

Rational™ ClearCase® Rational™ ClearCase® LT

プロジェクト管理ガイド

バージョン: 2003.06.10 およびそれ以降

G126-5399-00

UNIX/WINDOWS 版

法的通知

Copyright © 1992-2003, Rational Software Corporation. All Rights Reserved.

バージョン番号: 2003.06.10 およびそれ以降

本マニュアル (「本著作物」) は、アメリカ合衆国その他の国々の著作権法及び種々の条約により保護されています。Rational Software Corporation の文書による事前の同意を得ることなく本著作物を複製し又は頒布することは、禁じられています。

本著作物はライセンスに基づいて提供されるもので、ライセンス規定に従う場合にのみ、使用または複製できます。ライセンス契約で明示的に許可されている場合を除き、本著作物または本著作物の複製を第三者に提供することは禁じられています。本著作物の権利または所有権を譲渡することはできません。ライセンス条項の全文については、ライセンス契約書をお読みください。

Rational Software Corporation、Rational、Rational Suite、Rational Suite ContentStudio、Rational Apex、Rational Process Workbench、Rational Rose、Rational Summit、Rational Unified process、Rational Visual Test、AnalystStudio、ClearCase、ClearCase Attache、ClearCase MultiSite、ClearDDTS、ClearGuide、ClearQuest、PerformanceStudio、PureCoverage、Purify、Quantify、Requisite、RequisitePro、RUP、SiteCheck、SiteLoad、SoDa、TestFactory、TestFoundation、TestMate、TestStudio は、Rational Software Corporation の米国およびその他の国における登録商標です。Rational のロゴ、Connexis、ObjecTime、Rational Developer Network、RDN、ScriptAssure、XDE は、Rational Software Corporation の米国およびその他の国における商標です。その他すべての名前は、識別の目的でのみ使用されているものであり、それぞれの会社の商標または登録商標です。

米国特許番号 5,193,180、5,335,344、5,535,329、5,574,898、5,649,200、5,675,802、5,754,760、5,835,701、6,049,666、6,126,329、6,167,534、6,206,584 の請求の範囲内の部分。このほかにも米国特許及び国際特許申請中。

米国政府の権利

このソフトウェアおよび文書は、「商業的コンピュータソフトウェア」、「商業的ソフトウェア」または「使用が制限されたコンピュータソフトウェア」として提供され、規約は該当する DFARS 252.227、DFARS 252.211、FAR 2.101、FAR 52.227 (またそれ以前に定められた条項) に規定されています。本ソフトウェア製品およびドキュメントの使用、複製、または開示は、DFARS 227.7202、FAR 52.227-19 の下位条項 (c)、または FAR 52.227-14 (またはその改訂された規定) に定められるように、該当する Rational Software Corporation ライセンス契約書の条項の制約を受けます。

免責事項

本書および関連ソフトウェアは、ライセンス契約に基づいて使用することができます。そのような使用許諾契約書に別段の明示的な規定がある場合を除き、また、それぞれの国の法律により禁止または制限されている場合を除き、Rational Software Corporation は、本メディア、ソフトウェア製品、およびその関連文書について、明示的にも暗黙的にも、商品性に関する保証、非権利侵害性に関する保証、特定目的への適合性に関する保証、取り扱い、使用、または取引行為に伴う保証、およびライセンシーによる静穏無事な製品使用に対する妨害がないことの保証について一切の責任を負いません。

第三者の通知、コード、使用許諾および確認

Portions Copyright © 1992-1999, Summit Software Company. All rights reserved.

Microsoft、Microsoft のロゴ、Active Accessibility、Active Client、Active Desktop、Active Directory、ActiveMovie、Active Platform、ActiveStore、ActiveSync、ActiveX、Ask Maxwell、Authenticode、AutoSum、BackOffice、BackOffice のロゴ、bCentral、BizTalk、Bookshelf、ClearType、CodeView、DataTips、Developer Studio、Direct3D、DirectAnimation、DirectDraw、DirectInput、DirectX、DirectXJ、DoubleSpace、DriveSpace、FrontPage、Funstone、Genuine Microsoft Products のロゴ、IntelliEye、IntelliEye のロゴ、IntelliMirror、IntelliSense、J/Direct、JScript、LineShare、Liquid Motion、Mapbase、MapManager、MapPoint、MapVision、Microsoft Agent のロゴ、Microsoft eMbedded Visual Tools のロゴ、Microsoft Internet Explorer のロゴ、Microsoft Office Compatible のロゴ、Microsoft Press、Microsoft Press のロゴ、Microsoft QuickBasic、MS-DOS、MSDN、NetMeeting、NetShow、Office のロゴ、Outlook、PhotoDraw、PivotChart、PivotTable、PowerPoint、QuickAssembler、QuickShelf、RelayOne、Rushmore、SharePoint、SourceSafe、TipWizard、V-Chat、VideoFlash、Visual Basic、Visual Basic のロゴ、Visual C++、Visual C#、Visual FoxPro、Visual InterDev、Visual J++、Visual SourceSafe、Visual Studio、Visual Studio のロゴ、Vizact、WebBot、WebPIP、Win32、Win32s、Win64、Windows、Windows CE のロゴ、Windows のロゴ、Windows NT、Windows Start のロゴ、XENIX 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。

Sun、Sun Microsystems、Sun のロゴ、Ultra、AnswerBook 2、medialib、OpenBoot、Solaris、Java、Java 3D、ShowMe TV、SunForum、SunVTS、SunFDDI、StarOffice、および SunPCi は、Sun Microsystems の米国および他の国における商標または登録商標です。

Purify は、Sun Microsystems, Inc. の米国特許番号 5,404,499 の下にライセンス供与されています。

Globetrotter ソフトウェア (FLEXIm ライブラリおよびユーティリティ) の本来の用途は、ソフトウェアライセンス管理であり、他の製品またはアプリケーションにこれらのソフトウェアを組み込むことは、ライセンスに含まれません。

BasicScript は、Summit Software Company の登録商標です。

デザイン パターン : Erich Gamma、Richard Helm、Ralph Johnson および John Vlissides による再使用可能なオブジェクト指向のソフトウェアのエLEMENT。Copyright © 1995 by Addison-Wesley Publishing Company, Inc. All rights reserved.

Copyright © 1997 OpenLink Software, Inc. All rights reserved.

本ソフトウェアおよびドキュメントは部分的に、カリフォルニア大学理事会により使用許諾されている BSD Networking Software Release 2 に基づいています。当社はその開発におけるカリフォルニア大学バークレー校の Computer Systems Research Group および Electrical Engineering and Computer Sciences Department 並びに「その他の協力者」の役割を認めます。

本製品は、Apache (http://www.webdav.org/mod_dav/) における使用のために Greg Stein により開発されたソフトウェアを含んでいます。

追加の法的通知は、お客様の Rational ソフトウェア インストレーションに含まれています。

目次

まえがきxxiii
本書について	xxiii
製品固有の機能	xxiii
本書の構成	xxiii
ClearCase マニュアル ロードマップ	xxiv
ClearCase LT マニュアル ロードマップ	xxv
ClearCase とほかの Rational 製品との統合	xxvi
表記規則	xxviii
オンライン マニュアル	xxix
カスタマ サポート	xxx
 UCM かベース ClearCase かの選択	1
UCM とベース ClearCase との違い	1
ブランチとビューの作成	1
コンポーネントを使用したファイルの編成	3
ベースラインの作成と使用	3
アクティビティの管理	4
開発ポリシーの実施	4

UCM での作業

UCM の理解	9
UCM プロセスの概要	9
プロジェクトの作成	12
PVOB の作成	12
ディレクトリとファイルのコンポーネントへの編成	13
共有作業空間と個人用作業空間	14
ストリーム階層	14
単一ストリーム プロジェクト	14
ベースラインからの開始	15
UCM と ClearQuest の統合の設定	17

ポリシーの設定	18
作業の割り当て	18
テスト ストリームの作成	19
コンポーネントのビルド	20
MultiSite についての考慮点	20
新しいベースラインの作成	21
ベースラインの推奨	23
プロジェクト状態の監視	25
UCM と ClearQuest の統合の概要	25
UCM のオブジェクトと ClearQuest のオブジェクトの関連付け	26
UCM に適用可能なスキーマ	27
状態タイプ	27
UCM に適用可能な ClearQuest のスキーマでのクエリー	28

プロジェクトの計画 29

開始点としてのシステム アーキテクチャの使用法	29
コンポーネントへのシステム アーキテクチャのマッピング	29
バージョン管理の対象の決定	30
プロジェクトへのコンポーネントのマッピング	30
統合の度合い	31
並行リリースの必要性	31
例	31
コンポーネントの編成	32
使用する VOB 数の決定	32
追加のコンポーネントの指定	33
ディレクトリ構造の定義	34
読み取り専用コンポーネントの識別	36
ストリーム方針の選択	36
ストリーム階層	37
単一ストリーム プロジェクト	38
読み取り専用ストリーム	39
ベースライン ポリシーの指定	39
プロジェクト ベースラインの指定	40
ベースラインを作成するポイント	41
初期ベースラインの指定	41
後続のベースライン	42

命名規則の定義	43
開発の状態を反映するプロモーション レベルの設定	43
ベースラインのテスト方法の計画	43
PVOB の計画	44
使用する PVOB 数の決定	44
管理 VOB の役割の理解	45
複数の PVOB の使用法	46
特殊なエレメント タイプの指定	48
非マージ エレメント	48
非自動マージ エレメント	48
エレメント タイプの適用範囲の定義	49
UCM と ClearQuest の統合の使用法の計画	49
ClearQuest のユーザー データベースへの PVOB のマッピング	49
MultiSite の要件	49
同じデータベースにリンクされるプロジェクトには一意の名前を付ける	49
リンクしているデータベースへの 1 つのスキーマ リポジトリの適用	50
適用するスキーマの決定	51
UnifiedChangeManagement スキーマの概要	52
UCM へのスキーマの適用	53

ポリシーの設定 55

コンポーネントとベースライン	55
変更可能なコンポーネント	55
ベースラインを推奨する際のデフォルトのプロモーション レベル	55
デフォルトのビュー タイプ	55
プロジェクトとストリームを変更する権限	56
すべてのユーザーにプロジェクトの変更を許可する	56
すべてのユーザーにストリームとストリームのベースラインの変更を許可する	56
デリバー操作	57
未処理のチェックアウトがあるストリームからのデリバーを許可	57
デリバー前のリベース	57
デフォルト以外のターゲットへのデリバー操作	57
別プロジェクトのプロジェクトまたはストリームへのデリバーを許可する	59
ストリームからの変更と基本ベースラインからの変更をデリバーする	59
ターゲット ストリームがソース ストリームに存在するコンポーネントを 含んでいない場合でもデリバーを許可する	60

ソース ストリームの変更可能なコンポーネントがターゲット ストリームでは変更可能でない場合でもデリバーを許可する	61
UCM と ClearQuest の統合	61
作業前に ClearQuest のアクションを実行	61
デリバー前に ClearQuest のアクションを実行	61
アクティビティ変更前に ClearQuest のアクションを実行	62
デリバー後に ClearQuest のアクションを実行	62
アクティビティ変更後に ClearQuest のアクションを実行	62
デリバー後に完了に遷移	63
アクティビティ変更後に完了に遷移	63
デリバー前に ClearQuest のマスタースhipを転送	64
デリバー後に ClearQuest のマスタースhipを転送	65
UCM と ClearQuest の統合におけるプロジェクト間デリバーの処理	65
ClearQuest ユーザー データベースの設定	67
定義済みの UCM に適用可能なスキーマの使用	67
UCM に適用するためのスキーマの設定	68
カスタム レコード タイプを有効にするための要件	70
状態タイプの設定	71
レコード タイプに関する状態遷移のデフォルト アクションの要件	72
スキーマの最新 UCM パッケージへのアップグレード	73
ClearQuest のプロジェクト ポリシーのカスタマイズ	74
親アクティビティ レコードと子アクティビティ レコードの関連	74
親/子コントロールの使用	75
ユーザーの作成	75
環境の設定 (UNIX)	75
プロジェクトのセットアップ	77
プロジェクトの新規作成	78
プロジェクト VOB の作成 (Windows)	78
プロジェクト VOB の作成 (UNIX)	79
プロジェクト ベースラインを保存するコンポーネントの作成	80
エレメントを保存するコンポーネントの作成	81
複数のコンポーネントを保存する VOB の作成 (Windows)	81
複数のコンポーネントを保存する VOB の作成 (UNIX)	82
VOB あたり 1 つのコンポーネントの作成 (Windows)	83
VOB あたり 1 つのコンポーネントの作成 (UNIX)	84

プロジェクトの作成	85
ベースライン命名テンプレートの設定	86
プロモーション レベルの定義	87
インテグレーション ビューの作成	88
プロジェクトを示す複合ベースラインの作成	89
アクティビティの作成と設定 (UNIX のみ)	90
ディレクトリ構造の作成	91
Windows 環境の場合	91
UNIX 環境の場合	91
ClearCase 外部からのディレクトリとファイルのインポート	92
ベースラインの作成と推奨	93
既存の ClearCase 構成に基づいたプロジェクトの作成	93
PVOB の作成	93
VOB からコンポーネントへの変換	93
ラベルからのベースラインの作成	94
プロジェクトの作成	95
インテグレーション ビューの作成	95
既存のプロジェクトに基づいたプロジェクトの作成	95
複合ベースラインを使用した最終ベースラインの取得	96
既存の PVOB とコンポーネントの再使用	96
プロジェクトの作成	96
インテグレーション ビューの作成	97
プロジェクトで UCM と ClearQuest の統合を使用可能にする	98
アクティビティの移行	98
プロジェクト ポリシーの設定	99
アクティビティの割り当て	100
プロジェクトと ClearQuest ユーザー データベース間のリンクの無効化	101
リンクされているアクティビティとリンク解除されているアクティビティが 混在するプロジェクトの修正	101
問題の検出	102
問題の修正	102
UCM と ClearQuest の統合に対する MultiSite の影響	103
レプリカと命名に関する要件	103
プロジェクトのマスターシップの転送	103

ClearQuest レコードへのアクティビティのリンク	103
プロジェクト ポリシー設定の変更	104
プロジェクト名の変更	104
Rational Suite を使用した作業 (Windows)	104
ベースライン テスト用開発ストリームの作成	105
機能別開発ストリームの作成	106

