Change を使用してこれらのビルドのレポートを作成する場合、各データベースで同じベースライン ビルド識別子を使用します。これにより、複数のデータベースで関連するベースラインを持つビルド レポートを作成できます。

e. **詳細**フィールドに、作成するベースラインの説明を入力します。

3. ベースラインに含まれるプロジェクトを変更します。

オプションダイアログボックスで**ベースラインの作成**アクションとして**リリースにかかわらず全てのサブプロジェクトを含む**オプションを選択した場合、リリース値に関係なくプロジェクト階層のすべてのプロジェクトを利用できます。このオプションを選択しなかった場合は、すべての *prep* サブプロジェクトが使用されます。最上位プロジェクトのコンポーネントと一致するコンポーネントを含む静的サブプロジェクトのみが使用されます。

このオプションの設定の詳細については、Rational Synergy ヘルプの[ベースライン作成オプションの変更]を参照してください。

a. 個々のプロジェクトを追加するには、**プロジェクトの追加**ボタンをクリックします。

プロジェクトの選択ダイアログボックスが表示されます。これは、**クエリ**ダイアログボックスと同じ働きをします。デフォルトで、プロジェクトのリリースと目的（たとえば、**toolkit/2.0** や**システム テスト**）をベースに、選択セット フィールドにプロジェクトが表示されます。さらに、クエリを定義して、ベースラインに追加するプロジェクトを探すことも可能です。

b. あるプロジェクト　グルーピングからすべてのプロジェクトを追加するには**プロジェクト グルーピングの追加**ボタンをクリックします。

プロジェクト グルーピング選択ダイアログボックスが表示されます。これは、**クエリ** ダイアログボックスと同じ働きをします。プロジェクト グルーピングを追加すると、プロジェクト グルーピングに関連するプロジェクトを追加できます。これは、[増分ベースライン]を作成する場合に便利です。さらに、クエリを定義して、ベースラインに追加するプロジェクト グルーピングを探すことも可能です。

c. 既存のベースラインからすべてのプロジェクトを追加するには**ベースラインの追加**ボタンをクリックします。

ベースライン選択ダイアログボックスが表示されます。これは、**クエリ** ダイアログボックスと同じ働きをします。ベースラインを追加する

と、ベースラインに関連するプロジェクトを追加できます。これは、増分ベースラインを作成する場合に便利です。さらに、クエリを定義して、追加するベースラインを探すことも可能です。

d. **OK** をクリックします。

4. ベースラインを作成したら開発者に公開します。

   使用者を限定してテスト ベースラインを作成している場合は、今手順を飛ばしてステップ 5 に進んでください。

   このオプションはデフォルトで選択解除されていますが、設定が必要な場合もあります。ベースラインを公開すると、開発者はすぐにプロジェクトを更新して、ベースラインから最新の変更を取り込むことができるようになります。

5. **OK** をクリックしてベースラインを作成します。

開発者にテスト ベースラインを公開

システム テスト プロジェクトを更新し、コンフリクト処理、製品のビルド、アプリケーションのテストが完了しました。次に開発者が変更を使用できるようにします。このためには、テスト ベースラインを公開します。

ベースラインは、公開されるとすぐ、更新時に選択されるベースラインとして使用可能となります。プロセス ルールを使用すると、プロジェクトに必ず最新のベースラインが使用されます。

1. システム テストに変遷したいテスト ベースラインを選択し、ベースラインを右クリックして**公開**を選択します。
2. 必要に応じて、自分のベースライン プロジェクト用のワークエリアを作成します。

   たとえば、外部プロジェクトを製品共有用に作成する場合、必ず全員から見えるワークエリアを作成してください。ベースライン プロジェクトの詳細については、ベースラインの使用方法の最後のパラグラフを参照してください。

   a. ワークエリア設定を修正するシステム テスト プロジェクトを右クリックし、プロパティを選択します。

      プロパティ ダイアログボックスが表示されます。

   b. **ワークエリア** タブをクリックします。

   c. **On/Off** オプションを選択します。

      このオプションを選択すると、**適用**ボタンをクリックした時に、プロジェクトのコピーがワークエリアに保持され、プロジェクトが自動的に同期されます。

   d. **適用**をクリックして変更を保存します。
3. 開発者に、プロジェクトを更新して変更を取り込むことができることを知らせます。

これで、システム テスト プロジェクトのベースラインを公開しました。ベースラインの公開により、他のユーザーがビルドを利用できるようになります。開発者は自分のプロジェクトを更新すれば、すぐに新しいベースラインを使用できるようになります。

ベースラインと更新プロセス

プロセス ルールに従ってプロジェクトを更新する際、使用すべき[ベースライン プロジェクト]を再評価します。プロセス ルールを調べてベースラインを確認し、次にベースラインを調べてプロジェクトのバージョンを特定します。ベースラインにプロジェクトの複数バージョンがあるときは、プラットフォーム値を比較して一致または互換するプラットフォーム値のあるプロジェクトを選択します。

たとえば、**editor-bob** プロジェクトが更新された場合、プロセス ルールを調べて最新のベースラインが選択されたことを確認します。次に、テンプレートの候補と一致する最新のベースライン（例 : **20070115**）を特定します。さらに、ベースライン **20070115** の **editor** プロジェクトのバージョンを調べ、ベースライン プロジェクトとして使用します。

ベースラインのプロジェクトのバージョンが見つからない場合、そのプロジェクトはベースライン プロジェクトなしで更新されています。これは、現行リリースのタスクと関連付けられていないオブジェクト バージョンが候補と見なされないため、プロジェクトを更新したときに選択されていないことを意味します。この場合、いくつかのディレクトリ エントリが空のままになります。

このように、前回のベースラインから変更されていないプロジェクトも含め、アプリケーションのすべてのプロジェクトをベースラインに入れることはとても重要です （静的プロジェクトは複数のベースラインで再利用できます)。

増分ベースラインの作成

ビルド管理用のベースラインのプロジェクトをコピーすることが不可能または現実的ではない場合、増分ベースラインという方法を使用できます。

たとえば、**proj1** から **proj100** までのプロジェクトがあり、**proj1-int** と **proj2-int** で新しいベースラインを作成したい場合（**proj3** から **proj100** までに変更がない場合など）、最新のベースラインとこれら 2 つのプロジェクトから新しいベースラインを作ることができます。

1. 増分ベースラインに含めたい 2 つのプロジェクトを選択して右クリックし、**ベースラインの作成**を選択します。

   ビルド管理プロジェクト グルーピング、または静的またはビルド管理プロジェクトであるプロジェクトから、ベースラインを作成できます。

   ベースラインの作成ダイアログボックスが表示されます。

2. 必要に応じてベースラインのプロパティを修正します。

   a. **名前**フィールドに名前を入力します。

      ベースラインの名前を指定します。これは、このデータベース内でベースラインを一意に識別します。デフォルトで、Rational Synergy は、作成日を使用してベースラインに名前を付けます。たとえば、20070309 は、2007 年 3 月 9 日を意味します。ただし、これは変更できます。**/ ¥ ' " : ? * [ ] @ - #** は、禁止されており、名前には使用できません。

      複数のデータベースにベースラインを作成し、Rational Change を使用してこれらのビルドのレポートを作成する場合、各データベースで同じベースライン名を使用します。これにより、複数のデータベースで関連するベースラインを持つビルド レポートを作成できます。

   b. リリースが正しいことを確認します。

      リリースは、特定のリリース固有のベースラインを識別するプロパティです。

   c. 目的が正しいことを確認します。

      プロジェクトの目的は、たとえば、**統合テスト**など、それが何に使われるかを定義します。ベースラインの目的を変更すると、Rational Synergy はプロジェクトまたはプロジェクト グルーピングの更新時に異なる選択基準を使用します。

   d. **ビルド** フィールドに入力して、ビルドの識別子を設定できます（オプション）。

      ビルド プロパティは、ベースラインに関連するビルドの識別子（文字、数字、またはその組み合わせ）を示します。ビルド識別子は、最大 64 文字まで含むことができます。

      複数のデータベースにベースラインを作成し、Rational Change を使用してこれらのビルドのレポートを作成する場合、各データベースで

同じベースライン ビルド識別子を使用します。これにより、複数のデータベースで関連するベースラインを持つビルド レポートを作成できます。

e. **詳細**フィールドに、作成するベースラインの説明を入力します。

3. ベースラインに含まれるプロジェクトを変更します。

   ベースラインの追加ボタンをクリックして、最新のベースラインを追加します。

   いま追加したビルド管理プロジェクトの旧バージョンを削除する必要はありませんが、削除すればベースラインのメンバーのより正確な情報がわかります。

   ベースライン選択ダイアログボックスが表示されます。これは、**クエリ**ダイアログボックスと同じ働きをします。デフォルトで、**含まれるプロジェクト** フィールドには、プロジェクトのリリースと目的をベースにプロジェクトが表示されます。たとえば、**toolkit/2.0** と**システム テスト**と表示されます。さらに、クエリを定義して、ベースラインに追加するプロジェクトを探すことも可能です。

4. ベースラインを作成したら開発者に公開します。

   このオプションはデフォルトで選択解除されていますが、設定が必要になります。ベースラインを作成すると、開発者はすぐにプロジェクトを更新して統合テストを通過した最新の変更を取り込むことができるようになります。

5. **OK** をクリックしてベースラインを作成します。

不要なベースラインの削除

不要なベースラインを削除するには、まず削除のマーク付けをおこないます。マーク付けをしておくと、オフ ライン保存と削除コマンドを設定して、削除のマークが付けられたベースラインが使われなくなったときに自動的に削除できます。

以下に、ベースラインに削除のマーク付けを行う手順を説明します（Rational Synergy CLI ヘルプで説明している soad コマンド を使用すると、削除とマーク付けたベースラインを自分で、または CM アドミニストレータが削除できます）。