プロジェクトの管理 109

コンポーネントの追加	109
コンポーネントを変更可能にする	110
ビューの同期	111
子ストリームの同期	111
スナップショット ビューのロード規則の更新	111
コンポーネントのビルド	112
インテグレーション ストリームのロック	112
デリバーできる作業結果の検索	113
リモート デリバー操作の完了	114
デリバー操作の取り消し	114
コンポーネントのビルドとテスト	114
新規ベースラインの作成	115
新規ベースラインの作成	115
複数のアクティビティを取り込んだベースラインの作成	117
1 つのコンポーネントのベースラインの作成	117
ストリームのアンロック	117
ベースラインのテスト	117
問題の解決	118
ベースラインの推奨	119
ベースラインの競合の解決	120
複合ベースラインと非複合ベースラインの競合	121
複合ベースライン間の競合	121
プロジェクト状態の監視	121
ベースライン履歴の表示 (Windows のみ)	121
ベースラインの比較	123
ClearQuest ユーザー データベースのクエリー	124
ClearCase レポートの使用 (Windows のみ)	126

プロジェクトのクリーンアップ	127
未使用のオブジェクトの削除	127
プロジェクト	128
ストリーム	128
コンポーネント	128
ベースライン	128
アクティビティ	129
プロジェクトとストリームのロックと不要化	129
トリガを使用した開発ポリシーの実施	131
トリガの概要	131
操作前トリガと操作後トリガ	132
トリガの範囲	132
トリガでの属性の使用法	132
UCM トリガの代わりに ClearQuest スクリプトを使用する場合	133
UNIX と Windows 間でのトリガの共有	134
異なるパス名または異なるスクリプトの使用	134
同一スクリプトの使用	134
ヒント	135
順次デリバー操作の実施	135
セットアップ スクリプト	135
操作前トリガ スクリプト	137
操作後トリガ スクリプト	138
開発者へのデリバー操作に関するメールの送信	139
セットアップ スクリプト	139
操作後トリガ スクリプト	140
インテグレーション ストリームでのアクティビティの作成禁止	141
役割に基づいたアクセス制御システムの実装	142
操作前トリガ スクリプト	143
UCM トリガのその他の使用法	145
複数プロジェクトの並行リリースの管理	147
現在のプロジェクトと次回プロジェクトの同時管理	147
例	147
プロジェクト間のリベース操作の実行	149

プロジェクト新規バージョンへのパッチ リリースの組み込み	150
例	150
ほかのプロジェクトへの作業のデリバー	152
メインライン プロジェクトの使用法	153
プロジェクトから非 UCM ブランチへのマージ	153

ベース ClearCase での作業

ベース ClearCase でのプロジェクト管理	157
プロジェクトの設定	158
VOB の作成と入力	158
ブランチ作成ポリシーの計画	158
ブランチ名	159
ブランチと ClearCase MultiSite	159
共有ビューと標準構成仕様の作成	160
推奨されるビュー名	160
開発ポリシーの実装	161
ラベルの使用法	161
属性、ハイパーリンク、トリガ、ロックの使用法	161
グローバル タイプ	162
レポートの作成	162
変更の統合	163
プロジェクト ビューの定義	165
構成仕様の働き	165
デフォルト構成仕様	166
標準構成規則	166
標準構成規則の省略	167
構成仕様のインクルード ファイル	167
構成仕様のサンプルでのプロジェクト環境	168
プロジェクト開発用のビュー	170
ブランチ上の新規開発用のビュー	170
時間規則を使用するバリエーション	170

古い構成を変更するビュー	170
/main/LATEST 規則の省略	172
時間規則を使用するバリエーション	172
複数レベルのブランチ作成を実装するビュー	172
単一ディレクトリに変更を制限するビュー	173
プロジェクト状態を監視するためのビュー	174
バージョンを選択するための属性を使用するビュー	174
この構成を開発で使用する場合の注意点	175
1 人の開発者による変更を表示するビュー	177
バージョン ラベルによって定義される履歴ビュー	177
時間規則によって定義される履歴ビュー	178
プロジェクト ビルド用のビュー	178
夜間ビルドの結果を使用するビュー	178
プロジェクト ライブラリのバージョンを選択するバリエーション	179
アプリケーション サブシステムのバージョンを選択するビュー	180
特定のプログラムをビルドしたバージョンを選択するビュー	180
makefile の構成	181
プログラム内のバグの修正	181
一連のプログラムをビルドしたバージョンの選択	182
UNIX と Windows 間での構成仕様の共有	182
パス名区切り文字	183
構成仕様のエレメント規則内のパス名	183
構成仕様のコンパイル	184
例	184

プロジェクト開発ポリシーの実装 185

変更についての十分な説明	185
すべてのソース ファイルへのプログレス インジケータの添付	186
主要構成に使用されているすべてのバージョンへのラベルの関連付け	188
リリース バグの作業のブランチ上への隔離	188
ほかの開発者による作業の混乱の回避	189
必要に応じたプロジェクト データへのアクセスの拒否	190
関連する変更のチーム メンバーへの通知	190
すべてのソース ファイルのプロジェクト標準への準拠	192
変更と変更順序の関連付け	193
プロジェクト要求とソース ファイルの関連付け	194

特定コマンドの使用の禁止	196
特定ブランチの MultiSite のサイト間での共有	197
UNIX と Windows 間でのトリガの共有	198
異なるパス名または異なるスクリプトの使用	198
同スクリプトの使用	198
メモ	199
ベース ClearCase と ClearQuest の統合の設定	201
統合の概要	201
ClearQuest と ClearCase の設定	202
ClearQuest スキーマへの ClearCase 定義の追加	202
ClearCase VOB へのトリガのインストール	203
評価用のクイック スタート (V2 トリガのみ)	204
ClearQuest Web インターフェイスの環境変数の設定	204
ClearQuest Perl API の環境変数の設定	205
構成ファイルの編集 (V2 トリガのみ)	205
統合のテスト (V2 トリガのみ)	205
パフォーマンスのチェック (V2 トリガのみ)	206
統合クエリー ウィザードの使用法	206
変更の統合	207
マージの仕組み	207
GUI を使用したエレメントのマージ	209
コマンド行を使用したエレメントのマージ	210
一般的なマージのシナリオ	211
シナリオ: サブブランチからの選択マージ	211
シナリオ: 特定バージョンからの変更内容の削除	212
シナリオ: すべてのプロジェクト作業のマージ	213
すべてのプロジェクト作業のブランチ上への隔離	213
すべてのプロジェクト作業のビュー内への隔離	213
シナリオ: ソース ツリー全体の新規リリースのマージ	213
シナリオ: ディレクトリ バージョンのマージ	216
独自のマージ ツールの使用法	217

エレメント タイプを使用したファイル エレメントの処理のカスタマイズ ..	219
一般的なプロジェクトでのファイル タイプ	219
ClearCase によるエレメント タイプの割り当て方法	220
エレメント タイプとタイプ マネージャ	221
エレメント タイプのその他の応用	222
エレメント タイプを使用したビューの構成	222
エレメント タイプごとのファイル処理	222
定義済みエレメント タイプとユーザー定義のエレメント タイプ	223
定義済みタイプ マネージャとユーザー定義のタイプ マネージャ	223
UNIX: 新規タイプ マネージャの作成	224
UNIX: タイプ マネージャ プログラムの作成	224
メソッドの終了状態	225
マニュアル ページのソース ファイルのタイプ マネージャ	225
タイプ マネージャ ディレクトリの作成	225
別のタイプ マネージャからのメソッドの継承	226
create_version メソッド	226
construct_version メソッド	228
新規 compare メソッドの実装	229
タイプ マネージャのテスト	231
タイプ マネージャのインストールと使用法	231
GUI ブラウザによるアイコンの使用法	233
 開発サイクル全体を通じた ClearCase の使用法	 235
プロジェクトの概要	235
開発方針	237
プロジェクト マネージャと ClearCase 管理者	237
ブランチの使用法	237
プロジェクト ビューの作成	239
ブランチ タイプの作成	240
標準構成仕様の作成	240
ビューの作成、構成、登録	241
開発の開始	241
各自の作業を隔離するテクニック	242

ベースライン 1 の作成	242
2 つのブランチのマージ	243
統合とテスト	243
ソースへのラベルの関連付け	243
インテグレーション ビューの削除	244
進行中の開発作業のマージ	244
マージの準備	245
作業のマージ	247
ベースライン 2 の作成	248
r1_fix ブランチからのマージ	249
major ブランチからのマージの準備	249
major ブランチからのマージ	251
major ブランチの破棄	252
統合とテスト	252
最終確認: リリース 2.0 の作成	252
ソースへのラベルの関連付け	253
main ブランチの使用制限	253
テスト ビューのセットアップ	253
バグ修正を監視するトリガの設定	254
最終バグの修正	254
ラベルからの再ビルド	255
まとめ	255
ビュー プロファイルから UCM への移行	257
ビュー プロファイルと UCM	257
機能の比較	257
ブランチとストリーム	257
ブランチ間またはストリーム間での作業の移動	258
VOB とコンポーネント	258
チェックポイントとベースライン	258
ビュー プロファイル情報を UCM に移行する方法	259
ビュー プロファイル プロジェクトの準備	259
ビュー プロファイル情報の移行	259

ClearCase と ClearQuest の統合	261
ClearCase と ClearQuest の統合の理解	261
統合の共存の管理	262
スキーマ	262
画面表示	262
ClearCase レポートのカスタマイズ	265
ClearCase レポートの仕組み	265
ClearCase レポートのカスタマイズ可能な内容	266
レポート プログラミング インターフェイスの実行時の処理順序	268
共有レポート ディレクトリの構成	270
ソース管理へのレポート プロシージャの追加	271
カスタマイズしたディレクトリに対する Report Builder の設定	271
ClearCase レポートのデフォルトのディレクトリ構造	271
Report Builder のツリー ペインへのデータ指定	272
レポート プロシージャのインターフェイス仕様	273
All_Views.prl のインターフェイス仕様	274
description 仕様	275
ヘルプ ID 仕様	275
parameters 仕様	275
rightclick 仕様	278
fields 仕様	278
field_type 規則	279
パラメータ セレクタ	281
パス セレクタ	281
UCM ターゲット セレクタ	281
タイプ セレクタ	281
日付/時刻セレクタ	281
テキスト セレクタ	282
レポートの表示	282
レポート データの保存	283

レポート プログラミングの例	284
例 1: レポート出力への列の追加	284
処理ロジック	285
インターフェイス仕様	285
必要な変更	286
変更後のレポート プロシージャ	286
例 2: レポートのディレクトリ構造、説明、出力の変更	288
処理ロジック	289
インターフェイス仕様	289
必要な変更	290
変更後のレポート プロシージャ	290
例 3: レポート説明、パラメータ タイプ、レポート出力の変更	293
処理ロジック	293
インターフェイス仕様	294
必要な変更	294
変更後のレポート プロシージャ	295
例 4: 右クリック操作のショートカット メニューの変更	297
インターフェイス仕様	298
必要な変更	298
変更後のレポート プロシージャ	299
例 5: Report Viewer のショートカット メニューへの新しいコマンドの追加	301
インターフェイス仕様	302
必要な変更	302
変更後のレポート プロシージャ	302
トラブルシューティング	306
インターフェイス仕様内のエラー	306
ccperl 以外の高水準言語でのコーディング	309

索引	311
-----------------	------------

図目次

図 1	ベース ClearCase のブランチ階層	2
図 2	プロジェクト マネージャー、開発者、統合担当者のワーク フロー	11
図 3	複数のコンポーネントがある VOB	13
図 4	2 つのコンポーネントのベースライン	16
図 5	複合ベースライン	17
図 6	リベース操作	22
図 7	ベースラインのプロモート	24
図 8	UCM と ClearQuest の統合での UCM のオブジェクトと ClearQuest のオブジェクトの関連付け	26
図 9	Transaction Builder プロジェクトで使用するコンポーネント	32
図 10	VOB への複数コンポーネントの保存	33
図 11	読み取り専用コンポーネントの使用	36
図 12	機能固有の開発ストリームの使用	37
図 13	システム レベルの複合ベースラインの使用	41
図 14	関連プロジェクトでの PVOB の共有	45
図 15	ある 1 つの PVOB を複数の PVOB の管理 VOB として使用する	47
図 16	同一の ClearQuest データベースにリンクされた複数の PVOB 内のプロジェクト	50
図 17	複数の ClearQuest データベースへの同スキーマ リポジトリの適用	51
図 18	UCM に適用可能なレコード タイプ用のレコード フォームの [Unified Change Management] タブ	52
図 19	BaseCMAActivity レコード タイプ用のレコード フォームの [メイン] タブ	53
図 20	ストリーム階層におけるデフォルトとデフォルト以外のデリバリー ターゲット	58
図 21	基本ベースラインで行われた変更のデリバリー	60
図 22	UCM に適用可能なスキーマとのユーザー データベースの関連付け	68
図 23	レコード タイプの状態への状態タイプの割り当て	69
図 24	レコード タイプの状態遷移マトリックスへの移動	70
図 25	UCM に適用可能な BaseCMAActivity レコード タイプの状態遷移図	72
図 26	プロジェクト エクスプローラでのインテグレーション ストリームへの移動	88
図 27	[ベースライン依存関係の編集] ダイアログ ボックスの使用法	90
図 28	新規プロジェクト ウィザードのステップ 2	97
図 29	プロジェクトで ClearQuest ユーザー データベースを使用可能にする	99
図 30	UCMProjects クエリーへの移動	100
図 31	[ベースラインの追加] ダイアログ ボックス	110
図 32	[ベースラインの作成] ダイアログ ボックス	116
図 33	ClearCase コンポーネント ツリー ブラウザ	122
図 34	ベースラインの比較	123

図 35	アクティビティごとのベースラインの比較	124
図 36	ClearCase Report Builder	127
図 37	次回リリースの管理	148
図 38	パッチ リリースの組み込み	151
図 39	古いバージョンの変更	171
図 40	複数レベルの自動ブランチ作成	173
図 41	開発の構成仕様と QA の構成仕様	175
図 42	エレメントのブランチのチェックアウト	176
図 43	要求追跡	195
図 44	一般的なマージに関連するバージョン	208
図 45	ClearCase のマージ アルゴリズム	209
図 46	サブブランチからの選択マージ	211
図 47	特定バージョンからの変更内容の削除	212
図 48	ソース ツリー全体の新規リリースのマージ	214
図 49	ユーザー定義アイコンの表示	234
図 50	リリース 2.0 開発のプロジェクト計画	236
図 51	開発のマイルストーン: 一般的なエレメントの進展	239
図 52	ベースライン 1 の作成	242
図 53	新機能の開発の更新	245
図 54	ベースライン 1 の変更の major ブランチへのマージ	247
図 55	ベースライン 2	249
図 56	ベースライン 2 に先行するマージ後のエレメント構造	251
図 57	最終テストとリリース	252
図 58	ClearQuest GUI 内の変更セット	263
図 59	Report Builder インターフェイスのカスタマイズ可能な領域	266
図 60	Report Viewer ウィンドウ内のカスタマイズ可能なインターフェイス	267
図 61	実行時の処理順序	269
図 62	Report Builder のユーザー インターフェイス	272
図 63	Report Viewer ウィンドウ	283
図 64	無効なパラメータのある Report Builder ウィンドウ	308

表目次

表 1	コンポーネントの推奨ディレクトリ構造	34
表 2	UCM に適用可能なスキーマの状態タイプ	71
表 3	統合に必要な環境変数	76
表 4	UCM に適用可能なスキーマ内でのクエリー	125
表 5	一般的なプロジェクトで使用されるファイル	219
表 6	ビュー プロファイルと UCM のさまざまな機能の比較	258
表 7	ClearCase レポートのパラメータ	276
表 8	フィールド変更子	278
表 9	ClearCase レポートで指定可能なフィールド タイプ	279

まえがき

構成管理システムである Rational™ ClearCase® は、ソフトウェアのビルドに使用されたオブジェクトをソフトウェア開発チームが追跡することを支援するため設計されています。ベース ClearCase を使用して、構成管理環境をカスタマイズすることができます。また、統一変更管理 (UCM) プロセスを利用することもできます。

本書について

本書では、プロジェクト マネージャー向けに、UCM を使用して、またはベース ClearCase をカスタマイズして、開発チーム用に構成管理環境を設定して管理する方法を説明します。

製品固有の機能

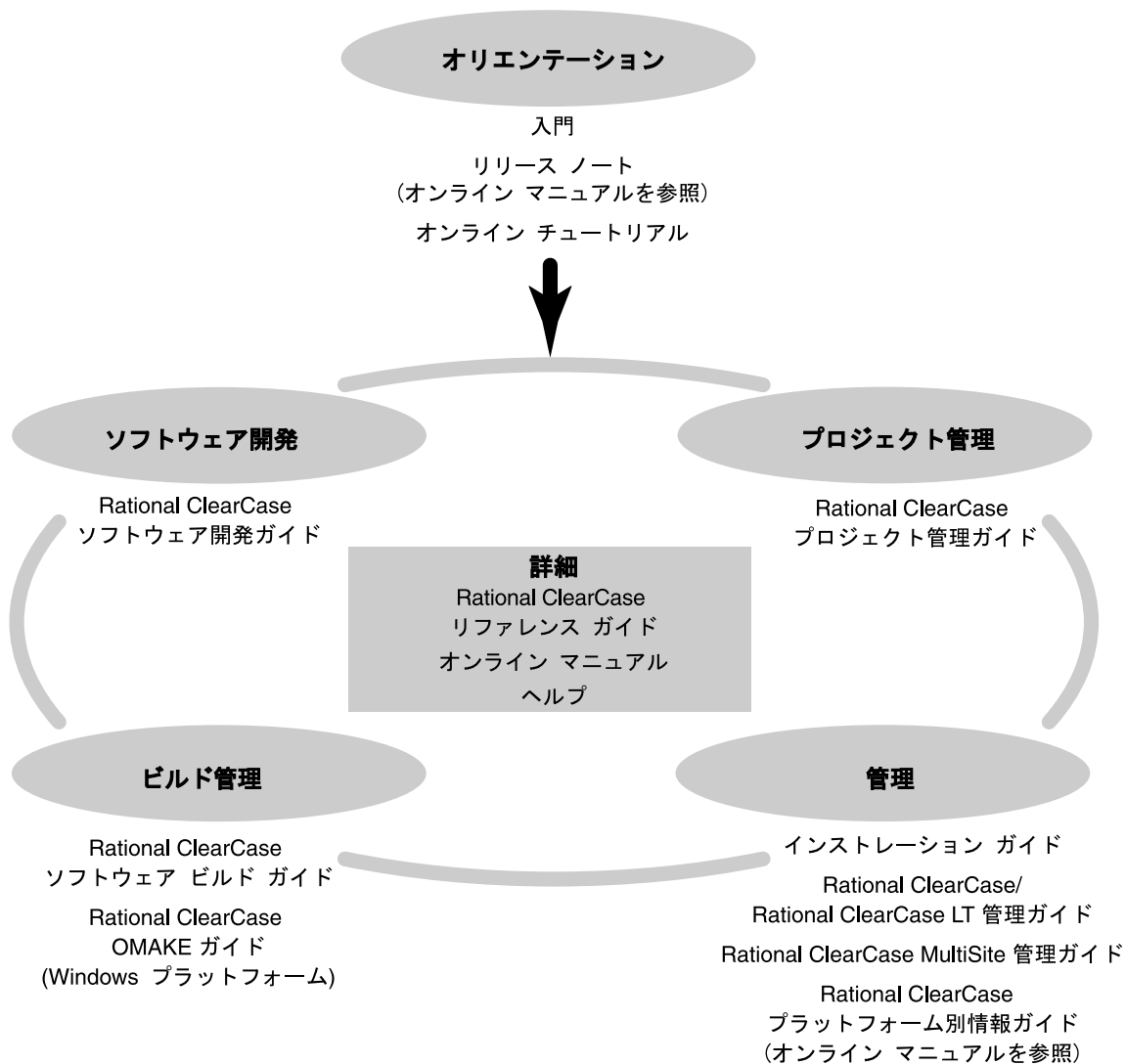
本書では、Rational ClearCase と Rational ClearCase LT について説明します。ClearCase LT には、ClearCase のすべての機能が含まれているわけではありません。また、2 つの製品では一部のユーザー インターフェイスが異なります。本書では、「製品メモ」というラベルを使用して、この違いを説明します。「製品メモ」の項以外で ClearCase という用語を使用する場合、それは両製品を意味します。

本書の構成

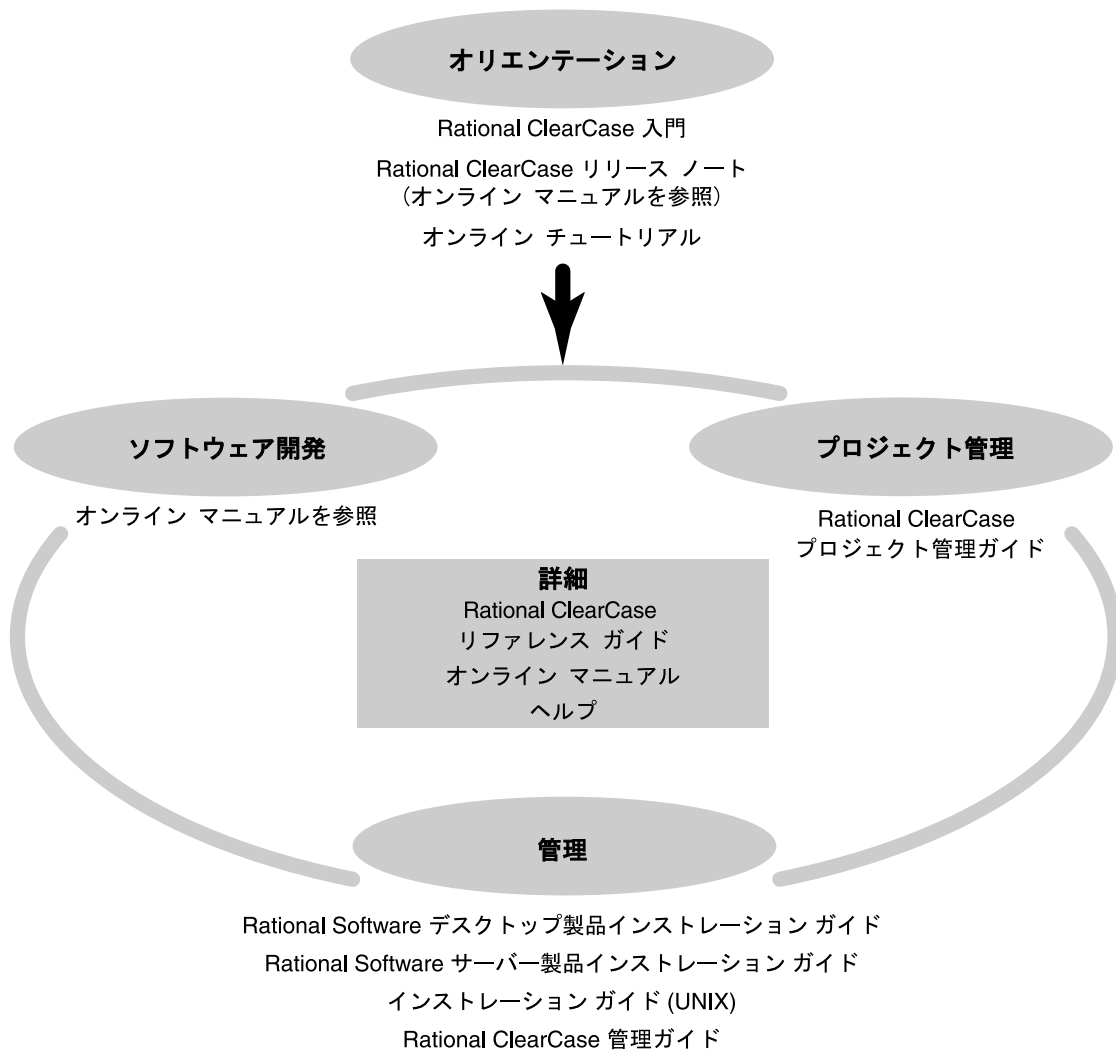
本書は以下の 2 部から構成されています。

- 第 1 部: UCM での作業 - UCM を使用してチームの開発プロセスの実装を計画している場合にお読みください。
- 第 2 部: ベース ClearCase での作業 - ベース ClearCase の機能を使用してチームの開発プロセスのカスタマイズを計画している場合にお読みください。

ClearCase マニュアル ロードマップ



ClearCase LT マニュアル ロードマップ



ClearCase とほかの Rational 製品との統合

統合	説明	説明が記載されている場所
ベース ClearCase - ClearQuest®	変更依頼を ClearCase エlement のバージョンに関連付けます。	『Rational ClearCase ソフトウェア開発ガイド』 『Rational ClearCase プロジェクト管理ガイド』 『Rational ClearQuest 管理ガイド』
ベース ClearCase - Apex	Apex 開発者がファイルを ClearCase に保存できるようにします。	『Installing Rational Apex (UNIX)』
ベース ClearCase - ClearDDTS	変更依頼を ClearCase エlement のバージョンに関連付けます。	『ClearCase ClearDDTS Integration』
ベース ClearCase - PurifyPlus	開発者が、PurifyPlus から ClearCase を呼び出せるようにします。	PurifyPlus のヘルプ
ベース ClearCase - RequisitePro	RequisitePro プロジェクトを ClearCase 内にアーカイブします。	『Rational RequisitePro ユーザーズガイド』 RequisitePro のヘルプ
ベース ClearCase - Rose	Rose モデルを ClearCase に保存します。	Rose のヘルプ
ベース ClearCase - Rose RealTime	Rose RealTime モデルを ClearCase に保存します。	『Rose RealTime Toolset Guide』 『Rose RealTime Guide to Team Development』
ベース ClearCase - SoDA	ClearCase から情報を収集し、その情報をさまざまなレポート形式で表示します。	『Using Rational SoDA for Word』 『Using Rational SoDA for Frame』 SoDA のヘルプ
ベース ClearCase - XDE™	XDE モデルを ClearCase に保存します。	XDE のヘルプ
UCM - ClearQuest	UCM アクティビティを ClearQuest レコードにリンクします。	『Rational ClearCase ソフトウェア開発ガイド』 『Rational ClearCase プロジェクト管理ガイド』 『Rational ClearQuest 管理ガイド』
UCM - PurifyPlus	開発者が PurifyPlus から ClearCase を呼び出せるようにします。	PurifyPlus のヘルプ

統合	説明	説明が記載されている場所
UCM - RequisitePro	RequisitePro 管理者が UCM で RequisitePro プロジェクトのベースラインを作成できるようにすると共に、ベースラインから RequisitePro プロジェクトを作成できるようにします。	『Rational RequisitePro ユーザーズガイド』 RequisitePro のヘルプ 『Rational Suite® 統一変更管理 (UCM) ユーザーズガイド』
UCM - Rose	Rose モデルを ClearCase に保存します。	Rose のヘルプ 『Rational Suite 統一変更管理 (UCM) ユーザーズガイド』
UCM - Rose RealTime	アクティビティをリビジョンに関連付けます。	『Rose RealTime Toolset Guide』 『Rose RealTime Guide to Team Development』
UCM - SoDA	ClearCase から情報を収集し、その情報をさまざまなレポート形式で表示します。	『Using Rational SoDA for Word』 『Using Rational SoDA for Frame』 SoDA のヘルプ
UCM - TestManager	テストアセットを ClearCase に保存します。	『Rational TestManager User's Guide』 TestManager のヘルプ 『Rational Suite 統一変更管理 (UCM) ユーザーズガイド』
UCM - XDE	XDE モデルを ClearCase に保存します。	XDE のヘルプ
UCM - XDE Tester	XDE Tester データストアを ClearCase に保存します。	XDE Tester のヘルプ

表記規則

本書の表記規則は次のとおりです。

- `ccase-home-dir` は、ClearCase 製品ファミリーがインストールされているディレクトリを表します。デフォルトのインストール ディレクトリは、UNIX では `/opt/rational/clearcase`、Windows では `C:\Program Files\Rational\ClearCase` です。
- `cquest-home-dir` は、Rational ClearQuest がインストールされているディレクトリを表します。デフォルトのインストール ディレクトリは、UNIX では `/opt/rational/clearquest`、Windows では `C:\Program Files\Rational\ClearQuest` です。
- **太字**は、コマンド名やブランチ名など、ユーザーが入力可能な名前に使用します。
- **sans serif フォント**は、ファイル名、ディレクトリ名、ファイル拡張子に使用します。
- メニュー名やチェック ボックス名のような、GUI 要素は、`[]` で囲んで表記します。
- **等幅フォント**は、例に使用します。ユーザー入力とプログラム出力を区別する必要がある場合、ユーザー入力には**太字**を使用します。
- 出力されない文字は、`<EOF>`、`<NL>` のように表示します。
- キー名やキーの組み合わせは、大文字で `SHIFT`、`CTRL+G` のように表示します。
- `[]` 大カッコは、書式や構文の記述でオプション項目を囲むために使用します。
- `{ }` 中カッコは、書式や構文の記述で選択項目のリストを囲むために使用します。
- `|` 縦棒は、選択項目のリストを区切るために使用します。
- ... 構文記述内の省略記号は、省略記号に先行する項目や行を 1 回以上繰り返せることを示します。それ以外の省略記号は、情報の省略を示します。