1. 削除したいベースラインを検索します。

   検索 > **ベースライン**

   クエリ ダイアログボックスが表示されます。

2. クエリ条件を設定してクエリを実行します。

   a. **リリース**を適切なリリースに設定します。

   b. プラス記号をクリックして、クエリに別のプロパティを追加します。

   c. **目的**を**統合テスト**に設定します。

   d.  をクリックしてクエリを実行します。

3. 不要なベースラインに削除マークを付けます。

   削除するベースラインすべてを選択し、右クリックして**削除**を選択します。

   削除ダイアログボックスに選択したベースラインが表示されます。**削除**をクリックします。選択したベースラインに削除のマークが付けられます。

   更新操作では、削除のマークが付けられたベースラインは選択されません。チームでプロジェクト グルーピングを使用し、プロジェクト グルーピングのいずれかでマーク付けされたベースラインを使用している場合、更新操作中は異なるベースラインが選択されます。

   ビルド マネージャはオフライン保存と削除コマンドを設定して、削除のマークが付けられたベースラインが使われなくなったときにそのベースラインを自動的に削除できます。詳細については、Rational Synergy CLI ヘルプの soad コマンドを参照してください。

   さらに、この時点で、完了したリリースの不活性化を決定する場合があります。また、Rational Synergy CLI ヘルプで説明している ccm clean_up コマンドを使用して、そのリリースのプロセス ルールと旧リ リースを整理できます。

   余分なリリースやプロセス ルールを保持するのに必要なオーバーヘッドはそれほどありません。ただし、リリースやプロセス ルールを整理することにより、対応するダイアログボックスで必要な情報が探しやすくなります。

また、プロジェクト グルーピングがベースラインを使用している場合、またはプロセス ルールがベースラインを使用している場合、そのベースラインは削除できません。関連付けられている 1 つまたは複数のプロジェクトまたは製品がベースラインに含まれないプロジェクトのメンバーであった場合、ベースラインおよびそのチェックイン済みプロジェクトと製品を削除しようとすると、ベースラインは削除されますが、プロジェクトまたは製品は削除されません。また、ベースラインに含まれる 1 つまたは複数のプロジェクトが別のベースラインのメンバーでもある場合、ベースラインは削除されますが、プロジェクトは削除されません。

古いベースラインを削除するためには、古くなったプロジェクト階層を削除し、プロジェクト グルーピングを空にする必要があります。しかし、空のプロジェクト グルーピングには、所有者が明示的にそのグルーピングに追加または削除したタスクの重要な情報は保持されている場合があります。

明示的に追加または削除されたタスクのない空のプロジェクト グルーピングを調べて削除するには、以下のコマンドを使用します。

```
ccm set role ccm_admin

ccm query -t project_grouping
"is_no_project_grouping() and
has_no_added_task_in_pg() and
has_no_removed_task_in_pg()"

ccm delete @
```

6 製品の共有

製品の共有とは、ビルド マネージャによってビルドされた製品の共有を開発者に認めることです。たとえば、Joe が **toolkit.exe** という実行形式ファイルについて作業しているとします。Joe は **toolkit.exe** の変更を担当しているため、その *working*（作業）バージョンを持っています。隣のオフィスでは、Mary が **guilib.lib** ライブラリを変更しています。**toolkit.exe** 実行形式ファイルは **guilib.lib** ライブラリと関連していますが、Joe は **guilib.lib** の変更は行っていないので、working（作業）バージョンは必要ありません。

Joe は、ビルド マネージャがビルドした **guilib.lib** のバージョンを使用できます。統合テスト プロジェクトがビルドされ、テストを通過した後、ビルド マネージャはベースラインを作成し、それにより *prep*（準備）製品をチェックインして、開発者が使用できるようにします。Mary の **guilib.lib** の最新の変更を使用する準備ができたら、Joe は自分の開発プロジェクトを更新して、最新のテスト済み製品ファイルを取り込みます。

これを使用する利点は何でしょうか。

このプロセスにより、自分が変更しない製品を開発者がビルドする必要がなくなるので、開発者が完了しなければならない余分の作業が削減されます。

製品の共有は、共有可能な外部プロジェクト（外部プロジェクトの共有）へのパッケージングによって可能になります。

外部プロジェクトの共有

以下では、外部プロジェクトを共有する 1 つの方法について説明します。外部プロジェクトを共有する様々な方法がありますが、以下では Rational Synergy 開発者が使用している方法を示します。

外部プロジェクトにより、自分が変更しないプロジェクトを開発者がコピーしたり更新したりする必要がなくなるので、開発者は時間を節約できます。外部プロジェクトは、ソフトウェア開発の最良の慣行とされているモジュール式コードと情報隠蔽の推進にもなります。

しかし、ビルド マネージャは、プロジェクトの再構築、makefile の更新、ビルド プロセスを自動化するビルド スクリプトの更新など、外部プロジェクト管理のために余分な作業が必要となります。また、下位プロジェクト（ライブラリなど）と、それに依存するプロジェクト（実行形式ファイルなど）の両方の作業を行う開発者は、外部プロジェクトと両方のソース オブジェクトの自分の *working*（作業）バージョンを管理する必要があります。

外部プロジェクトを使用する場合は、以下のセクションを参照してください。

- 外部プロジェクトの使用準備
- 外部プロジェクトの作成
- 段階的ビルドのためのビルド プロセス

外部プロジェクトの使用準備

外部プロジェクトでは、1 つのプロジェクトで開発された製品と関連オブジェクト（たとえばライブラリとその機能のヘッダー ファイル）を別のプロジェクトで使用できるようにします。通常、ヘッダー ファイルを取得できるように、プロジェクトをサブプロジェクトとして追加します。外部プロジェクトは、開発者が必要とする開発プロジェクトの数とサイズを最小限にする、プロジェクト構造設定方法の 1 つです。

たとえば、Joe の **toolkit.exe** 実行形式ファイルの *working*（作業）バージョンが **guilib.lib** ライブラリと関連しているとします。Joe は **guilib.lib** の変更はしていないので、その working（作業）バージョンは必要ありません。しかし自分の実行形式ファイルがこのライブラリと関連しているので、作業にはライブラリが必要です。Joe は **guilib.lib** の最新の変更を使用する際、**guilib** プロジェクトが自分の開発プロジェクトのメンバーになっていることを確認し、更新して、新たにビルドされた製品ファイルを取り込みます。

この例に基づき、Joe が 10 か 15 の異なるプロジェクトの製品を使用しており、そのいずれも変更の必要がない場合を考えます。Joe は各プロジェクトの開発プロジェクトを必要とし、したがって自分の必要なオブジェクトを取り込むために更新する際、かなりの時間を要することが考えられます。外部プロジェクトにより、この状況が改善できます。

外部プロジェクトには、製品と、その製品を使用するために必要なヘッダー ファイル**のみ**が含まれます（「外部プロジェクト」と呼ばれるのは、プロジェクトが、これらのファイルを他のプロジェクトで使用できるようにするものではあっても、実際にファイルが開発されたプロジェクトから見ると外部であるからです）。

外部プロジェクトは他のプロジェクトに対応するものであるため、類似の名前を付けると便利です。たとえば、**guilib** プロジェクトに対応する外部プロジェクトには **guilib_ext** のような名前を付けるとよいでしょう （また、makefile を変更して元のプロジェクトの代わりに外部プロジェクトが参照されるようにするため、同じ構造を維持しておくと便利です）。サブプロジェクトを追加する必要がないケースもあります。たとえば、Java では、ヘッダー ファイルが必要ないので、ライブラリを単独で追加できます。

これで、外部プロジェクトと製品が統合テストを通過したら、ビルド マネージャはチェックインのためのベースラインを作成できます。製品を参照先とするコードを持つ開発者は外部プロジェクトを共有できます。開発者には、自分が変更しなくてもよいプロジェクトの working（作業）バージョンは必要ありません。

開発者は、更新時、ビルド マネージャが最近チェックインした製品を共有します。サイトでベースラインを使用していない場合は、製品と外部タスクが統合テストを通過したあとでチェックインする必要があります。外部プロジェクトを使用している場合は、外部ベースライン プロジェクト用に、全員に見えるワークエリアを作成する必要があります。

注記：外部プロジェクトの使用は任意です。チームの製品共有ニーズを検討してください。

外部プロジェクトの作成

すでにビルド管理プロジェクト階層があることを確認します。統合テスト プロジェクトの作成については、統合テスト プロジェクトの作成を参照してください。システム テスト プロジェクトの作成については、システム テスト プロジェクトの作成を参照してください。

1. 外部プロジェクトを作成するタスクを作成し、自分を担当者にします。
   作成したタスクがカレント タスクに設定されます。
2. プロジェクトを外部プロジェクトとしてコピーします。
   a. 新しいプロジェクトを作成します。
   b. 新しいプロジェクトのプロパティを設定します。
   c. **OK** をクリックします。
3. ドラッグ アンド ドロップまたはコピーと貼り付けにより、製品オブジェクト（および他のオブジェクト）を外部プロジェクトに追加します。
4. 新しい外部プロジェクトを元のプロジェクトに追加します。
   a. **ワーク** ペインに適切なビルド管理プロジェクトを表示します。
   b. 元のプロジェクトから元のサブプロジェクトを切り取ります（既存のサブプロジェクトから外部プロジェクトを作成する場合、この手順が必要です）。
      元のプロジェクトが **guilib** プロジェクトで、**guilib_ext** 外部プロジェクトを作成した場合、**guilib** プロジェクトが使用されている箇所をすべて **guilib_ext** で置き換える必要があります。これを、外部プロジェクトごとに実行する必要があります。
      たとえば、Joe は自分の **toolkit** プロジェクトに **guilib_ext** を追加し、自分の **toolkit** プロジェクトから **guilib** プロジェクトを削除します。さらに、ビルドに必要なければ、**guilib** プロジェクトそのものも削除できます。
   c. 新しい外部プロジェクトを元のプロジェクトに追加します （新しい外部プロジェクトがサブプロジェクトの場合はこのステップはスキップできます）。
      上記ステップ 3 を参照してください。
5. 外部プロジェクトを含めるプロジェクトごとにステップ 4 を繰り返します。
   注記：新しい外部プロジェクトを適切に示すようにビルド スクリプトと makefile を修正してください。
6. カレント タスクを完了します。
7. 統合テスト プロジェクトで統合テスト サイクルを実行します。
8. プロジェクト階層からベースラインを作成します。

9. システム テスト ビルド管理プロジェクトを外部ベースライン プロジェクトからコピーします。
10. システム テスト プロジェクトでシステム テスト ビルドとテスト サイクルを実行します。

段階的ビルドのためのビルド プロセス

通常のビルド プロセスについては、ビルド管理の基本を参照してください。

外部プロジェクトがある場合、ビルド プロセスは段階的になります。外部プロジェクトを含むプロジェクト階層は、一時には完全に更新できないことがあります。これは、プロジェクト階層の一部がビルドされるまで、外部プロジェクトに取り込みたい新しい生成物が存在しないことがあり得るからです(生成物がビルド前に書き込み禁止状態にある場合、新しいバージョンがチェックアウトされます。ビルドの完了後、これらの新しいバージョンの生成物を選択して外部プロジェクトに組み込む必要があります)。まず階層の一部を更新して、段階的にビルドする必要があります。

新しいタスクを選択しないようにするには、初めての更新後にプロジェクト グルーピングの自動更新を解除し、この段階的ビルドの最後の更新後にプロジェクト グルーピングの自動更新を再度設定します。以下の例で手順を説明します。

1. プロジェクト グルーピングを最新の情報に更新して凍結します。
   a. プロジェクト グルーピングを最新の情報に更新してクエリ フォルダを更新します。

      プロジェクト グルーピングを右クリックし、**更新**をポイントし、**ベースラインとタスク**を選択します。

   b. プロジェクト グルーピングの自動更新を無効にします。

      プロジェクト グルーピングを右クリックし、**ベースラインとタスクの自動更新**の選択を解除します。
2. 外部プロジェクトで使用される生成物を生成するすべてのローレベル プロジェクト（ライブラリ プロジェクトなど）を更新します。
3. コンフリクトを特定し、解決します。
4. 外部プロジェクトで使用されるすべての生成物（ステップ 2 で更新したすべてのプロジェクトについて）をビルドします。
5. 外部プロジェクトを含む、ステップ 2 で更新しなかったすべてのプロジェクトを更新します。

   インストール プロジェクトをビルドしている場合は、ここではそれらを除外します。インストール プロジェクトのためのビルド プロセスで、後で更新する必要があります。

   注記：この更新により、ステップ 4 でビルドしたすべての製品が外部プロジェクトに選択されます。
6. コンフリクトを特定し、解決します。
7. 残りの生成物（ステップ 5 で更新したすべてのプロジェクト）をすべてビルドします。
8. インストール エリアまたは CD を準備します。

9. ソフトウェアをテストします。
10. 生成物と外部プロジェクトをチェックインするベースラインを作成します。
11. 最後の更新の後に、プロジェクト グルーピングを右クリックし、**ベースラインとタスクの自動更新**を選択します。

    まず、ライブラリなどのローレベル プロジェクトを更新し、ビルドします。次に、外部プロジェクトと、それを使用するハイレベル プロジェクト（実行形式ファイルなど）を更新します。これでハイレベル プロジェクトをビルドできます。

7 アプリケーションのパッケージング

アプリケーションのパッケージングとは、アプリケーションを、内部顧客（SQE など）または外部顧客（CIA など）に関わらず、任意の顧客にリリースする方法です。

各サイトでは、ソフトウェアに最も有用な方法でアプリケーションのパッケージングを行います。標準的なソフトウェア提供形態には以下のようなものがあります。

- DVD
- CD-ROM
- 企業のウェブ サイト

通常、製品を CD-ROM または企業ウェブ サイトで提供することが多いでしょう。パッケージングするデータを準備する方法の 1 つが、インストール プロジェクトからインストール エリアを作成することです。

インストール エリアとプロジェクトについて

製品ファイルを Rational Synergy で管理する場合、インストール エリアの構造を持つプロジェクトを作成できます。これを**インストール プロジェクト**と呼びます。

構造がインストール エリアと一致するインストール プロジェクトを作成できます。これで、インストール プロジェクトのワークエリアを使用してインストール イメージ（Windows）または一組の tar ファイル（UNIX）を作成し、それを CD に入れて、テストまたはソフトウェアのリリースに使用できます。

たとえば、テストのため内部的にソフトウェアをリリースする必要がある場合、統合テスト プロジェクトから開始し、テスターが使用するすべての製品とオブジェクトを含むインストール プロジェクトを作成します。最後に、インストール プロジェクトを統合テスト プロジェクト階層に追加します。チームの開発が SQE グループによるテストを要する時点に到達したら、システム テスト プロジェクトをビルドし、インストール プロジェクトを更新し、インストール イメージ（Windows）または一組の tar ファイル（UNIX）を作成して CD に入れます。

以下のセクション インストール プロジェクトの作成で、インストール プロジェクトの作成方法を説明します。

インストール プロジェクトの作成

適切なビルド管理プロジェクト階層があることを確認します。統合テスト プロジェクトの作成については、統合テスト プロジェクトの作成を参照してください。システム テスト プロジェクトの作成については、システム テスト プロジェクトの作成を参照してください。

注記：インストール プロジェクトの使用は任意です。作成する前に、チームのニーズを検討してください。

1. インストール プロジェクトを作成するタスクを作成し、自分を担当者にします。

   作成したタスクがカレント タスクに設定されます。
2. プロジェクトをインストール プロジェクトとしてコピーします。
   a. 新しいプロジェクトを作成します。
   b. 新しいプロジェクトのプロパティを設定します。
   c. **OK** をクリックします。
3. 新しいプロジェクトに、インストールエリアと一致するディレクトリを作成します。
4. ドラッグ アンド ドロップまたはコピーと貼り付けにより、製品オブジェクト（および成果物に含まれる他のオブジェクト）をインストール プロジェクトに追加します。
5. **ワーク** ペインに適切なビルド管理プロジェクトを表示します。
6. 新しいインストール プロジェクトをビルド管理プロジェクト階層に追加します。

   上記ステップ 4 を参照してください。
7. インストール イメージを構築するようにビルド スクリプトと makefile を拡張します。
8. カレント タスクを完了します。
9. 統合テスト プロジェクトで統合テスト サイクルを実行します。
10. システム テスト プロジェクトをベースラインのインストール プロジェクトからコピーします。
11. システム テスト プロジェクトでシステム テストのテスト サイクルを実行します。

    注記：統合テスト プロジェクトとシステム テスト プロジェクトをチェックアウトして、プラットフォームごとにこのプロセスを完了します。

インストール プロジェクトのためのビルド プロセス

通常のビルド プロセスについては、ビルド管理の基本を参照してください。

これは、段階的ビルド アプローチの別の段階です。インストール プロジェクトを含むプロジェクト階層は、すぐには完全に更新されません。これは、プロジェクト階層の一部がビルドされるまで、外部プロジェクトに取り込みたい新しい生成物が存在しないことがあり得るからです（生成物がビルド前の書き込み禁止状態にある場合、新しいバージョンがチェックアウトされます。ビルドの完了後、これらの新しいバージョンの生成物を選択して外部プロジェクトに組み込む必要があります）。まず階層の一部を更新して、段階的にビルドする必要があります。

新しいタスクを選択しないようにするには、初めての更新後にプロジェクト グルーピングの自動リフレッシュを解除し、最後の更新後にプロジェクト グルーピングの自動リフレッシュを再度設定します。

1. 各インストール プロジェクトを更新して、新たにビルドされた生成物を選択します。
2. コンフリクトを特定し、解決します。
3. 各インストール プロジェクトのワークエリアからインストール エリアまたは CD を準備します。

   まず、生成物を生成するプロジェクトを更新し、ビルドします。次に、インストール プロジェクトを更新し、ビルドされた生成物を取得します。

8 パラレル リリース

この章では、以下のパラレル リリースについて説明します。

- リリースのパッチの作成
- パラレル開発環境の使用

リリースのパッチの作成

Rational の定義では、パッチとは、1 つ以上の修正でアプリケーションを部分的に再ビルドする、任意のリリースのことです。

パッチを作成するためには、リリースされたソフトウェア バージョンが再現および再ビルド可能である必要があります。したがって、アプリケーションのビルドに使用したプロジェクトのベースラインを作成することは重要です。ベースラインによって、ビルドされたもののスナップショットが作成されます。リリースされたプロジェクトの再現または再ビルドが必要な場合（パッチの場合など）、これを使用することができます。

パッチ リリースの設定

パッチ リリースは小規模（シングル パッチ）の場合も、大規模（多数のパッチを含むサービス パック）の場合もあります。規模に関わらず、パッチにはリリースを設定する必要があります。

シングル パッチの名前は **toolkit/3.0patch** などのようになります。サービス パック規模のリリースの名前は **toolkit/2.0sp1** などのようになります。

パッチ リリースに使用する目的を確認します。通常、必要なのは **Collaborative Development**（個別開発）と **System Testing**（システム テスト）のみです。

含めるプロジェクト

パッチを作成する際、再ビルドする新しいプロジェクト バージョンを含める必要があります。以下に、パッチ プロジェクトに何を含めるかについて説明します。

- 実行形式ファイルの不具合を修正している場合、開発者はその実行形式ファイルを再ビルドするプロジェクトでコードを変更します。したがって、パッチには、実行形式ファイルをビルドするために使用する変更後コードを含むプロジェクト バージョンのみが含まれます。
- ライブラリに不具合がある場合は、開発者がコードを変更し、ビルド マネージャがライブラリを再ビルドし、それと関連するすべての実行形式ファイルを再ビルドします。したがって、ライブラリとそれにリンクするすべての実行形式ファイルのプロジェクトが必要になります。また、外部プロジェクトを使用する場合はそれも必要です。

  多くの場合、どのプロジェクトを再ビルドする必要があるかは、バグの修正を割り当てられた開発者に問い合わせることができます。

  注記：Rational Synergy データベースでパッチを整理するための**グルーピング プロジェクト**を設定できます。詳細については、グルーピング プロジェクトを参照してください。

  インストール イメージを使用してパッチをビルドする場合は、インストール プロジェクトが必要です。

開発者からの修正の取得

開発者から修正を取得するには、以下の方法があります。

1. 開発者がパッチ プロジェクトを作成し、バグを修正し、修正を単体テストし、タスクを完了します。
2. ビルド マネージャが、そのプロジェクトをコピーします。コピーを更新して開発者のタスクを取得します。