メモ: あるコンテキストでは、`"*` や `"?"` と同様に、`"..."` をパス名内でワイルドカードとして使用することができます。詳細については、`wildcards_ccase` のリファレンス ページを参照してください。

- コマンド名またはオプション名に省略形がある場合、`"中点" (·)` は最短の有効な省略形を示します。たとえば次のような機能があります。

`lsc:checkout`

オンライン マニュアル

ClearCase 製品ファミリー (CPF) には、次のオンライン マニュアルが含まれています。

ヘルプ システム: [ヘルプ] メニュー、[ヘルプ] ボタン、[F1] キーを使用します。一連のオンライン マニュアルの内容を表示するには、次のいずれかを行います。

- UNIX の場合は、「cleartool man <コマンド名>」と入力します。
- Windows の場合は、[スタート]、[プログラム]、[Rational Software]、[Rational ClearCase] をポイントし、[ヘルプ] をクリックします。
- いずれのプラットフォームの場合でも、Rational ClearCase MultiSite の内容を表示するには、「multitool man <コマンド名>」と入力します。
- ダイアログ ボックスに関する情報を表示するには、ダイアログ ボックスの [ヘルプ] ボタンをクリックするか、[F1] を押します。

リファレンス ページ: cleartool man コマンドと multitool man コマンドを使用します。

詳細については、man のリファレンス ページを参照してください。

コマンド構文: -help コマンド オプションまたは cleartool help コマンドを使用します。

チュートリアル: 製品の重要な機能の段階的な説明を表示します。チュートリアルを開始するには、次のいずれかを行います。

- UNIX の場合は、「cleartool man tutorial」 と入力します。
- Windows の場合は、[スタート]、[プログラム]、[Rational Software]、[Rational ClearCase] をポイントし、[ClearCase チュートリアル] をクリックします。

PDF マニュアル: 次のディレクトリに移動します。

- UNIX の場合は、ccase-home-dir/doc/books
- Windows の場合は、ccase-home-dir¥doc¥books

カスタマ サポート

ソフトウェアやマニュアルの問題については、電話、ファックス、または電子メールで以下の Rational カスタマ サポートまでお問い合わせください。サポートの時間帯、対応言語、その他のサポート情報については、日本ラショナル ソフトウェア社の Web サイトの **サポート情報** (<http://www.rational.co.jp/supports/>) を参照してください。

地域	電話	Fax	電子メール
アジア太平洋 (日本を含む)	+61-2-9419-0111	+61-2-9419-0123	support@apac.rational.com (英語のみ対応) support@japan.rational.com (日本語対応可)

UCM かベース ClearCase かの選択

1

ClearCase を使用して開発プロジェクトのバージョン管理と構成のニーズの管理を開始するためには、まず構成済みの統一変更管理 (UCM) プロセスとベース ClearCase のどちらを使用するかを決定する必要があります。本章では、2 つの方法の主な違いを、プロジェクト管理の観点から説明します。

本書の第 2 章以降は、2 部構成になっています。第 1 部では、UCM を使用してプロジェクトを管理する方法を説明します。第 2 部では、ベース ClearCase の各種ツールを使用してプロジェクトを管理する方法を説明します。

UCM とベース ClearCase との違い

ベース ClearCase は開発環境を確立するための一連の強力なツールから構成されています。その環境内で、複数の開発者が一連のファイルを共有して並行作業を進める一方、プロジェクトマネージャーは協同作業のポリシーを定義することができます。

UCM は、ClearCase を使用してバージョン管理と構成管理を行うのに最適な方法の 1 つです。UCM は ClearCase の機能に基づいて構築されています。したがって、UCM を使用すると、ベース ClearCase を詳細にマスターしなくとも、効率よく作業を進めることができます。

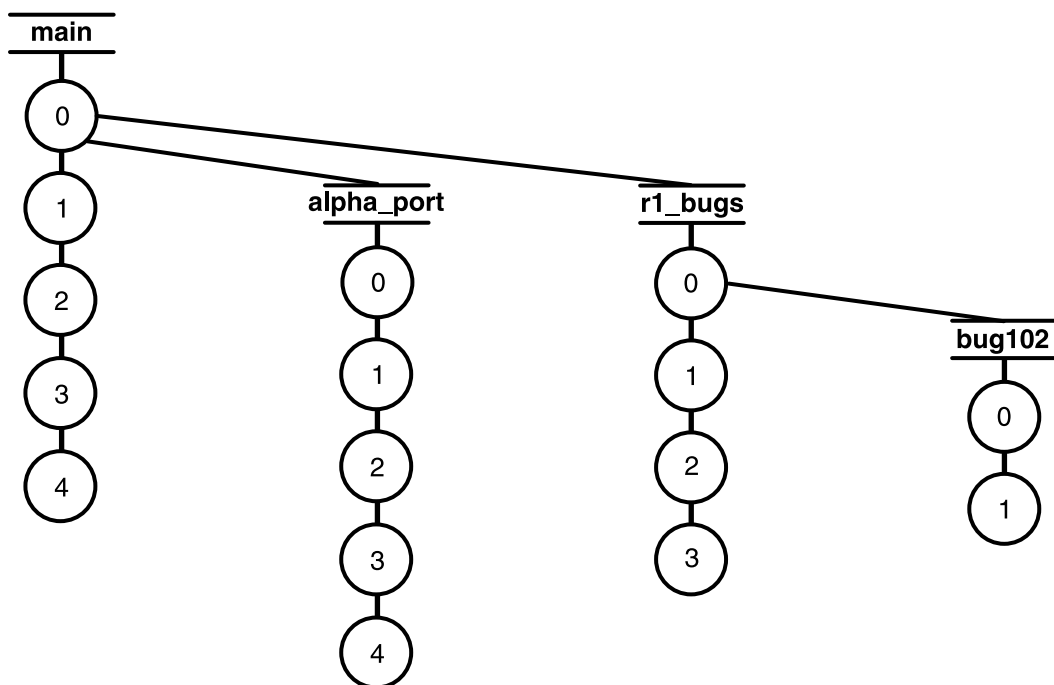
UCM では、便利な構成済みのソリューションが提供されます。一方、ベース ClearCase では、自社の環境に適したさまざまな構成管理ソリューションを実現できる柔軟性が提供されます。

ブランチとビューの作成

ベース ClearCase では、ブランチを使用して並行開発を行うことができます。ブランチとは、エレメントのバージョンのシーケンスを指定する直列形のオブジェクトです。どのエレメントにも必ず 1 つの main ブランチがあり、それが開発の主要な系列を表します。main ブランチには複数のサブブランチを作成できます。各サブブランチがそれぞれ異なる開発系列を表します。たとえば、プロジェクト チームは main ブランチを使用して新しい開発作業を進めながら、並行してサブブランチを使用してバグの修正を行うことができます。

サブブランチは複数のサブブランチを持つことができます。たとえば、製品を別のプラットフォームに移植するためのサブブランチを作成し、そのサブブランチからバグを修正するためのサブブランチを派生させることもできます。**ClearCase** では、複雑なブランチ階層を作成することができます。図 1 にブランチ階層の例を示します。このような環境の場合、プロジェクトマネージャーは、開発者が正しいブランチで作業を行うように注意する必要があります。開発者はビューを使用して作業します。ビューとは、開発者がエレメントのバージョンを作成するための作業空間をいいます。各ビューには構成仕様と呼ばれる規則のセットがあります。これらの規則によって、ビューが選択するエレメントのバージョンが決まります。

図 1 ベース ClearCase のブランチ階層



プロジェクト マネージャーは開発者に対し、構成仕様にどの規則を含めるかを指示します。これによって、開発者のビューが適切なバージョンのセットにアクセスできます。

UCM でもブランチを使用しますが、ブランチを直接操作する必要はありません。UCM ではブランチの上にストリームという層が置かれます。ストリームは、アクティビティとベースラインのリストを維持して開発者のビュー内のエレメントのバージョンを決定する、**ClearCase** オブジェクトです。UCM の一般的なプロジェクトには、1 つのインテグレーション ストリームと複数の開発ストリームがあります。前者はプロジェクトで共有する一連のエレメントを記録したものです。後者は、開発者がプロジェクト チームから独立して自分の担当の作業を行うた

めに使用されます。プロジェクトのインテグレーション ストリームでは 1 つのブランチが使用されます。開発ストリームでは、それぞれに固有のブランチが使用されます。開発ストリームの階層を作成できます。階層内の開発ストリームをサポートするブランチ階層が自動的に作成されます。

ほとんどの顧客は ClearCase を使用して並行開発環境を実装しますが、UCM とベース ClearCase は直列開発もサポートします。ベース ClearCase で直列開発環境を実装するには、すべての開発者が同一ブランチで作業するようにします。UCM では、単一ストリーム プロジェクトを作成します。これは、インテグレーション ストリーム 1 つだけからなるプロジェクトです。この場合、すべての開発者が開発ストリームではなくインテグレーション ストリームで作業します。直列開発は、開発者が緊密な共同作業を行う小規模なプロジェクト チームでのみ行ってください。

UCM プロジェクトのプロジェクト マネージャーは構成仕様の規則を決める必要はありません。ストリームによって、適切なブランチ上の適切なバージョンにアクセスするように、開発者のビューが構成されるからです。

コンポーネントを使用したファイルの編成

システム内のファイルとディレクトリの数が増大するにつれて、それらの管理を単純化する手段が必要になります。UCM では、コンポーネントを使用することによって、ファイルとディレクトリの編成を簡略化できます。エレメントをグループ化してコンポーネントに編成することにより、システム アーキテクチャの再利用可能な一部として実装できます。関連するファイルとディレクトリをコンポーネントに編成することによって、システムをディレクトリとファイルの大量の集まりとしてではなく、識別可能な少数のコンポーネントと見なすことができます。

ベースラインの作成と使用

ベースラインは、1 つ以上のコンポーネントに含まれる個々のエレメントの 1 つのバージョンに対応します。ベースラインを使用することによって、プロジェクトの特定のマイルストーンに対応するファイルのバージョンを表すことができます。たとえば、プロジェクトのソース ファイルの初期のスナップショットを示すために、**beta1** という名前のベースラインを作成することができます。

ベースラインには以下の 2 つの利点があります。

- ソフトウェア プロジェクトの初期リリースを再生成することが可能。
- プロジェクトに関連する一連のファイルをまとめることが可能。これには、ソース ファイル、製品要求文書、文書化計画、機能仕様、設計仕様、テスト計画などが含まれます。

UCM では、ベースラインの作成プロセスが自動化されており、ベースラインの操作を支援するための追加機能も用意されています。ベース ClearCase では、バージョン ラベルを作成し、それを一連のバージョンに適用することによって、ベースラインに相当するものを作成できます。

UCM では、ユーザー インターフェイス全体を通じて、ベースライン サポート機能を利用することができます。UCM ではベースラインを使用することが前提になっているからです。開発者はプロジェクトに参加したときに、まず最初にプロジェクトで推奨されているベースラインの内容を自分の作業空間に取り込む必要があります。それによって、チーム メンバー全員が確実に同じ共有ファイルを使用して作業を開始できます。また、UCM ではベースラインが示すバージョンの品質レベルを表すプロパティをベースラインに設定することができます。品質レベルの例として、「正常にプロジェクトをビルド」、「初期テストに合格」、「リグレッション テストに合格」が挙げられます。さらに高いレベルの安定性を反映するようにベースラインの品質レベル プロパティを変更することによって、実質的にベースラインをプロモートすることができます。

アクティビティの管理

ベース ClearCase ではバージョンとファイルのレベルで作業を行います。UCM ではさらに抽象化のレベルを高めたアクティビティを用意しています。アクティビティとは、開発タスクの完成に必要な作業を記録するための ClearCase のオブジェクトです。たとえば、グラフィカル ユーザー インターフェイス (GUI) を変更するアクティビティがあるとします。その変更を行うためには、いくつかのファイルを編集する必要があります。UCM では、そのアクティビティを完了するために開発者が作成する一連のバージョンを変更セットに記録します。アクティビティは UCM のユーザー インターフェイスに常に表示されています。したがって、多数のバージョンをそれぞれ指定するのではなく、アクティビティを指定することによって関連する一連のバージョンに操作を加えることができます。

アクティビティはかなりのプロジェクト タスクに相当するため、プロジェクトの進捗の把握が容易になります。たとえば、どのベースライン内でどのアクティビティが完了したかを判定することができます。UCM と ClearQuest を統合すると、アクティビティに状態と状態遷移を割り当てるなどの、プロジェクト管理機能を追加することができます。また、クエリーを実行してレポートを生成することができます。たとえば、「Pat という開発者に割り当てられていて Ready 状態にあるすべてのアクティビティを表示する」というクエリーを実行できます。

開発ポリシーの実施

ソフトウェア開発の構成管理の鍵となるのは、開発ポリシーを確立して実施することです。並行開発環境では、チーム メンバーによる一連の共有ファイルへのアクセスと更新を規制する規則を確立することが非常に重要です。そのようなポリシーは以下の 2 点に役立ちます。