パッチ用リリースの作成

以下のプロセスでは、パッチ用リリース作成の大まかな手順を示します。実行する必要のあるローレベル操作の多くは、前述のセクションで説明しています。このセクションにあるリンクや自分で設定したブックマークを使用して該当セクションを参照できます。

このプロセスは、すべてのプロジェクトでリリース値を**コンフリクトなく**設定していることを前提としています。そうでない場合は、このプロセスは**うまくいきません**。

1. パッチ用のリリースを作成します（手順については、Rational Synergy ヘルプのリリースのコピーまたは作成を参照してください）。

   通常は、パッチを作成するリリースをコピーします。

2. **リリースの作成**ダイアログボックスでパッチに使用するプロセス ルールを設定します。

   通常、必要なのは **Collaborative Development**（共同開発）と **System Testing**（システム テスト）プロセス ルールのみです。

パッチの作成

以下に、パッチ作成の大まかな手順を示します。

1. タスク（1 つまたは複数）を作成し、パッチ リリースに適した開発者に割り当てます。
2. プロジェクト情報と新しいリリース値を開発者に付与します。

   開発者は新しい開発プロジェクトをベースライン リリースからコピーし、カレント タスクを設定し、問題の診断と修正、単体テストを行ってカレント タスクを完了する必要があります。

   注記：問題を修正している開発者は、必ずファイルのチェックアウト**前に**プロジェクトでリリース値を設定するようにしてください。また、開発者は、チェックアウトするオブジェクトが自動的に正しいタスクに関連付けられるようにカレント タスクを設定する必要があります。これらのステップを忘れると、いくつかのステップを手動で実行して正しいオブジェクトが確実にパッチに含まれるようにしなければなりません。

また、開発者は、パッチ プロジェクト内の他の作業を完了**しないでください**。これは、チェックアウトされた他のバージョンにパッチのリリース値のタグがつけられ、誤ってパッチに含まれないようにするためです。

3. パッチ作成中の各プロジェクトのシステム テスト プロジェクトを作成します。

   同様の手順でシステム テスト プロジェクトも作成できますが、**1 つだけ異なる点があります**。

   コピーするのは、パッチに含めるプロジェクトだけです。

   システム テスト プロジェクトの作成を参照してください。

   注記：統合 *prep*（準備）パッチ プロジェクトは必要ありません。

4. 特定のタスクを使用するビルド を参照してください。
5. テスト エリアを設定します。

   パッチ用のテスト エリアを設定するため、リリース済みソフトウェアのコピーをインストール エリア（**patch_test_1.2** など）にインストールし、そのパッチにビルドされた製品をそのエリアにコピーできます （理想的には、顧客がインストールする場合と同じようにパッチをインストールすべきです）。

   パッチ インストール用の特別なユーティリティがある場合は、それを使用します。
6. システム テストを続行します。 不具合が見つかったら、サイクルをステップ 1 から再開します（ただし、ステップ 2 とステップ 3 は繰り返さないでください）。
7. パッチが品質基準を達成したら、顧客がそのパッチを利用できるようにします。

パラレル開発環境の使用

パラレル開発環境は、複数のプラットフォームについて製品を出荷する（Windows に 1 つ、UNIX に 1 つなど）ことを会社が決めた場合や、製品のリリースを複数出荷する（メインの製品リリースとパッチ リリースなど）必要がある場合に発生します。

パラレル プラットフォーム環境についてはパラレル プラットフォームのビルドを、環境の設定方法についてはパラレル プラットフォームの設定を参照してください。パラレル リリース環境については、パラレル リリースの設定を参照してください。

パラレル プラットフォームのビルド

ソフトウェアを複数のプラットフォーム向けにビルドする必要がある場合、プラットフォーム プロパティを使用して、プラットフォームごとに各プロジェクトのバージョンを作成します。これらのプロジェクトは、バリアント プロジェクトと呼ばれます。バリアント プロジェクトは、ほとんどのソース メンバーを共有しますが、異なるビルド引数を設定して、その結果発生する複数の製品を各バリアント プロジェクトで保存できます。ただし、特定のプラットフォームに個々のタスクを指定したり、プラットフォームごとにフォルダを設定する必要はありません。1 つのタスクにすべてのプラットフォームの変更されたファイルを含めることができます。パラレル バージョンが発生すると、各プロジェクトはプラットフォームに適合するオブジェクト バージョンを選択します。

たとえば、Windows と HP-UX® の **toolkit** プロジェクトをビルドするには、プロジェクト階層の 2 つの異なったバージョンをコピーします。それぞれの **プラットフォーム** プロパティを適切な値に設定します（プラットフォーム値を **om_hosts.cfg** ファイルで設定しておく必要があります。これについては、プラットフォーム ファイルについてを参照してください）。

sp1_win32_2.0、**hp_2.0** など、プロジェクトが分かりやすいバージョン名を付与できます。
プロジェクト名をこのように付けると一目で識別できます。

> **注意！** プラットフォーム属性をオブジェクトに追加した場合、将来のバージョンからその属性を削除する際、注意が必要となります。たとえば、製品のリリース 1 で 2 つのパラレル バージョンを持つファイルがあったとします。バージョン `win_1` のプラットフォーム値は `x86`、バージョン `sol_1` のプラットフォーム値は `sparc` です。リリース 2 で、これら 2 つのパラレル ファイルをマージして、クロス プラットフォームのバージョン 2 を作成することにしました。バージョン 2 ではプラットフォーム属性を消去します。Rational Synergy は、プラットフォーム値の一致を優先するため、プラットフォーム値 `x86` を持つプロジェクトは、マージ後のバージョン 2 ではなく、バージョン `win_1` を取り込

みます。
この問題を解決するには、古い `win_1` バージョンと `sol_1` バージョンからプラットフォーム属性を取り除きます。ただし、この方法をとると古いリリースのパッチはビルドできなくなります。これを解決するためには、マージ後のオブジェクトの名前を変更し、古いバージョンが候補にならないようにします。

製品もプラットフォーム固有です。プラットフォームごとに各製品のパラレル ブランチをチェックアウトし、適宜プラットフォーム値を設定する必要があります。

注記：ユーザーは、同じプロジェクトを使用して、ビルド前に、**プラットフォーム** プロパティ、make マクロ、ワークエリアを変更することにより、複数の異なるプラットフォーム向けに同じ製品をビルドできます。

ただし、これはユーザーがビルドを実行するのによい方法では**ありません**。

ビルド マネージャは、自分がビルドする製品を再現できる必要があります。複数のプラットフォームをビルドするために構成を変更し続けていると、製品がどのようにビルドされたかを把握できなくなります。その結果、問題の追跡、修正のテスト、マイルストーン到達時のソフトウェア維持が非常に困難になります。

また、この方法でビルドすると、プラットフォーム変更のたびに再ビルドが必要になります。

パラレル プラットフォーム用の更新のしくみについては、選択ルールの作業を参照してください。

パラレル プラットフォームの設定

この手順では、パラレル プラットフォーム用に新しいプロジェクト階層を既存のプロジェクト階層からコピーする方法を示します。

1. **om_hosts.cfg** ファイルに、必要となる新しいプラットフォーム値を設定します。

   手順については、プラットフォーム ファイルについてを参照してください。

2. 既存のプロジェクト階層から新しいプロジェクト階層をコピーします。

   必ずプラットフォーム値を設定し、プロジェクト バージョンが分かりやすい名前を付けてください。パラレル プロジェクト用のワークエリアが

明確になるように、必ずバージョンとプラットフォームをワークエリア パスに使用します。

3. 新しいプロジェクト階層がうまくビルドされることを確認します。

   makefile とプロジェクト マクロを変更する必要がある場合があります。

4. 新しいプロジェクト階層にベースラインを設定します。

   手順については、開発者にテスト ベースラインを公開を参照してください。

これで、このベースラインからビルド管理プロジェクトをコピーできます。

パラレル リリースの設定

企業でアプリケーションのパラレル リリースを並行して開発することがあります。たとえば、あるチームで **toolkit** アプリケーションのリリース **toolkit/3.0** の新機能に取り組んでいる一方で、別のチームがリリース **toolkit/2.1** のバグ修正の作業をしているような場合です。

複数リリースのためにアプリケーションをビルドする必要があるため、開発リリースごとに別のプロジェクト バージョンを作成しなければなりません。

たとえば、チームで新機能リリース **toolkit/3.0** の作業と並行してバグ修正リリース **toolkit/2.1** の作業をしており、リリース **toolkit/3.0** にはリリース **toolkit/2.1** のバグ修正を含め、リリース **toolkit/2.1** にはリリース **toolkit/3.0** の新機能を含めないものとします。

この場合は、リリース **toolkit/3.0** のプロセス ルールを以下のように変更します。**All Completed Tasks for Release toolkit/2.1** フォルダ（フォルダ テンプレート**ではなく**）を **toolkit/3.0** 統合テスト プロセス ルールに追加し、**toolkit/3.0** 統合テスト プロジェクトで両方のリリースからタスクが取り込まれるようにします。**toolkit/3.0** リリースの別のプロセス ルールも同じように変更する必要があります。

これによって開発者がパラレル変更をマージする必要がなくなるわけではありません。Joe が **toolkit/3.0** のファイルを変更して、Mary が **toolkit/2.1** の同じファイルを変更した場合、これらの変更はパラレルとなります（Mary の変更のほうが新しいので、**toolkit/3.0** に選択されます）。2 つのバージョンをマージして **toolkit/3.0** の新しいバージョンにする必要があります。

9 プロジェクトの再構築

プロジェクト再構築とは、既存ディレクトリをプロジェクトに変換したり、階層に対してプロジェクトの追加や削除を行ったりすることにより、統合テストまたはシステム テスト プロジェクトのメンバーを再整理することです。

サイトでプロジェクト再構築を決定する理由は、以下を初めとしてたくさんあります。

- 製品の方向性が変わり、階層からサブプロジェクトを削除する必要が生じた。
- プロジェクトが大きくなりすぎ、小さいものに分割することにした。
- チームで製品に多数の新機能を追加し、階層にサブプロジェクトを追加する必要が生じた。
- ソフトウェアの一部の担当チームが変わり、別のプロジェクトに移すことにした。
- 次のリリースで、製品に大きな影響を及ぼす変更を加えることをチームが決定し、階層でサブプロジェクトを使用解除する必要が生じた。
- 外部プロジェクトを追加することにした。
- インストール プロジェクトを追加することにした。

プロジェクトを再構築する場合は必ず、それに対応するように makefile、ビルド プロセス、すべての自動化ジョブを変更する必要があります。

統合テスト プロジェクト階層とシステム テスト プロジェクト階層の両方に変更を適用する必要があります。統合テスト プロジェクト階層を先に更新し、次に、新しいプロジェクトをチェックアウトした後で更新して変更を取り込むことにより、システム テスト プロジェクト階層に変更を適用してください。

また、プロジェクト再構築時は、更新を実行し、プロジェクト階層の再ビルドも行って、アプリケーションの整合性を維持する必要があります。統合テスト プロジェクトについては、通常の短いテスト スイートで十分です。システム テスト プロジェクトについては、SQE チームでアプリケーションを再テストする必要があるでしょう。

ここで説明する手順の流れは以下のとおりです。

- 階層への既存プロジェクトの追加
- 階層からのプロジェクトの切り取り
- 階層からのプロジェクトの削除
- ディレクトリからサブプロジェクトへの変換
- 既存階層への新規プロジェクトの追加

  注記：プロジェクトを再構築する場合は、統合テスト プロジェクトを更新し、まず統合テスト サイクルを実行して問

題を特定して修正してください。システム テスト サイクル中に変更が自動的に選択され、システム テスト プロジェクトに含められます。

階層への既存プロジェクトの追加

1. タスクを作成し、自分を担当者にします。

   新しいタスクがカレント タスクに設定されます。
2. 既存プロジェクトを追加するプロジェクトを参照します。
3. ドラッグ　アンド　ドロップまたはコピーと貼り付けにより、既存のプロジェクトをカレント プロジェクトに追加します。

   追加したいプロジェクトの名前が分からない場合、**クエリ** ダイアログ ボックスを使用してプロジェクトを検索します。
4. カレント タスクを完了します。

階層からのプロジェクトの切り取り

1. タスクを作成し、自分を担当者にします。

   新しいタスクがカレント タスクに設定されます。
2. 切り取るサブプロジェクトを含む親プロジェクトを参照します。
3. サブプロジェクトを右クリックし、**切り取り**を選択します。

   この操作によってプロジェクトからサブプロジェクトが切り取られますが、データベースからは削除されません。
4. カレント タスクを完了します。

階層からのプロジェクトの削除

注記：削除操作により、プロジェクトがデータベースから完全に削除されます。

階層からプロジェクトを削除してもデータベースには残しておきたい場合は、階層からのプロジェクトの切り取りを参照してください。

1. 削除するプロジェクトを参照します。
2. 削除するプロジェクトを右クリックして**削除**を選択します。

   Scope で適切な削除範囲（プロジェクト、プロジェクトとメンバーなど）を選択します。

ディレクトリからサブプロジェクトへの変換

この操作は CLI からのみ実行できます。

1. コマンド プロンプトから Rational Synergy を開始します。

   ```
   ccm start -h engine_hostname -d database_path -nogui
   ```

   セッションの開始後、コマンド ウィンドウ（Windows）またはセッションを開始したシェル（UNIX）に Rational Synergy アドレス（CCM_ADDR）が表示されます。
2. ロールを *build_mgr*（ビルド マネージャ）に設定します。

   ```
   ccm set role build_mgr
   ```
3. タスクを作成し、自分に割り当て、デフォルトとして設定します。

   ```
   ccm task -create -synopsis "string" -default
   ```
4. プロジェクトに変換するディレクトリの上のワークエリア内ディレクトリに移動します。
5. プロジェクトを作成し、ディレクトリをルートに指定します。

   ```
   ccm create -type project -root existing_dir -version int
   -release release -purpose "Integration Testing"
   ```
6. 必要に応じてサブプロジェクトのプラットフォームを作成します。

   ```
   ccm attr -create platform -type string -value platform
   -project project_spec
   ```
7. `ccm unuse` コマンドを使用してディレクトリを使用解除します。
8. 新しい統合テスト プロジェクトを統合テスト プロジェクト階層に追加します。

   ```
   ccm use -p project_name delimiter version
   ```
9. **Windows ユーザー**：絶対サブプロジェクトを使用する場合は、makefile、ビルド プロセス、すべての自動化ジョブを、変更が反映されるように変更します。

   相対サブプロジェクトを使用する場合は、変更の必要はありません。
10. カレント タスクを完了します。

    ```
    ccm task -complete default
    ```
11. 統合テスト サイクルを実行して、ベースラインの作成を行います。
12. システム テスト プロジェクトを新しいプロジェクトからコピーします。

    この操作については、システム テスト プロジェクトの作成を参照してください。
13. 最上位のシステム テスト プロジェクトを更新し、アプリケーションを再ビルドして、テスト スイートを実行します。

注記：統合とシステム テスト プロジェクトをチェックアウトし、プラットフォームごとにこのプロセスを繰り返します。

14. Rational Synergy CLI を終了します。

```
ccm stop
```

既存階層への新規プロジェクトの追加

既存階層に追加する必要のある新しいプロジェクトを開発者が作成した場合、以下のステップ a とステップ b を実行する必要があります。この操作を行う開発者は、Rational Synergy Classic で *component_developer* ロールを設定する必要があります。

1. 開発者は必ず以下のことを行います。
   a. 開発者はカレント タスクを完了する必要があります。
   b. 開発者は新しいプロジェクトをチェックインする必要があります。
2. チェックインしたプロジェクトから統合テスト プロジェクトをコピーします。

   バージョン、目的、プラットフォーム、リリースを必ず設定してください。
3. 新しい統合テスト プロジェクトを統合テスト プロジェクト階層に追加します。

   開発者がすでにプロジェクトを階層内のディレクトリに追加し、そのディレクトリをチェックインしている場合は、統合テスト プロジェクト階層を更新して、Rational Synergy で確実に新しいディレクトリが選択され、新しいプロジェクトが含まれるようにしてください。更新については、プロジェクトの更新を参照してください。

   開発者が新しいプロジェクトを階層に追加していない場合は、ビルド マネージャがプロジェクトを統合テスト プロジェクト階層に追加する必要があります（手順については、階層への既存プロジェクトの追加を参照してください）。また、タスクを作成し、変更が完了した時点でタスクを完了する必要があります。

   新しいプロジェクトに空のディレクトリ エントリがある場合は、このリリースのタスクに関連付けられていないオブジェクトがある可能性があります。
4. 必要に応じて外部プロジェクトを作成します。1 つを統合ビルド管理プロジェクト用に、1 つをシステム テスト プロジェクト用に追加する必要があります。

   この操作については、外部プロジェクトの作成を参照してください。

5. 必要に応じてパラレル　プラットフォームのバージョンを作成します。1 つを統合テスト プロジェクト用に、1 つをシステム テスト プロジェクト用に追加する必要があります。

   この操作については、プラットフォーム ファイルについてを参照してください。

6. 新しいプロジェクトが複数のリリースに適用される場合、そのパラレル リリース バージョンを作成します。1 つを統合テスト プロジェクト用に、1 つをシステム テスト プロジェクト用に追加する必要があります。

   この操作については、パラレル リリースの設定を参照してください。

7. makefile、ビルド プロセス、すべての自動化ジョブを、変更が反映されるように変更します。
8. 再構築に使用したタスクをすべて完了します。
9. 開発者にテスト ベースラインを公開を参照してください。
10. 統合テスト プロジェクト階層を更新し、アプリケーションを再ビルドして、テスト スイートを実行します。
11. 別のベースラインを作成します。
12. 新しい統合テスト プロジェクトそれぞれに対応するシステム テスト プロジェクトを作成します。

    この操作については、システム テスト プロジェクトの作成を参照してください。

13. システム テスト プロジェクト階層を更新し、アプリケーションを再ビルドして、テスト スイートを実行します。
14. システム テストを実行します。

    注記：統合テスト プロジェクトとシステム テスト プロジェクトをチェックアウトして、プラットフォームごとにこのプロセスを完了します。

10 さまざまなビルド管理

アプリケーションを構築する方法は、企業によって異なります。企業にはそれぞれ固有のニーズがあります。新しくて規模の小さい Q 社は提供製品が 1 つしかないのに対し、大企業の V 社は数多くの製品を複数のプラットフォームで提供しているなどという場合があります。

Rational Synergy のタスクベース方法論では、Rational Synergy のさまざまな機能が使用されます。ビルド マネージャが使用する機能の一部は、必須ではないので、このドキュメントでは詳しく取り上げません。ただし、それらの機能により操作が自動化され、開発者が複雑な操作を行わなくてもよくなるため、ビルド マネージャの作業は大幅に簡素化されます。

ここで説明する内容は、これまでの章で説明した標準の方法論が完全に当てはまらない企業のためのものです。標準の手法で対応できない場合は、ビルド管理の遂行に方法論のバリアントを用いてください。以下のバリアントについて説明します。

UNIX と PC 両方のビルド管理

UNIX と PC の両方で動作するアプリケーションの場合、ビルドの前に必ず以下の点を考慮してください。

- ワークエリアの視認性

  UNIX での Rational Synergy セッションは、PC のワークエリアでは見えません。また、PC でのセッションは UNIX のワークエリアでは見えません。

- Makefile のフォーマット

  多くの場合、UNIX と PC でのビルドのためにパラレル makefile がすでに存在しています。その場合、双方のプラットフォームの makefile を使用してもかまいません。そうでない場合は以下の手順に従います。

  a. プラットフォームごとにプロジェクトのパラレル バージョンを設定します。

  b. 各 makefile の**プラットフォーム** プロパティを適切な値に設定し、makefile のパラレル バージョンを設定します。

  プロジェクトのパラレル バージョンを更新すると、プラットフォームごとに適切な makefile が取り込まれます。

- 自動化

  更新／ビルド プロセスを自動化する場合、シェル スクリプトまたはバッチ ファイルを使用して、Windows と UNIX のジョブを別個に自動化するか、Perl や Cygwin などのクロスプラットフォーム スクリプトを使用する必要があります。