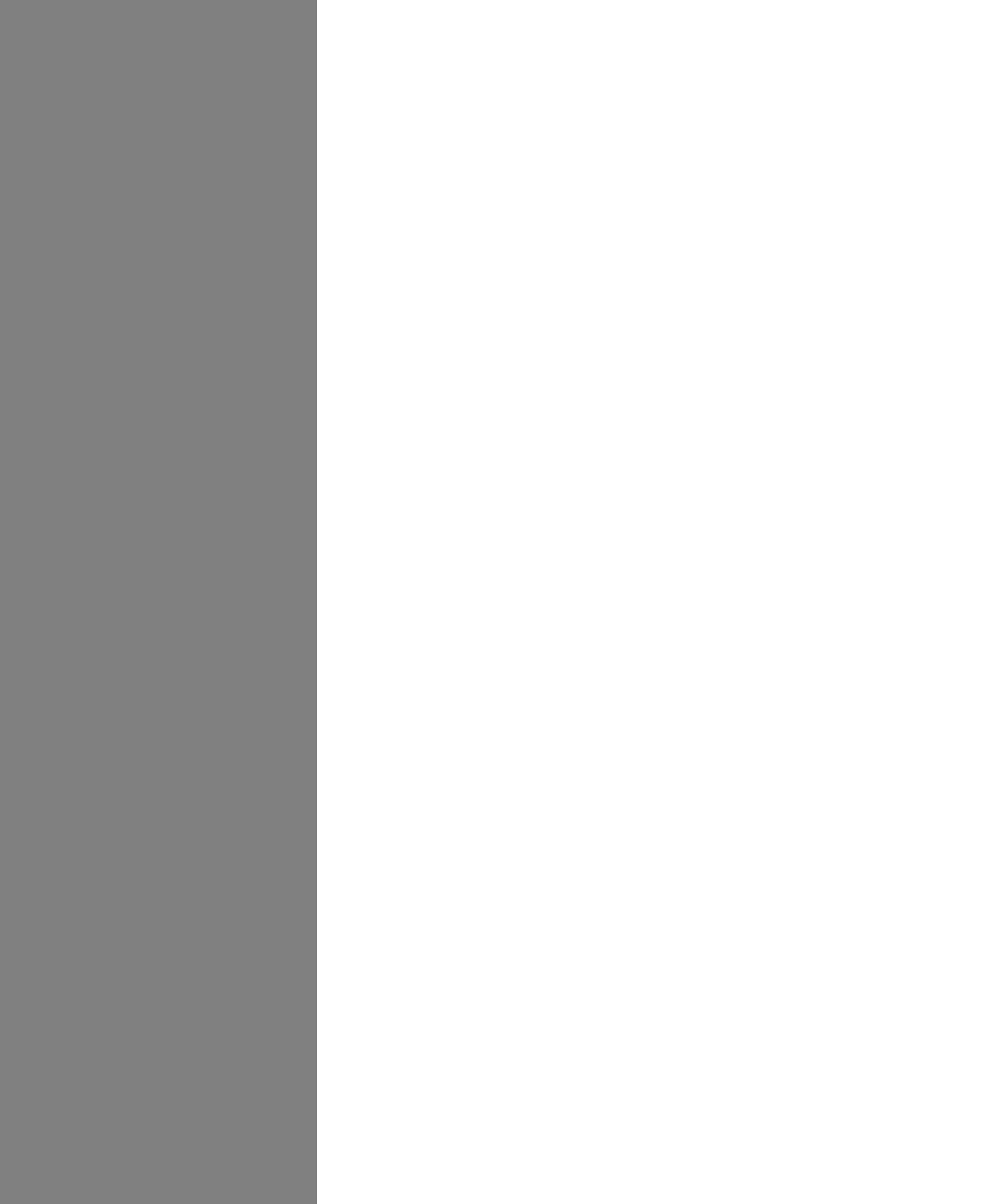
- 複数の開発者によって行われる競合する変更を可能な限り早期に検出することによって、プロジェクトのビルドに関する問題を最小限に抑える。
- チーム メンバー間のコミュニケーションを促進する。

一般的な開発ポリシーの例を以下に示します。

- 開発者は自分の作業空間とプロジェクトの推奨ベースラインとの同期を取るまでは、作業結果をプロジェクトの共有作業空間にデリバーしてはならない。
- 作業結果をプロジェクトの共有作業空間にデリバーしたときには、開発者はほかのチームメンバーにそのことを電子メールで通知しなければならない。

ベース **ClearCase** では、トリガや属性のようなツールを使用して、開発ポリシーを実施することができます。UCM には一連の一般的な開発ポリシーが組み込まれているため、GUI またはコマンド行インターフェイス (CLI) を通じて、ポリシーを設定することができます。これらのポリシーは、プロジェクト レベルとストリーム レベルで設定できます。さらに、トリガと属性を使用して、新しい UCM ポリシーを作成することもできます。

UCM での作業



本章では、Rational ClearCase に付属の統一変更管理 (UCM) について概要を説明します。特に、UCM の主要なオブジェクトを紹介し、UCM プロジェクトの管理に関するタスクについて説明します。これらのタスクの実行に必要な手順の詳細については、以降の章を参照してください。

UCM プロセスの概要

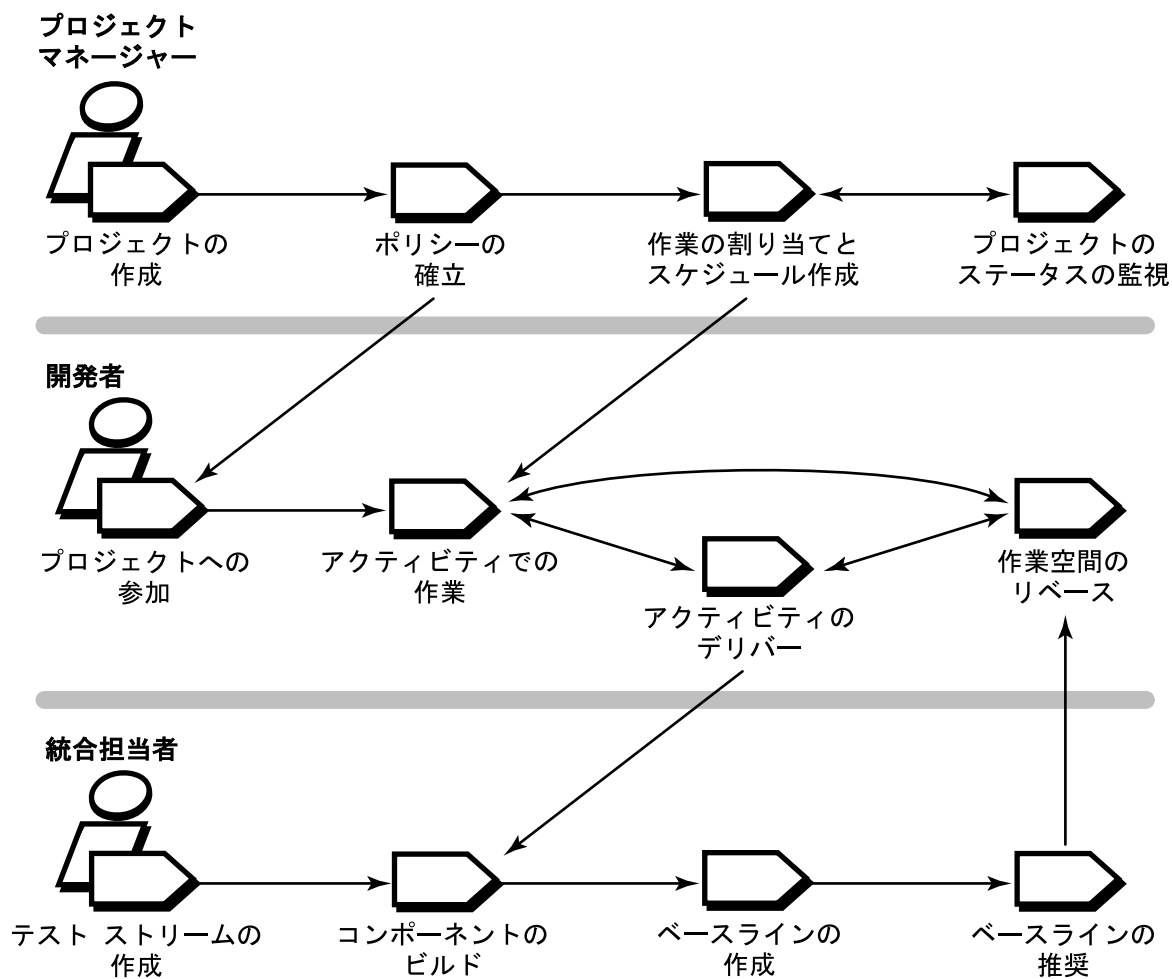
UCM では、反復的なソフトウェア開発プロセスを補完する管理サイクルに従って作業を進めます。プロジェクトチームのメンバーは UCM プロジェクト内で作業を行います。プロジェクトは、大規模な開発作業を管理するために必要な製品リリースなどの構成情報を記録したオブジェクトです。プロジェクトには 1 つのメイン共有作業空間と通常は複数の個人用作業空間があります。個人用作業空間を使用することにより、開発者はほかのメンバーから独立してアクティビティを実行することができます。プロジェクトマネージャーとプロジェクト統合担当者は、プロジェクトの共有作業空間を維持管理する責任を負います。プロジェクト内では以下のように作業を進めます。

- 1 プロジェクトマネージャーがプロジェクトを作成し、1 つ以上のコンポーネントについて一連の初期ベースラインを指定します。コンポーネントとは、開発、統合、リリースの対象となる互いに関連性を持ったディレクトリとファイルの集まりをいいます。ベースラインは、1 つまたは複数のコンポーネントの特定のバージョンを表します。
- 2 開発者は、独自の個人用作業空間を作成し、その中にプロジェクトのベースラインの内容を配置して、プロジェクトに参加します。
- 3 開発者がアクティビティを生成して、一度に 1 つのアクティビティを対象にして作業を行います。アクティビティには、バグの修正のような開発タスクを完了するために開発者が作成または修正する、一連のファイルが記録されます。アクティビティに関連する一連のファイルを変更セットと呼びます。
- 4 開発者が個人用作業空間内でアクティビティを完了し、ビルドとテストを終了後、デリバリー操作を行って作業結果をプロジェクトチームと共有します。デリバリー操作とは、開発者の個人用作業空間からプロジェクトの共有作業空間に作業結果をマージすることです。
- 5 統合担当者は、デリバリーされた作業結果を使用して、共有作業空間で定期的にプロジェクトの実行可能ファイルをビルドします。

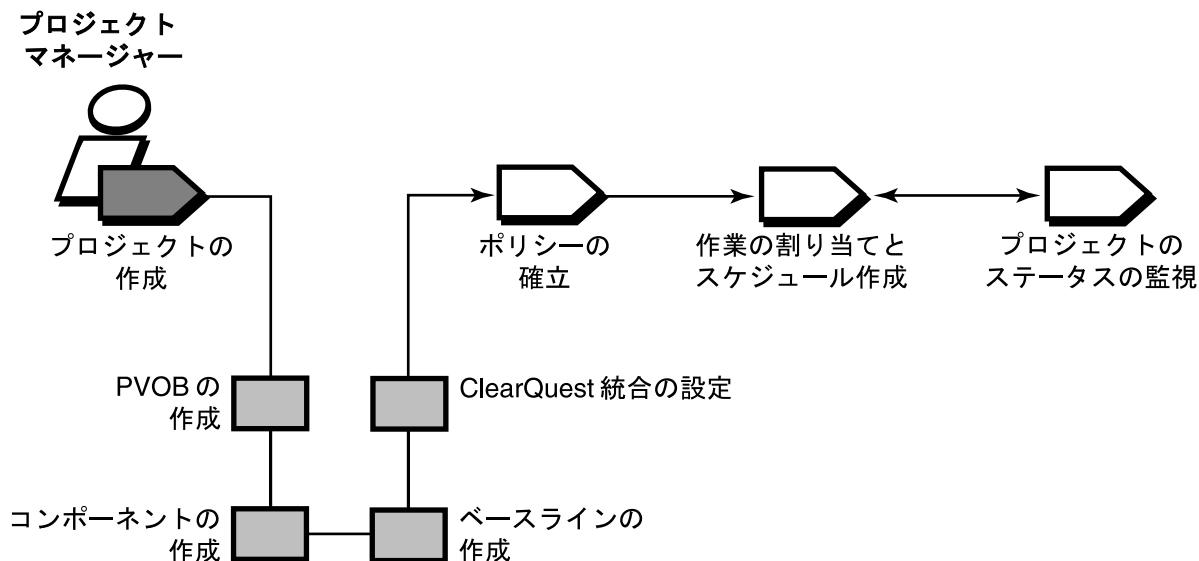
- 6 プロジェクトを正しくビルドできたら、統合担当者は新しいベースラインを作成します。別の作業空間で、ソフトウェア品質技術者のチームが新しいベースラインについてさらに詳細なテストを行います
- 7 統合担当者は、ベースラインの品質と安定性が向上するのにもない、ベースラインのプロモーション レベル属性を適切なマイルストーン (ビルド済み、テスト済み、リリース済みなど) に合わせて定期的に調整します。新しいベースラインが十分なレベルのテストに合格したら、統合担当者はそれらを推奨ベースラインに指定します。
- 8 開発者はリベース操作を行って、新しい推奨ベースラインによって示される一連のバージョンを組み込むように、各自の個人用作業空間を更新します。
- 9 開発者はアクティビティに対して作業し、完了したアクティビティをデリバーし、新しいベースラインで個人用作業空間を更新するというサイクルを繰り返します。

図 2 に、プロジェクトの管理サイクル、開発サイクル、統合サイクルの関係を示します。本書では、プロジェクト マネージャーと統合担当者が行う作業について説明します。開発者が行う作業の詳細については、『Rational ClearCase ソフトウェア開発ガイド』を参照してください。

図 2 プロジェクトマネージャー、開発者、統合担当者のワークフロー



プロジェクトの作成



プロジェクトを作成して設定するには、以下のタスクを実行します。

- プロジェクト情報を保存するリポジトリを作成する
- 開発者が作業の対象とする一連のファイルが含まれるコンポーネントを作成する
- 開発者が作業の開始点とするファイルのバージョンを指定したベースラインを作成する

ClearQuest と共に UCM を使用するには、さらにセットアップ手順を実行する必要があります。

PVOB の作成

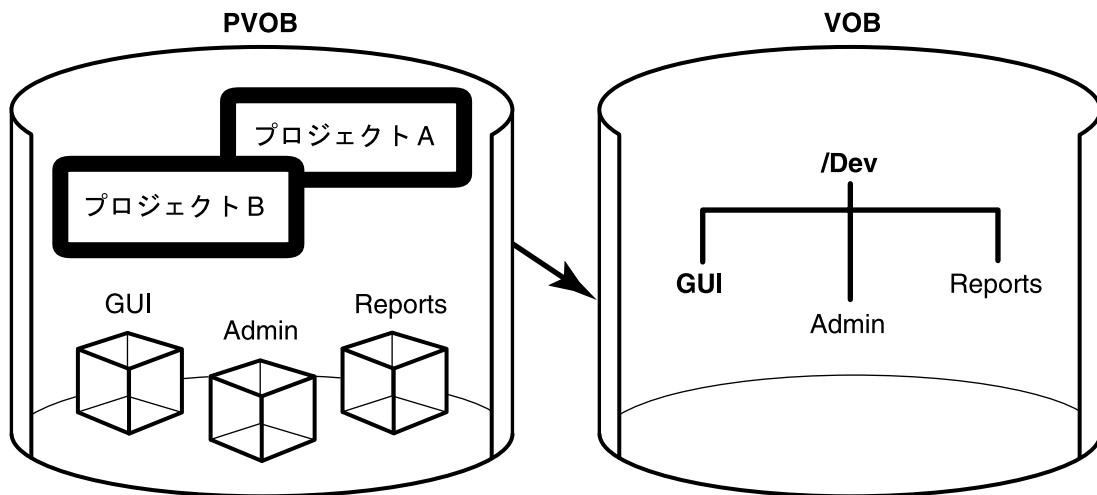
ClearCase では、バージョン付きオブジェクト ベース (VOB) というリポジトリに、ファイルエレメント、ディレクトリ エレメント、派生オブジェクト、メタデータを格納します。UCM では、各プロジェクトにプロジェクト VOB (PVOB) が必要です。PVOB は、プロジェクト、アクティビティ、変更セットなどの UCM オブジェクトを格納する、特殊な VOB です。PVOB を作成してからでないと、プロジェクトを作成することはできません。PVOB が既に作成されているかどうかは、ClearCase 管理者に確認してください。PVOB を作成する方法の詳細については、78 ページの「プロジェクト VOB の作成 (Windows)」または 79 ページの「プロジェクト VOB の作成 (UNIX)」を参照してください。

ディレクトリとファイルのコンポーネントへの編成

システム内のファイルとディレクトリの数が増大するにつれて、それらの管理を単純化する手段が必要になります。コンポーネントはファイルとディレクトリの編成を単純化するための UCM の仕組みです。エレメントをグループ化してコンポーネントに編成することにより、システムアーキテクチャの再利用可能な一部として実装できます。関連するファイルとディレクトリをコンポーネントに編成することによって、システムをディレクトリとファイルの大量の集まりとしてではなく、識別可能な少数のコンポーネントと見なすことができます。

コンポーネントのエレメントであるディレクトリとファイルの物理的な保存場所は VOB です。コンポーネントオブジェクトは PVOB に置かれます。コンポーネント内では、ディレクトリとファイルをディレクトリツリーに編成します。図 3 では、コンポーネント GUI、Admin、Reports のディレクトリツリーが VOB のルートディレクトリのすぐ下に表示されています。既存の VOB または VOB 内のディレクトリツリーをコンポーネントに変換することも、新しいコンポーネントを作成することもできます。コンポーネントを新たに作成する方法の詳細については、81 ページの「エレメントを保存するコンポーネントの作成」を参照してください。VOB をコンポーネントに変換する方法の詳細については、93 ページの「VOB からコンポーネントへの変換」を参照してください。

図 3 複数のコンポーネントがある VOB



共有作業空間と個人用作業空間

作業空間はビューとストリームから構成されます。ビューはプロジェクト内の各ファイルの単一のバージョンを示すディレクトリ ツリーです。ストリームは **ClearCase** のオブジェクトで、アクティビティとベースラインのリストを保持し、ビュー内のエレメントのバージョンを決定します。

プロジェクトにはインテグレーション ストリームが 1 つあります。そこにプロジェクトのベースラインが記録され、プロジェクトの共有エレメントのバージョンへのアクセスが可能になります。インテグレーション ストリームとそれに対応するインテグレーション ビューによって、プロジェクトのメイン共有作業空間が表現されます。

一般的なプロジェクトでは、参加している各開発者に開発ストリームとそれに対応する開発ビューで構成される個人用作業空間を割り当てます。開発ストリームは、開発者のアクティビティのリストを保持し、開発者のビューに表示されるエレメントのバージョンを決定します。

UCM の GUI からプロジェクトを作成すると、**ClearCase** によりインテグレーション ストリームが作成されます。コマンド行インターフェイスからプロジェクトを作成する場合は、インテグレーション ストリームを自分で明示的に作成する必要があります。開発者はプロジェクトに参加するときに、自分の開発ストリームと開発ビューを作成します。プロジェクトへの参加方法については、『**Rational ClearCase** ソフトウェア開発ガイド』を参照してください。

ストリーム階層

基本的な UCM プロセスでは、インテグレーション ストリームがプロジェクトの唯一の共有作業空間です。プロジェクトの特定の部分について共同で作業している開発者のために、共有作業空間を追加する必要があることもあります。その場合は、開発ストリームの階層を作成できます。たとえば、開発ストリームを作成し、特定の機能を担当する開発者の共有作業空間に指定することができます。その後、この機能の開発ストリームの下位に開発者がそれぞれに固有の開発ストリームとビューを作成します。開発者は、この機能の開発ストリームの推奨ベースラインに対して、作業結果のデリバーとストリームのリベースを行います。開発ストリーム階層の詳細については、36 ページの「ストリーム方針の選択」を参照してください。

単一ストリーム プロジェクト

ほとんどの顧客は UCM を使用して並行開発環境を実装しますが、UCM では単一ストリームプロジェクトを作成することによって直列開発環境も実装できます。単一ストリーム プロジェクトとは、インテグレーション ストリーム 1 つだけからなるプロジェクトのことです。この場合、すべての開発者が開発ストリームではなくインテグレーション ストリームで作業します。各開発者がそれぞれ固有のビューを使用します。直列開発は、開発者が緊密な共同作業を行う小規模なプロジェクト チームでのみ行ってください。単一ストリーム プロジェクトの詳細については、36 ページの「ストリーム方針の選択」を参照してください。

ベースラインからの開始

プロジェクトのコンポーネントを新規に作成するか、既存のコンポーネントを選択した後で、チームに属する開発者が作業の開始点として使用するベースラインをいくつか指定し、推奨する必要があります。ベースラインにはコンポーネント内で参照できる各エレメントのバージョンが指定されています。図 4 に示すベースライン **BL1** と **BL2** は、それぞれコンポーネント **A** とコンポーネント **B** を表しています。

開発者は、プロジェクトに参加するときに、プロジェクトの推奨ベースラインによって示されるディレクトリ エレメントとファイル エレメントのバージョンを自分の作業空間に取り込みます。また、開発者は特定の機能に対応する開発ストリーム レベルでプロジェクトに参加することもできます。その場合は、その開発ストリームの推奨ベースラインを自分の作業空間に取り込みます。それによって、プロジェクト チームのすべてのメンバーが、確実に同じファイルを使用して作業を開始することができます。

プロジェクト チームが複数のコンポーネントで作業を行う場合は、複合ベースラインを使用できます。複合ベースラインは、ほかのコンポーネントからベースラインを選択したものです。図 5 の複合ベースライン **PB1** には、コンポーネント **A** のベースライン **BL1** とコンポーネント **B** のベースライン **BL2** が選択されています。コンポーネント **Proj** には固有のエレメントはありません。このコンポーネントには、プロジェクトのコンポーネントの推奨ベースラインを選択する複合ベースラインだけが含まれています。このように複合ベースラインを使用することによって、1 つのベースラインでプロジェクト全体を表すことができます。

図 4 2つのコンポーネントのベースライン

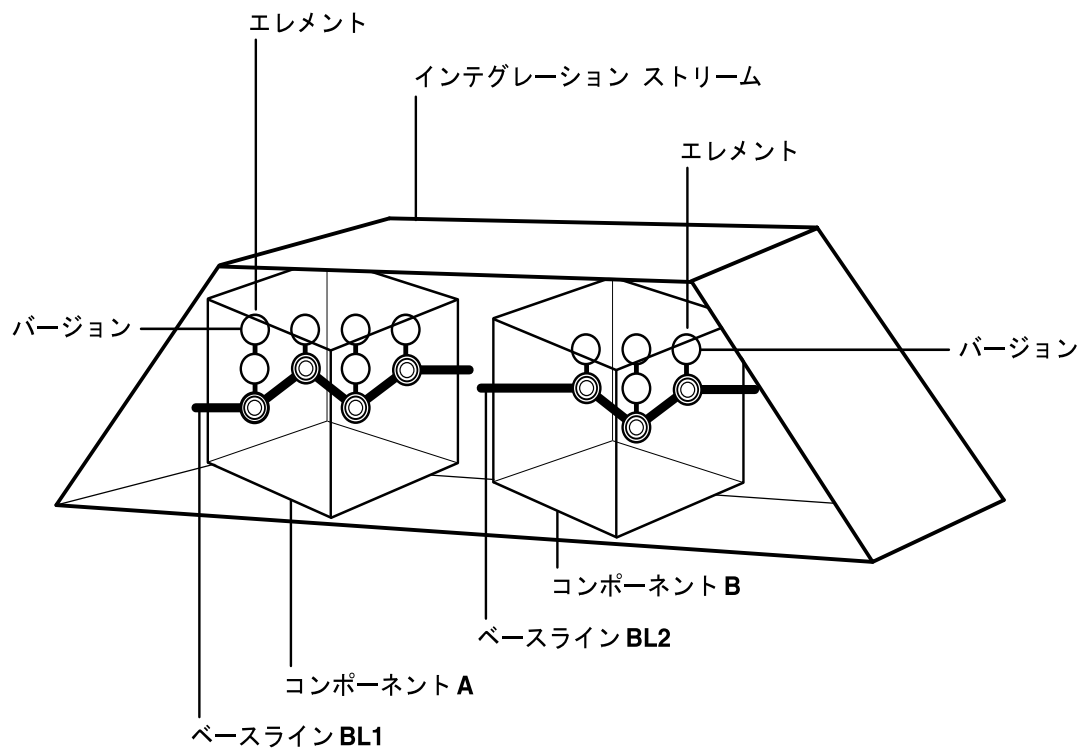
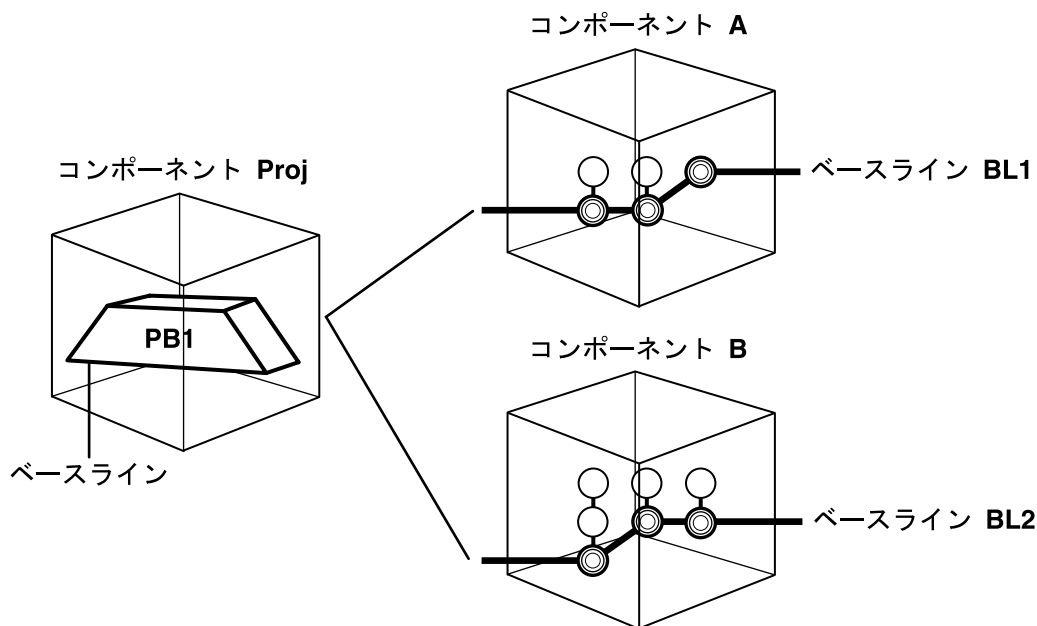


図 5 複合ベースライン



UCM と ClearQuest の統合の設定

変更依頼管理ツール Rational ClearQuest を使用しなくても UCM を使用できますが、ClearQuest と統合すると、プロジェクト管理機能とアクティビティ管理機能が大幅に強化されます。UCM プロジェクトを ClearQuest と併用するように設定すると、統合機能によりすべてのプロジェクト アクティビティが ClearQuest のレコードにリンクされます。これにより、UCM-ClearQuest の状態遷移モデルと ClearQuest のクエリー、レポート作成、グラフ機能を利用することができます。以下の操作が可能になります。

- アクティビティを開発者に割り当てる
- 状態と状態遷移規則を使用してアクティビティを管理する
- データベース クエリーに基づいてレポートを生成する
- 実施すべき追加の開発ポリシーを選択する

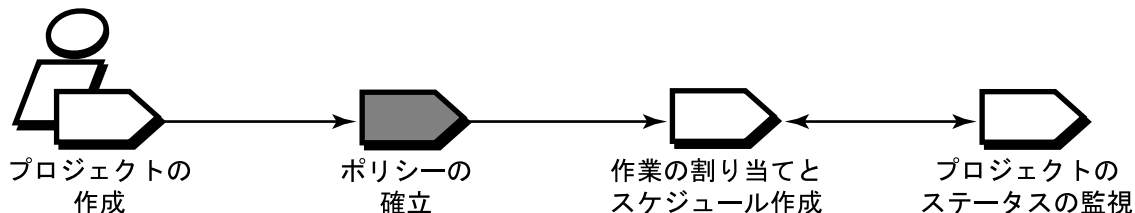
UCM と ClearQuest の統合を設定するには

- 1 ClearQuest のスキーマを UCM に適用できるようにします。または、定義済みの UCM に適用可能なスキーマを使用します。
- 2 ClearQuest のユーザー データベースを上記のスキーマを使用するように作成またはアップグレードします。
- 3 UCM プロジェクトで ClearQuest を使用可能にします。