UNIX ワークエリアとローカル ファイル

デフォルトでは、UNIX ワークエリアには、管理対象ファイルを含む確実なディレクトリ構造へのシンボリック リンクが含まれています。しかし、必要に応じてローカル ファイルのコピーを含む UNIX ワークエリアを設定することもできます。デフォルトを変更する前に、以下の利点と欠点を考慮してください。

利点

- ローカル マシンを切断して自分のワークエリアを使用できる。

  これは、開発者にとって有益です。

- すべてのファイルがローカルであれば、NFS のようなファイル サーバーでアクセスするよりもビルド時間が短くなることがある。

  大人数のソフトウェア チームではファイル サーバーを通じてビルドするとパフォーマンスが低く、ビルドができない場合があります。ビルド時にローカル ディスクにファイルがあれば、ファイル サーバーを通じてファイルにアクセスする必要がありません。

欠点

- 通常の操作が若干遅くなる。

  ローカル コピーを使用するとローカルのビルドを速くすることはできますが、チェックアウトやチェックイン、更新など他の通常操作は少し遅くなります。これは、データベースとワークエリアの間で、ネットワークを通じてファイルを相互コピーする必要があるためです。

- 共有ファイルのコピーが複数あり、Rational Synergy 外からアクセスすると問題が発生することがある。

  ローカル コピーがある場合、開発者が 2 つのプロジェクトで同じ *working*（作業）オブジェクト バージョンを持っているので、そのファイルのコピーが 2 つあることになります。ワークエリアごとに自分のコピーを持ちます。ファイルの各コピーは、他のコピーを認識しません。

  Rational Synergy では、すべての視認可能なワークエリアにあるファイルのコピーが Rational Synergy によって常に最新のものに維持されます（視認可能なワークエリアとは、操作を実行している Rational Synergy クライアントにファイル システムの場所が見えるワークエリアのことです。Rational Synergy クライアントにファイルのあるワークエリアがすべて見えているかぎり、Rational Synergy の作業中、変更があれば各ワークエリアは更新されます）。しかし、自分のワークエリアで直接（FrameMaker™ などで）作業すると、作業中のファイルだけが変更されます。

  注記：複数ワークエリアで複数のファイル コピーを変更する際、同期しなかったり、Rational Synergy を使用せずにファイルにアクセスしたりすると、何らかの操作（プロジェクトやサブプロジェクトのワークエリア コンフリクトを検

出するのに**同期**オプションを使用して）を行わなければ解決できないワークエリア コンフリクトが発生します。

グルーピング プロジェクト

グルーピング プロジェクトにより、プロジェクトが明確に分類されます。たとえば、グルーピング プロジェクトにより、ソフトウェア アプリケーションの複数のプラットフォームを含めることができます。すべてのプロジェクトが 1 つの大きな階層構造に入っている場合、**プロジェクトのコピー** ダイアログボックスの**サブプロジェクト** リスト ボックス オプションを使用して、すべてのプロジェクトの新規バージョンをチェックアウトできます。

注記：グルーピング プロジェクトは必須ではありません。しかし、すべてのプロジェクトが 1 つの階層に入れられている場合、プロジェクトのセットのチェックインや、プロジェクトの新しいセットのチェックアウトが簡単になるので、グルーピング プロジェクトは非常に便利です。

グルーピング プロジェクトの作成について

例として、**toolkit_top-3.0** というプロジェクトの複数のプラットフォームを 1 つにグループ化します。このプロジェクトにより、**toolkit-win** と **toolkit-unix** をひとまとめにします。

toolkit_top-3.0 グルーピング プロジェクトを設定するには、固有の名前を持つプロジェクトを設定する必要があります。

名前が同じ 2 つのプロジェクトを、同じ親プロジェクト内のサブプロジェクトにすることはできません。たとえば、**toolkit-win** と **toolkit-unix** を同じプロジェクトに分類することはできません。

同じ名前のプロジェクトを分類するため、固有の名前をもつプロジェクトのレベルをもう 1 つ作成します。ワークエリアを持つ 1 つのプロジェクトに、親プロジェクトから見えないワークエリアを持つサブプロジェクトを含めることはできません。この場合、グルーピング プロジェクトの作成 のステップ 3 で説明しているように、ワークエリア管理を無効にする必要があります。

グルーピング プロジェクトとプロジェクト グルーピング

グルーピング プロジェクトとプロジェクト グルーピングは同じではありません。プロジェクト グルーピングは Rational Synergy によって自動的に作成されるもので、同じリリースと目的に所属するすべてのプロジェクトをまとめます。しかし、プロジェクトをいったんチェックインすると、そのプロジェクトはどのプロジェクト グルーピングのメンバーでもなくなります。

グルーピング プロジェクトを作成して、これを完全に管理することができます。

グルーピング プロジェクトの作成

この例では、**toolkit_top-3.0** というプロジェクトを作成し、このプロジェクトに複数のプラットフォームの全 **toolkit** プロジェクトを含むものとします。この操作を始める前に、必ずグルーピング プロジェクトの作成についてを読んでください。

1. プロジェクトを作成し、プラットフォームごとに固有の名前を付けます。

   特定プラットフォーム用のプロジェクトを含む場合は、必ず各プロジェクトのプラットフォーム値を設定します。

2. 新規プラットフォーム プロジェクトそれぞれに対し、そのプラットフォームの既存プロジェクトをサブプロジェクトとして追加します （このためには、ドラッグ アンド ドロップを行います)。

   追加するプロジェクトの名前がわからない場合は、**クエリ** ダイアログ ボックスを使用してプロジェクトを検索し、ドラッグ アンド ドロップを行って追加します。

3. 最上位グルーピング プロジェクトを作成し、新規グルーピング プロジェクトのワークエリア管理を無効にします。

   a. 新しいプロジェクトを作成します。

      タスク > 新規 > プロジェクト

   b. 作成したプロジェクトを右クリックし、**プロパティ**を選択します。**ワークエリア** タブで、ワークエリア管理を無効にします。**OK** をクリックして変更を保存します。

      注記：プラットフォーム値は設定しないでください。このプロジェクトには、プラットフォーム値の異なる複数のサブプロジェクトが含まれます。最上位グルーピング プロジェクトにプラットフォーム値を設定すると、最上位グルーピング プロジェクトの更新後、同じプラットフォーム値を持つサブプロジェクトのみがプロジェクトに含められます。

   c. 各新規プラットフォーム　プロジェクトを最上位グルーピング　プロジェクトのメンバーとして追加します。

      ルート ディレクトリを右クリックし、**メンバーの作成**をポイントし、**サブプロジェクト**を選択します。**サブプロジェクトの作成**ダイアログ ボックスで新しいサブプロジェクトの名前を入力します。サブプロジェクトがある場合、最上位のグルーピング プロジェクトの下にドラッグ アンド ドロップします。

リリースのパッチの作成

以下に、パッチ用リリース作成の大まかな手順を示します。実行する必要のあるローレベル操作の多くは、前述のセクションで説明しています。このセクションにあるリンクや自分で設定したブックマークを使用して該当セクションを参照できます。

このプロセスは、すべてのプロジェクトでリリース値を**コンフリクトなく**設定していることを前提としています。そうでない場合は、このプロセスは**うまくいきません**。

1. パッチ用のリリースを作成します（手順については、Rational Synergy ヘルプのリリースのコピーまたは作成を参照してください）。

   通常は、パッチを作成するリリースをコピーします。

2. **リリースの作成** ダイアログボックスでパッチに使用するプロセス ルールを設定します。

   通常、必要なのは **Collaborative Development**（共同開発）と **System Testing**（システム テスト）プロセス ルールのみです。

パッチの作成

以下のプロセスでは、パッチ作成の大まかな手順を示します。

1. タスク（1 つまたは複数）を作成し、パッチ リリースに適した開発者に割り当てます。
2. プロジェクト情報と新しいリリース値を開発者に付与します。

   開発者は新しい開発プロジェクトをベースライン リリースからコピーし、カレント タスクを設定し、問題の診断と修正、単体テストを行ってカレント タスクを完了する必要があります。

   注記：問題を修正している開発者は、必ずファイルのチェックアウト**前に**プロジェクトでリリース値を設定するようにしてください。また、開発者は、チェックアウトするオブジェクトが自動的に正しいタスクに関連付けられるようにカレント タスクを設定する必要があります。これらのステップを忘れると、いくつかのステップを手動で実行して正しいオブジェクトが確実にパッチに含まれるようにしなければなりません。

   また、開発者は、パッチ プロジェクト内の他の作業を完了**しないでください**。これは、チェックアウトされた他のバージョンにパッチのリリース値のタグがつけられ、誤ってパッチに含まれないようにするためです。

3. パッチ作成中の各プロジェクトのシステム テスト プロジェクトを作成します。

   同様の手順でシステム テスト プロジェクトも作成できますが、**1 つだけ異なる点があります**。

   コピーするのは、パッチに含めるプロジェクトだけです。

   システム テスト プロジェクトの作成については、システム テスト プロジェクトの作成を参照してください。

注記：統合 *prep*（準備）パッチ プロジェクトは必要ありません。

4. 特定のタスクを使用するビルド を参照してください。
5. テスト エリアを設定します。

   パッチ用のテスト エリアを設定するため、リリース済みソフトウェアのコピーをインストール エリア（**patch_test_1.2** など）にインストールし、そのパッチにビルドされた製品をそのエリアにコピーできます （理想的には、顧客がインストールする場合と同じようにパッチをインストールすべきです）。

   パッチ インストール用の特別なユーティリティがある場合は、それを使用します。
6. システムテストを続行します。不具合が見つかったら、サイクルをステップ 1 から再開します（ただし、ステップ 2 とステップ 3 は繰り返さないでください）。
7. パッチが品質基準を達成したら、顧客がそのパッチを利用できるようにします。
8. 新しいプロジェクトをチェックインし、新しいオブジェクトと階層内の他のプロジェクトを含むベースラインを作成します。