統合の詳細については、25 ページの「UCM と ClearQuest の統合の概要」を参照してください。

ポリシーの設定

プロジェクト
マネージャー

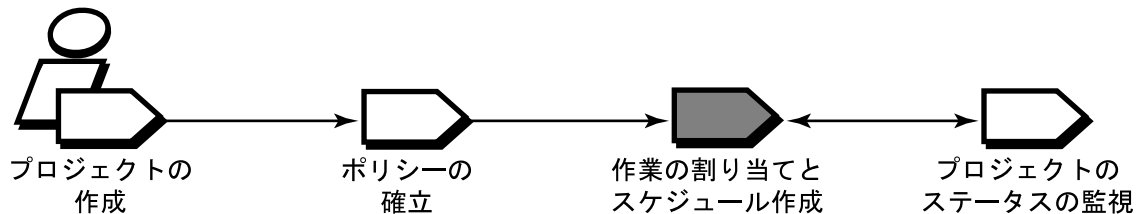


UCM には一連のポリシーが用意されており、このポリシーに基づいて、プロジェクトチームのメンバーに開発の実践原則を実施させることができます。ポリシーを設定することにより、プロジェクトチームのメンバー間のコミュニケーションを改善するとともに、各開発者の作業結果の統合時に遭遇する可能性のある問題を最小限に抑えることができます。たとえば、開発者は自分の作業空間をプロジェクトの最新の推奨ベースラインで更新してから作業結果をデリバーする、というポリシーを設定します。これにより、作業結果をデリバーする際に、開発者が複雑なマージ操作を行う必要性が低下します。UCM で設定可能なすべてのポリシーの説明については、「第 4 章 ポリシーの設定」を参照してください。ポリシーはプロジェクトとストリームに設定できます。

UCM に用意されているポリシーに加え、カスタマイズした開発ポリシーを実施するトリガを作成し、UCM 操作に適用することもできます。トリガを作成する方法の詳細については、「第 8 章 トリガを使用した開発ポリシーの実施」を参照してください。

作業の割り当て

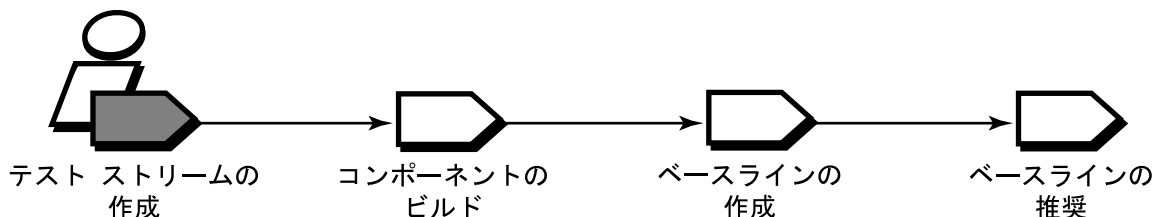
プロジェクト
マネージャー



このタスクは任意であり、UCM と ClearQuest を統合している場合のみ実行できます。プロジェクト マネージャーは、プロジェクト チームの高レベルのタスクを特定し、スケジュールする責任を負います。組織によっては、プロジェクト マネージャーがアクティビティを作成し、開発者に割り当てます。また、開発者がそれぞれ自分のアクティビティを作成することもあります。ClearQuest でアクティビティを作成し、割り当てる方法の詳細については、100 ページの「アクティビティの割り当て」を参照してください。

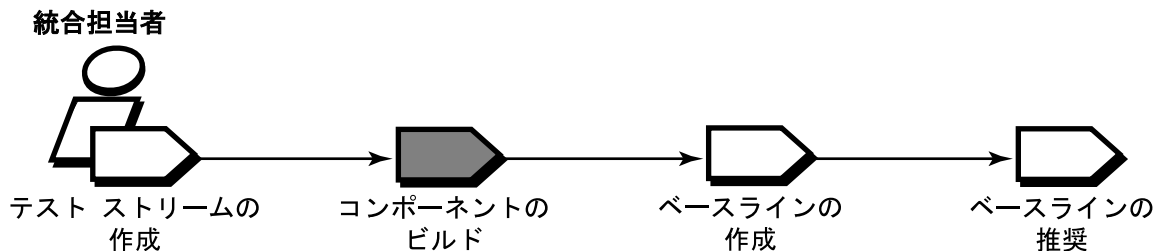
テスト ストリームの作成

統合担当者



統合担当者は、開発者からデリバーされた作業結果のビルド、ベースラインの作成、作成したベースラインのテストに責任を負います。インテグレーション ストリームにベースラインを作成するときには、開発者が作業結果をデリバーできないようにインテグレーション ストリームをロックします。これにより、一連のファイルを確実に静的な状態で扱うことができます。インテグレーション ストリームに作成した新しいベースラインに対して簡単な検証テストを実行することもできます。ただし、デリバーのバックログが作成されるため、インテグレーション ストリームを長時間ロックしないことをお勧めします。リグレッション テストなどの厳密なテストを実行するには、ベースラインのテスト専用の開発ストリームを作成する必要があります。テスト ストリームを作成する方法の詳細については、105 ページの「ベースライン テスト用開発ストリームの作成」を参照してください。

コンポーネントのビルド



新しいベースラインを作成する前に、インテグレーション ストリームで、現在のベースラインとその作成後にインテグレーション ストリームにデリバーされた作業結果を使用してコンポーネントをビルドします。コンポーネントをビルドするときには、一連のファイルを実際に静的な状態で扱えるようにインテグレーション ストリームをロックします。ビルドが完了したら、デリバーされた最新の作業結果を選択するベースラインを作成します。プロジェクトで機能固有の開発ストリームを使用している場合は、インテグレーション ストリームに加え、機能固有の開発ストリームでも、このタスクを実行します。

MultiSite についての考慮点

製品メモ: 現在、Rational ClearCase LT は Rational ClearCase MultiSite をサポートしていません。

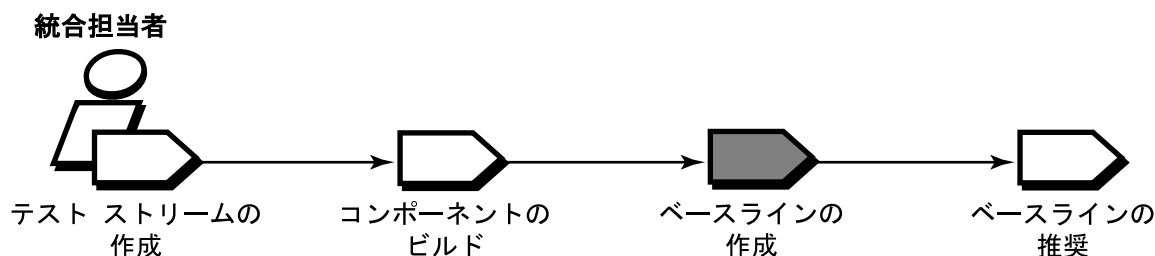
ほとんどの場合、開発者は開始したデリバー操作を完了します。プロジェクトで ClearCase MultiSite を使用する場合は、一部のデリバー操作を完了してからでなければコンポーネントをビルドできないことがあります。ClearCase MultiSite は、ClearCase の上層に位置する製品です。地理的に散在するプロジェクト チームの間で並行ソフトウェア開発をサポートするために、多くの ClearCase ユーザーがこの製品を使用しています。MultiSite を使用すると、多くの開発者が異なる場所から同じ VOB を対象にして同時に作業を行うことができます。各ロケーションでは専用にコピーした VOB に対して作業を行います。そのコピーをレプリカと呼びます。

競合を避けるために、MultiSite ではマスタシップと呼ばれる排他的な修正権限を伴うスキーマを使用しています。ストリームやブランチのような VOB オブジェクトにはマスター レプリカが割り当てられます。マスター レプリカにはそれらのオブジェクトを修正または削除する排他的な権限があります。

MultiSite 構成では、開発者チームがリモート サイトで作業する場合があるため、プロジェクトのインテグレーション ストリーム用のレプリカと開発者の開発ストリーム用のレプリカは別物である可能性があります。そのような状況では、開発者はインテグレーション ストリームへのデリバー操作を完了することはできません。プロジェクト マネージャーがその操作を完了する必要があります。UCM では、リモート デリバーと呼ばれるデリバー操作のバリエーションを用意しています。UCM は、インテグレーション ストリームがリモート サイトでマスター登録されたと判断した場合、デリバー操作をリモート デリバーとします。これによって、デリバー操作は開始されますが、バージョンはマージされません。プロジェクト マネージャーがデリバー操作をリモート サイトで完了する必要があります。

リモート操作の完了方法については、113 ページの「デリバーできる作業結果の検索」を参照してください。

新しいベースラインの作成



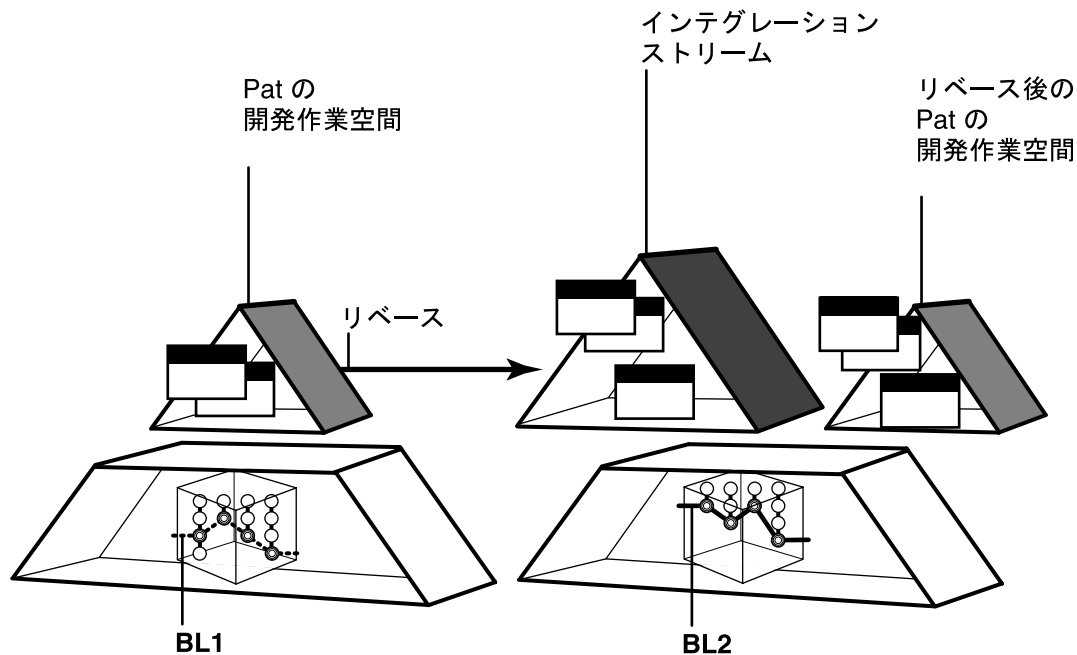
開発者間で作業結果を確実に同期するために、定期的に新しいベースラインを作成します。新しいベースラインは、前回ベースラインを作成した時点より後に開発者によってデリバーされた作業結果を含みます。プロジェクトで機能固有の開発ストリームを使用している場合は、インテグレーション ストリームに加え、機能固有の開発ストリームでも、このタスクを実行します。環境によっては、ある機能に責任を持つ開発者が機能固有の開発ストリームの統合担当者を兼ねる場合があります。新しいベースラインを作成するには、次の手順を実行します。

- 1 インテグレーション ストリームがロックされていて、ベースラインの作成中に開発者が作業結果をデリバーできないことを確認します。開発者は各自の開発環境で作業を継続することができます。
- 2 コンポーネントをテストして、プロジェクトの安定性を検証します。
- 3 ベースラインを作成します。
- 4 インテグレーション ストリームのロックを解除して、開発者が作業をデリバーできるようにします。

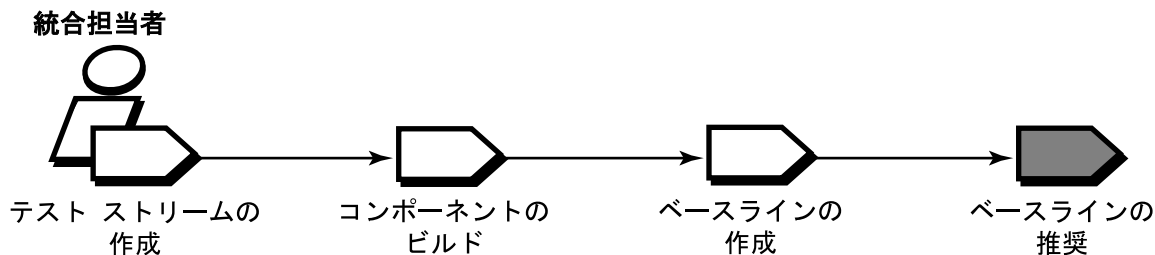
ソフトウェア品質保証技術者のチームが新しいベースラインについて詳細なテストを実施し、安定性を確認した後で、そのベースラインを推奨ベースラインにします。開発者はリベース操作を行って、各自の作業空間を新しいベースラインで更新します。つまり、インテグレーションストリームまたは機能固有の開発ストリームから自分の開発ストリームにファイルとディレクトリをマージします。

図 6 にベースライン BL1 から BL2 へのリベース操作を示します。ベースラインを作成する方法の詳細については、115 ページの「新規ベースラインの作成」を参照してください。

図 6 リベース操作



ベースラインの推奨

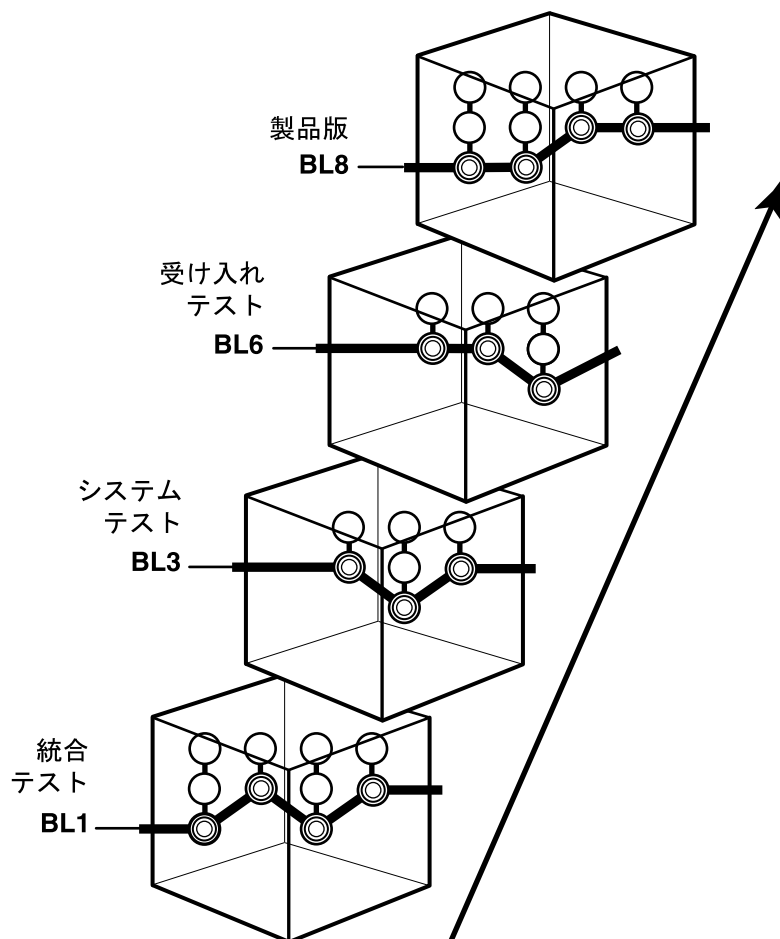


プロジェクトの作業が進行し、コンポーネントの品質と安定度が向上したら、重要なマイルストーンに合わせてベースラインのプロモーション レベル属性を変更します。プロモーション レベル属性は通常、テスト水準を表します。たとえば、図 7 は 3 レベルのテストを経てベースラインが上がってゆく様子を示しています。ベースライン **BL8** は実稼動可能な状態であることを示します。