カスタム フォルダ テンプレート クエリの作成

ビルド マネージャは、特定のプロパティを持つタスクを収集するフォルダのフォルダ テンプレートを作成する必要があるかもしれません。

このシナリオでは、チームで 2 つのパラレル リリースを同時に開発しています。両方のリリース（ここではリリース **toolkit/2.1** とリリース **toolkit/3.0**）の変更を収集する完了タスク フォルダを設定します。

1. **クエリ** ダイアログボックスを使用して、コピーするフォルダ テンプレートを検索します。

   検索 > フォルダ テンプレート

   クエリ ダイアログボックスが表示されます。
2. カスタマイズするフォルダ テンプレートを選択し、**フォルダ テンプレートのコピー**をクリックします。

   フォルダ テンプレートのコピー ダイアログボックスが表示されます。
3. 以下のカスタム クエリを設定します。
   a. 最初のリストを **状態で検索** に設定し、状態を **completed** に設定します。

b. **リリース** リストを適切なリリースに設定します。

c. 必要に応じて、別のクエリ文節を別のリリース用に追加します。

d. **修正データベースで検索** リストを適切なデータベース名に設定します。

4. **OK** をクリックして設定を保存します。

組み込まれているフォルダ テンプレートはカスタマイズできません。

テスト フェーズの追加

テスト フェーズを追加するには、フェーズを示すプロセス ルールと目的を作成する必要があります。

テスト フェーズの追加は、リリースの任意の時点で行うことができます。

1. プロセス ルールを作成します（Rational Synergy ヘルプのプロセス ルールの設定を参照してください）。
2. 目的の作成が必要な場合は、Rational Synergy ヘルプの目的の作成を参照してください。
3. 新しいプロセス ルールを使用するリリースを編集します。

   リリースを右クリックし、**プロパティ**を選択します。

   プロセス ルール タブで、**プロセス ルールの追加**をクリックし、新しいプロセス ルールを選択して、**OK** をクリックします。

これで、新しい目的のプロジェクトをコピーできます。

付録 A：プロセス ルールへの変換

この章には、既存のユーザーに有益な情報が掲載されています。また、プロセス ルールを使用せずに新規システムをスタートした後に、プロセス ルールへの移行を決めたユーザーも、この章を読むとよいでしょう。

新規ユーザーはこの章をスキップし、ビルド管理の準備とビルド管理の基本の手順を参照することも可能です。プロセス ルール使用について、新規ユーザーを対象とした説明が掲載されています。

プロセス ルールの必要性

目的、プロセス ルール、フォルダ テンプレートを利用したビルド管理方法論を使用してください。方法論に関する説明は、これらの機能を中心としています。

手動によるプロジェクトの更新は古い機能です。すべてのサイトで、プロセス ルールを使用する必要があります。

プロジェクトの変換

サイトでは、プロセス ルール使用への変換をいつでも行うことができます。リリース開始時であれば、チームが確実に一貫してプロセス ルールを使用することになるので、変換しやすくなります。

以下のセクションで、ビルド マネージャと開発者の変換プロセスについて説明します。

ビルド マネージャのプロセス ルール変換手順

ユーザーがプロジェクトを変換してプロセス ルールを使用するようにするためには、ビルド マネージャが以下を実行する必要があります。この操作は Rational Synergy CLI から実行できます。

1. コマンド プロンプトから Rational Synergy を開始します。

   ```
   ccm start -h engine_host -d database_path -nogui
   ```

   セッションの開始後、コマンド ウィンドウ（Windows）またはセッションを開始したシェル（UNIX）に Rational Synergy アドレス（CCM_ADDR）が表示されます。

2. ロールを *build_mgr*（ビルド マネージャ）に設定します。

   ```
   ccm set role build_mgr
   ```

3. 新しいリリースのすべてのプロセス ルールを検索します。

   選択セット（query output）でコマンドを実行します。

   ```
   ccm query -type process_rule "release='new_release'"
   ```

   ここで、*'new_release'* は新しいリリースの名前です。

4. 新しいプロジェクトで、デフォルトでプロセス ルールを使用するよう指定します。

   ```
   ccm process_rule -modify -default @
   ```

   手動による更新から変換する場合は、アクティブ プロジェクトのプロセスとプロセス ルールを設定する必要があります。Rational Synergy に組み込まれているデフォルトのプロセスとプロセス ルールが、サイトのニーズと合致する場合があります。あるいは、カスタムのプロセスとプロセス ルールを作成する必要があるかもしれません。

5. 開発者のプロセス ルール変換手順で説明するとおり、開発者に、自分のプロジェクトを変換してプロセス ルールを使用するようにできる旨を通知します。

6. Rational Synergy CLI を終了します。

   ```
   ccm stop
   ```

開発者のプロセス ルール変換手順

開発者は、プロセス ルールを使用できるようになったら、自分の既存プロジェクトを変換する必要があります。新規リリースを開始する場合は、開発者は以前のプロジェクトを変換する代わりに新しいプロジェクトをコピーできます。

既存のプロジェクトを変換してプロセス ルールを使用するようにするには、以下のステップを実行します（この手順を実行するには、ビルド マネージャがカレント リリースのプロセス ルールを設定している必要があります。ビルド マネージャのプロセス ルール変換手順を参照してください）。

1. **プロジェクト プロパティ** ダイアログボックスを使用して、各プロジェクトのリリース設定を変更します。階層内の全プロジェクトのリリース値を変更するには、以下を実行します。
   a. 最上位のプロジェクトを右クリックし、**プロパティ**を選択します。
      プロジェクト プロパティ ダイアログボックスが表示されます。
   b. **リリース** リストから新しいリリース値を選択します。
      プロジェクトにサブプロジェクトがある場合、すべてのサブプロジェクトでリリースが変更されます。
   c. **目的**が **Insulated Development**（個別開発）、**Collaborative Development**（共同開発）、または **Custom Development**（カスタム開発）に設定されていることを確認してください。**Custom Development**（カスタム開発）使用する場合は、自分のベースラインを選択する必要があります（説明については、Rational Synergy ヘルプの新規ベースラインの選択を参照してください）。
   d. 変更を保存します。
2. コマンド プロンプトから Rational Synergy を開始します。
   ```
   ccm start -h engine_host -d database_path -nogui
   ```
   セッションの開始後、コマンド ウィンドウ（Windows）またはセッションを開始したシェル（UNIX）に Rational Synergy アドレス（CCM_ADDR）が表示されます。
3. 新しいリリースのすべてのプロセス ルールを検索します。
   選択セット（query output）でコマンドを実行します。
   ```
   ccm query -type process_rule "release='new_release'"
   ```
   ここで、*'new_release'* は新しいリリースの名前です。
4. 新しいプロジェクトで、デフォルトでプロセス ルールを使用するよう指定します。
   ```
   ccm process_rule -modify -default @
   ```
5. Rational Synergy CLI を終了します。
   ```
   ccm stop
   ```

「ビルド管理の準備」の章から直接この章に進んできて、再度操作のフローを把握したい場合は、次にパラレル リリースとパラレル プラットフォームについてを読んでください。

付録 B：特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、または サービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の 製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの 評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示 もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムと その他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、製造元に連絡してください。

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, Massachusetts 02142
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

IBM および関連の商標については、www.ibm.com/legal/copytrade.shtml をご覧ください。Microsoft、Windows、Windows 2003、Windows XP、Windows Vista および / またはその他の Microsoft 製品は、Microsoft Corporation の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

用語解説

- テスト ベースライン
- データベース
- デフォルト タスク、カレント タスクを参照
- 同期
- トラディショナル モード
- バージョン
- パラレル コンフリクト
- パラレル バージョン
- 汎用プロセス ルール
- 比較
- ビルド
- ビルド マネージャ
- ファイル
- フォルダ
- ブレッドクラム
- プロジェクト
- プロジェクト グルーピング
- プロジェクト タスク
- プロジェクトの 更新プロパティ
- プロジェクトのコピー
- プロセス
- プロセス ルール
- プロパティ
- ベースライン
- ベースライン プロジェクト
- 別バージョンの使用
- 変更依頼
- マージ
- マージ コンフリクト
- 目的
- リコンサイル、同期を参照
- リコンフィギュア、更新を参照
- リリース
- リリース

- 履歴
- ワークエリア
- ワークエリア コンフリクト

DCM

Rational Synergy の分散型変更管理（Distributed Change Management）を使用すると、複数のデータベース間でデータを転送でき、マルチサイト開発を可能にします。

Rational Change

Rational Change は、Rational Synergy と統合された、ウェブベースの変更依頼管理システムです。本書では、Rational Change を Rational Synergy とともに使用する場合の説明をしています。

インスタンス

インスタンスはオブジェクトのプロパティです。同じ名前、タイプの複数のオブジェクトを区別するために使われます。それぞれのバージョンではありません。

インライン差分

インライン差分は、2 つの比較ファイルの個々に修正された行と文字の差分を表示します。

ウェブ モード

Rational Synergy 7.1a では、Rational Synergy クライアントが HTTP プロトコルでウェブベースの Rational Synergy サーバーと通信する、新しいアーキテクチャを導入しています。

オブジェクト

オブジェクトは Rational データベースに格納できる（ファイル、ディレクトリ、プロジェクト、タスク、変更依頼を含む）データの異なるタイプを示す総称です。

カレント タスク

カレント タスクは、現在作業しているタスクです。

管理製品

管理製品はビルド、または生成されたファイルです。管理製品はプロジェクトとディレクトリ以外のどのようなオブジェクト タイプにもなりますが、一般的な管理製品は実行形式ファイルとライブラリです。

共通祖先

共通祖先は、マージされている 2 つのファイルの最新に一番近い祖先です。

グルーピング プロジェクト

論理グループを作成するために使用するプロジェクトです。これにより、ワークエリアの使用が不要または不可能となります。

更新

更新操作により、自分のプロジェクトやディレクトリを他のユーザーによってチェックインされた最新のバージョンで更新できます（旧リリースでは、「更新」は「リコンフィギュア」と呼んでいました）。

更新プロパティ

誰かがプロジェクトを更新するとき、選択するオブジェクト バージョンを決めるためにオブジェクトが使用するプロパティです（旧リリースでは、「更新」は「リコンフィギュア」、「更新テンプレート」は「リコンフィギュアテンプレート」と呼んでいました）。

コンフリクト

コンフリクトはワークエリアが同期していないか、オブジェクトにパラレル バージョンがあることを示しています。

コンポーネント タスク

コンポーネント タスクは、ベースラインからプロジェクトまたは製品を集めるタスクです。このタスクは常に *component_task* 状態にあります。

コンポーネント名

コンポーネント名はリリースの省略可能部分です。コンポーネント名はアプリケーションまたはコンポーネントの名前を示します。たとえば、**Synergy** または **editor** などがあります。

コンポーネント リリース

コンポーネント リリースはリリースの一部です。コンポーネント リリースは、アプリケーションまたはコンポーネントの特定のリリースを識別します。

削除（delete）

削除操作は Rational Synergy データベースからオブジェクトを取り除きます。

削除（remove）

削除操作はオブジェクトをディレクトリやプロジェクトから取り除きますがデータベースからは削除しません。

差分
差分は、比較またはマージされた 2 つのファイルの差異です。2 つのファイルには 1 つ以上の差分があることがあります。

正規表現
正規表現は、一致テキストを検索するために使用するパターンを定義する文字列です。

製品
製品は、他のファイルを処理することによってビルドされるファイルです。製品の例としては、**.class** ファイル、**.jar** ファイル、**.exe** ファイルなどがあります。

製品タスク
製品を管理するために自動的に Rational Synergy が作成するタスク。

増分ベースライン
プロジェクトのサブセットの単位で直前のベー スラインと差異を持つベースライン。たとえば、**proj1** から **proj100** までのプロジェクトがあり、**proj1-int** と **proj2-int** で新しいベースラインを作成したい場合（**proj3** から **proj100** までに変更がない場合など)、最新のベースラインとこれら 2 つのプロジェクトから新しいベースラインを作ることができます。

タイプ
オブジェクトに含まれるデータのクラスです。タイプはオブジェクトの振る舞いまたは特性を定義します。タイプの例は java、ライブラリ、実行形式ファイル、HTML です。

タスク
タスクはユーザーに割り当てられた todo リスト アイテムです。タスクは、完了するために修正されたファイルも追跡します。

タスクの完了
タスク上のすべての作業を完了したときにタスクを完了できます。これでチェックインすべきすべてのオブジェクトがタスクに割り当てられ、ビルド マネージャが製品をビルドできるようになります。

タスクベースの方法論
タスクベース方法論を使用すると、開発組織は作業の基本単位として個々のファイルではなくタスクを使用してソフトウェア アプリケーションへの変更を追跡できます。

チェックアウト	チェックアウト操作は、それをチェックアウトしたユーザーが編集できるファイルの新しいバージョンを作成します。
チェックイン	チェックイン操作は 1 つ以上のファイルを保存し、他のユーザーが使用できるようにします。
ディレクトリ	Rational Synergy ディレクトリはどのファイルが属するかを維持管理します。
ディレクトリ エントリ	ディレクトリに属する各ファイルについて、ディレクトリはディレクトリ エントリと呼ばれるプレース ホルダを持っています。ディレクトリ エントリは属するファイルを識別しますが、ファイルのバージョンは識別しません。
テスト ベースライン	テスト ベースラインはまだまだ全員に対して使用可能にはなっていないベースラインです。ビルドが SQE のテストに合格したら、開発者が使用するためにベースラインを公開できます。
データベース	Rational Synergy データベースはソース、データファイル、それらのプロパティ、他への関連ファイルを含む使用中の管理データのすべてを格納するデータ レポジトリです。
同期	同期操作により 1 つまたは複数のプロジェクト用のワークエリアが作成され、ワークエリアのファイルとデータベースが比較されます。これにより、ワークエリアとデータベースとの差分をリコンサイルできるようになります。
トラディショナル モード	トラディショナル モードは、RFC を使用する Classic アーキテクチャです。
バージョン	バージョンはファイル、ディレクトリ、プロジェクトの特定のバリエーションです。
パラレル コンフリクト	パラレル コンフリクトは 1 つ以上のパラレル バージョンがチェックアウトされているが、マージされていないときに発生します。

パラレル バージョン
パラレル バージョンは、1 つのファイルから 2 つ以上のバージョンがチェックアウトされたときに発生します。

汎用プロセス ルール
プロセス ルールは、特定の目的のプロジェクトが更新時に新しいメンバーを選択する方法を指定するものです。これは、システムが開始点として使用するベースラインを検索する方法と新しいメンバーを検索するためにシステムが使用すべきタスクを指定して行います。汎用プロセス ルールは、リリースの一部ではないプロセス ルールです。

比較
比較操作では 2 つのオブジェクトの差分が表示されます。

ビルド
ビルドは、makefile でターゲット向けのコマンドを実行することです。ライブラリ、実行形式ファイル、再配置可能なオブジェクトなどの製品を作成します。

ビルド マネージャ
ビルド マネージャは開発チームによる変更を集め、ビルドするユーザーです。

ファイル
ファイルは、データや情報の集まりです。

フォルダ
フォルダは、名前を付けたタスク グループのことです。

ブレッドクラム
ブレッドクラムは、ドキュメントのどの部分が表示されているかを追跡する技術です。ブレッドクラムは、通常ウェブ ページ上部でタイトルバーまたはヘッダーの下に水平方向に表示されます。たどった経路が記録されるので、読み始めた箇所まで遡ることができます。

プロジェクト
プロジェクトとは、ファイルやディレクトリの選択バージョンを特定の構造に配列した論理グループです。

プロジェクト グルーピング
Rational Synergy では、プロジェクトはその目的とリリースによって、たとえば **My 3.0 Collaborative Projects** のようにグループ化されます。これをプロジェクト グルーピングといいます。プロジェクト

	グルーピングは、プロジェクトを更新するときに使用するタスクとベースラインを保持します。これにより、プロジェクト グルーピング内の全プロジェクトの更新プロパティの一貫性を維持します。
プロジェクト タスク	プロジェクトを管理するために自動的に Rational Synergy が作成するタスク。
プロジェクトの更新プロパティ	プロジェクトの更新プロパティは、プロジェクトのプロジェクト グルーピング上のベースラインおよびタスクです。
プロジェクトのコピー	プロジェクトのコピー操作により、プロジェクトを個人使用のためにコピーできます。内容を変更するためにはプロジェクトのコピーを作る必要があります。
プロセス	プロセスは、リリースのプロジェクトを更新する方法を定義するプロセス ルールの集まりです。たとえば、リリースは Integration Testing（統合テスト）という目的を含むことがあります。Integration Testing（統合テスト）目的内に、ビルド マネージャは Hotlist Testing、Integration Testing、Resolved CRs という 3 つのプロセス ルールを持つことがあります。Integration Testing（統合テスト）目的は柔軟です。これは、目的がどのプロセス ルールを持つようビルド マネージャが設定するかによって統合テスト エリアのビルドまたはホットリスト モードのテストの実行に使用できます。開発者はプロセスやプロセス ルールを設定しません。プロジェクトの目的だけを設定します。
プロセス ルール	プロセス ルールは、プロジェクトがどのように更新されるかを定義するパターンを含みます。このパターンは、プロジェクトを更新するとき使用する、ベースラインおよびタスクとフォルダのセットを決定するルールを指定します（旧リリースでは、「更新」は「リコンフィギュア」、「更新テンプレート」は「リコンフィギュア テンプレート」、「プロセス ルール」は「更新テンプレート」と呼んでいました）。

プロパティ

オブジェクトのプロパティ（単にプロパティともいう）から、さまざまな情報を見つけることができます。プロパティの例は名前、バージョン、リリースです。

ベースライン

ある時点における一連のプロジェクトとタスクのセットのスナップショットです。以降の開発の開始点として使用されることがあり、参照のために他のベースラインと比較されることがあります。

ベースライン プロジェクト

自分のプロジェクトのベースにするプロジェクトバージョンを、そのベースライン プロジェクトといいます。たとえば、「**editor-2.0**」プロジェクトのベースライン プロジェクトは、「**editor-1.0**」です。プロジェクトの新規バージョンをチェックアウトすると、そのベースライン プロジェクトが自動的に設定されます。ベースラインは、ベースライン プロジェクトで構成されています。

別バージョンの使用

プロジェクトのファイルまたはディレクトリの別のバージョンを使用できます。別バージョンの使用操作は、単体テスト中に以前のファイル バージョンに戻したいときなどに実行できます。

変更依頼

変更依頼は、Rational Change で作成された変更に対する依頼です。

マージ

マージ機能により、ファイルの 2 つのパラレル バージョンからの情報を組み合わせることができます。2 つのファイルをマージすると、3 つ目のファイルが作成されます。3 つ目のファイルには両ファイルからの情報が含まれます。

マージ コンフリクト

マージ コンフリクトは、2 つの修正されたファイル間で同じ行が違う方法で修正された不整合です。

目的

プロジェクトの目的は、それが何に使用されるかを定義します。たとえば、Insulated Development（個別開発）、Integration Testing（統合テスト）、System Testing（システム テスト）など。プロジェクトの目的を変更すると、Rational Synergy はプロジェクトの更新時に異なる選択基準を使用します。

リコンサイル	**同期** を参照してください。
リリース	リリースは、アプリケーションの特定のリリース固有のプロジェクトまたはタスクを識別するプロパティです。
リリース	リリースはコンポーネント名（オプション）とリリース区切り文字、およびコンポーネント リリースで構成されます。コンポーネント名はアプリケーションまたはコンポーネントの名前を示します。たとえば、**Synergy** または **editor** などがあります。コンポーネント リリースは、アプリケーションまたはコンポーネントの特定のリリースを識別します。**Synergy/7.0** は、リリースの一例です。
履歴	履歴操作はファイル、ディレクトリ、プロジェクトの全バージョン、およびそれぞれの関連を表示します。
ワークエリア	ワークエリアは、プロジェクトによって体系化されたファイルの個人使用コピーを含むファイル システムの場所です。
ワークエリア コンフリクト	ワークエリア コンフリクトは、使用しているワークエリアとデータベースの間の不整合です。

索引

な

は

ひ

ふ

へ

ほ

ま