ベースラインが安定と見なされるレベルのテストに合格したら、それを推奨ベースラインに指定します。これに伴い、開発者は各自の開発ストリームを推奨ベースラインにリベースします。開発ストリームを推奨ベースラインに合わせるリベースを行ってから作業結果をデリバリーするようポリシーを設定することもできます。このポリシーにより、ベースラインが許容レベルのテストに合格すると、開発者が確実に作業空間をリベースします。

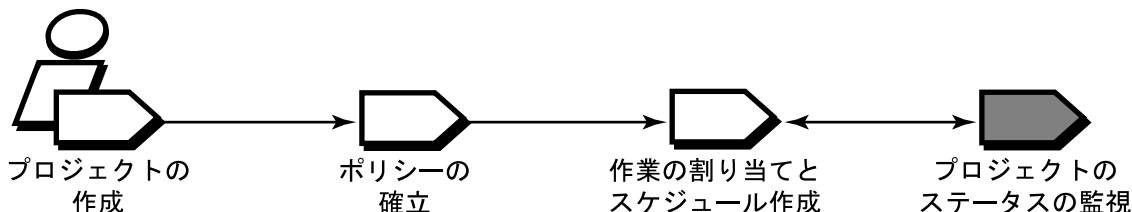
推奨ベースラインの詳細については、119 ページの「ベースラインの推奨」を参照してください。

図 7 ベースラインのプロモート



プロジェクト状態の監視

プロジェクト
マネージャー



ClearCase には、プロジェクトの進捗を確認するために役立つツールがいくつか用意されています。

- UCM と ClearQuest を統合すると、6 つの ClearQuest クエリーを使用して、プロジェクトのアクティビティに関する情報を取得できます。たとえば、アクティブな状態にあるすべてのアクティビティや特定の開発者に割り当てられたすべてのアクティブなアクティビティを表示できます。さらに、独自の ClearQuest クエリーを作成することもできます。
- ベースラインの比較 GUI は、あるコンポーネントの 2 つのベースラインを比較し、各ベースラインに関連付けられたアクティビティとバージョンの違いを表示します。この機能を使用すると、特定の機能がベースラインに追加されたことを確認できます。
- コンポーネント ツリー ブラウザ (Windows のみ) には、コンポーネントのベースライン履歴が表示されます。GUI には、表示をフィルタし、指定したストリームまたは指定したプロモーション レベル以上のベースラインだけを表示できる機能があります。
- ClearCase Report Builder と Report Viewer (Windows のみ) では、プロジェクト環境固有のレポートを作成して参照できます。Report Builder には、プロジェクト、ストリーム、エレメント、ビューなどの ClearCase オブジェクト別に構成されたレポートが用意されています。さらに、レポートを生成して表示する手順をカスタマイズすることもできます。

これらのツールの使用方法の詳細については、121 ページの「プロジェクト状態の監視」を参照してください。

UCM と ClearQuest の統合の概要

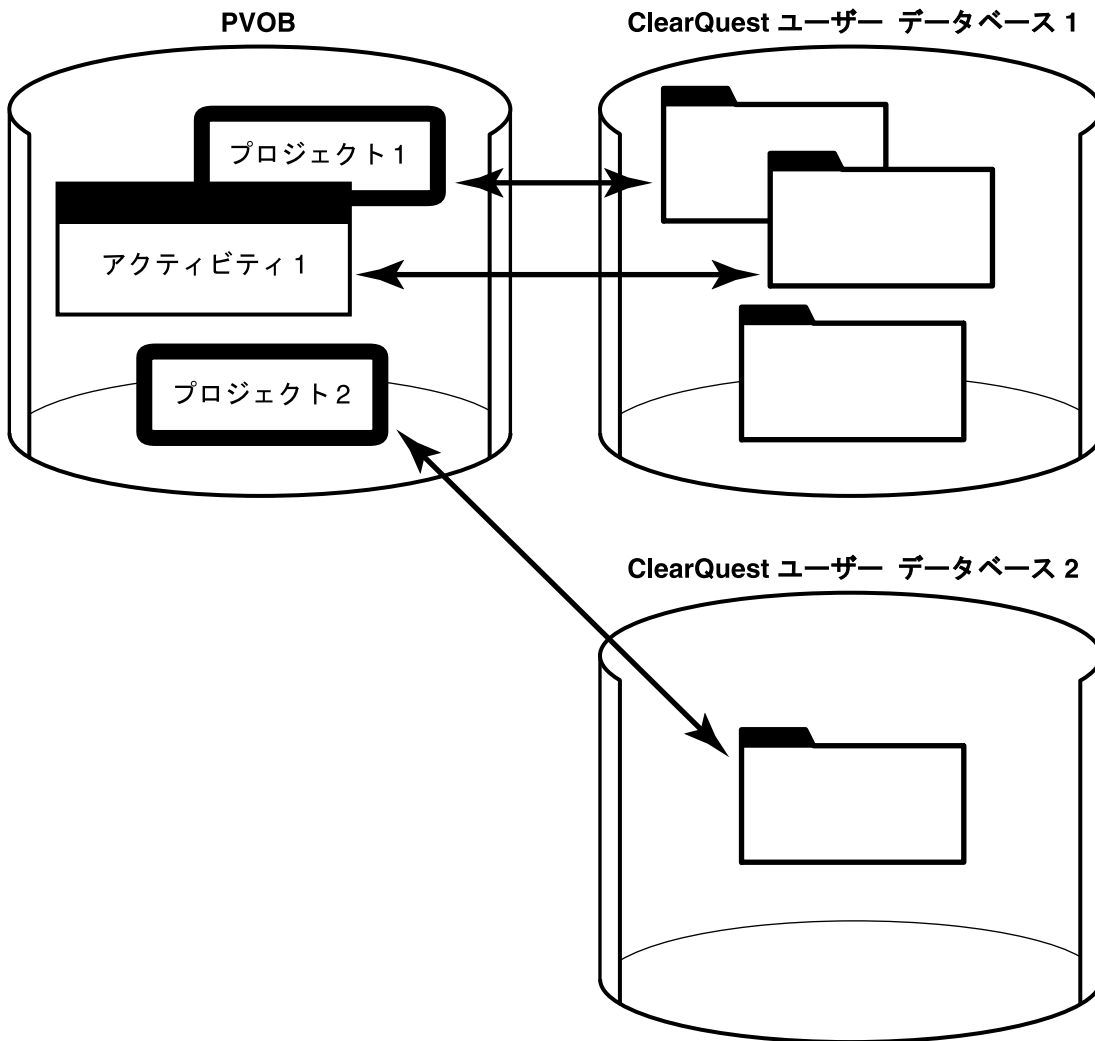
ここでは、UCM と ClearQuest の統合に関する以下の概念について説明します。

- UCM のオブジェクトと ClearQuest のオブジェクトの関連付け
- UCM に適用可能なスキーマ
- クエリー
- 状態タイプ

UCM のオブジェクトと ClearQuest のオブジェクトの関連付け

UCM と ClearQuest の統合を設定すると、UCM のオブジェクトと ClearQuest のオブジェクトがリンクされます。これらのオブジェクトの双方向のリンクを図 8 に示します。

図 8 UCM と ClearQuest の統合での UCM のオブジェクトと ClearQuest のオブジェクトの関連付け



プロジェクトから ClearQuest のユーザー データベースへのリンクを有効にすると、プロジェクトの PVOB 内にそのデータベースへの参照が記録されます。

ClearQuest を適用可能な各プロジェクトは、ClearQuest のユーザー データベース内のレコードタイプが **UCM_Project** であるプロジェクト レコードにリンクされます。

ClearQuest を適用可能なプロジェクト内の各アクティビティは、データベース内のレコードにリンクされます。アクティビティのタイトルは、対応する ClearCase レコードの headline フィールドにリンクされます。ClearCase 内のアクティビティのタイトルを変更すると、それに合わせて ClearQuest 内の見出しも変更されます。逆の場合も同様です。同様に、アクティビティの ID は、その ClearQuest レコード内の ID フィールドにリンクされます。

ClearQuest のユーザー データベースにはアクティビティにリンクされたレコードとリンクされていないレコードを混在させることが可能です。図 8 に示す ClearQuest ユーザー データベース 1 には、アクティビティにリンクされていないレコードがあります。UCM を採用する前に ClearQuest のユーザー データベースが存在していると、このような状況が発生します。アクティビティを作成すると、対応する ClearQuest レコードが作成されます。しかし、UCM と ClearQuest を統合する前からユーザー データベース内に存在していたレコードはリンクされないままです。また、開発者が作業結果をレコードに設定するまで、そのレコードはアクティビティにリンクされません。

UCM に適用可能なスキーマ

ClearQuest のスキーマはデータベースの定義です。UCM と ClearQuest の統合を使用するには、UCM に適用可能なスキーマに基づいた ClearQuest のユーザー データベースを作成またはアップグレードする必要があります。UCM に適用可能なスキーマには、ある種のフィールド、スクリプト、アクション、状態タイプが含まれます。ClearQuest にはユーザーが使用できる定義済みの UCM に適用可能なスキーマが 2 つ用意されています。別途作成したスキーマまたは別の定義済みのスキーマを UCM に適用することもできます。UCM に適用可能なスキーマの詳細については、51 ページの「適用するスキーマの決定」を参照してください。

状態タイプ

ClearQuest では、要求が提出されてから完了するまでの進捗状況を把握するために状態を使用します。状態は進捗の段階を表します。ある状態から別の状態に移ることを状態遷移と呼びます。UCM と ClearQuest の統合では独自の状態遷移モデルを使用します。そのモデルを実装するために、状態タイプが使用されます。状態タイプとは、UCM において状態遷移の順序を定義するために使用される状態のカテゴリのことです。状態は必要なだけ定義することができます。しかし、UCM に適用可能なレコード タイプ内のすべての状態が以下の状態タイプのどれか 1 つに基づいている必要があります。

- Waiting
- Ready
- Active
- Complete

同じ状態タイプに複数の状態が属することもできます。ただし、状態タイプの状態間の遷移の経路を少なくとも 1 つ「Waiting、Ready、Active、Complete」の順に定義する必要があります。状態タイプの詳細については、71 ページの「状態タイプの設定」を参照してください。

UCM に適用可能な ClearQuest のスキーマでのクエリー

UCM に適用可能なスキーマにはクエリーが 6 つあります。UCM に適用可能なスキーマを使用するように ClearQuest のユーザー データベースを作成またはアップグレードすると、ユーザー データベースのワークスペース内の [共用クエリー] フォルダの 2 つのサブフォルダ内にそれらのクエリーがインストールされます。これらのクエリーを使用すると、開発者は各自に割り当てられているアクティビティを容易に参照し、プロジェクト マネージャーはどのプロジェクトのどのアクティビティがアクティブかを簡単に参照できます。クエリーの詳細については、124 ページの「ClearQuest ユーザー データベースのクエリー」を参照してください。

本章では、Rational ClearCase で UCM プロジェクトを構成管理環境として使用する際に考慮すべき事項について説明します。プロジェクトなどの UCM のオブジェクトを作成する前に、構成管理計画を作成することを強くお勧めします。作成した計画を実施する方法については、「第 6 章 プロジェクトのセットアップ」を参照してください。

開始点としてのシステム アーキテクチャの使用法

高品質のソフトウェアを開発し保守するには、システム アーキテクチャの定義が不可欠です。Rational Unified Process® には、システム アーキテクチャを定義して使用することが、ソフトウェア開発における 6 つの最善の実践原則の 1 つだと説明されています。システム アーキテクチャはシステムをその環境内で最高のレベルまで抽象化したものです。Rational Unified Process では、システム アーキテクチャは以下の面にかかわると解説しています。

- ソフトウェアの編成に関連する重要な意思決定
- システムを構成するエレメントとそのインターフェイスと、それらのエレメント間でのコラボレーションに指定されている動作の選択
- 構造的エレメントと動作的エレメントを順次さらに大きなサブシステムに構成すること
- それらの編成、エレメント、インターフェイス、コラボレーション、構成の指針となるアーキテクチャ スタイル

システム アーキテクチャが明確に文書化されていると、ソフトウェア開発プロセスの質が向上します。また、構成管理環境の構造を定義する理想的な開始点になります。

コンポーネントへのシステム アーキテクチャのマッピング

建築では、各種の設計図を使用して、建物のアーキテクチャのさまざまな面（間取り、電気配線、配管など）を表します。これと同様に、優れたソフトウェア システム アーキテクチャには各種の側面を表すさまざまなビューがあります。Rational Unified Process では、特定の観点からシステムを単純化して表現（抽象化）したものとして、アーキテクチャ ビューを定義しています。その際、具体的な関心事項を取り上げますが、その視点に関係のない事項は省きます。

Rational Unified Process では 5 つのアーキテクチャ ビューを使用するよう推奨しています。その中で、構成管理にとっては、実装ビューが最も重要です。実装ビューでは、システムの論理パッケージ、オブジェクト、またはモジュールを実装する物理的なファイルとディレクトリを指定します。たとえば、システム アーキテクチャにライセンス モジュールが含まれるとします。その場合、ライセンス モジュールを構成するディレクトリとファイルを実装ビューに指定します。

実装ビューから、システムに必要な一連の UCM コンポーネントを指定できる必要があります。コンポーネントとは、開発、統合、リリースする一連のディレクトリとファイル エLEMENT です。大規模なシステムは通常、多数のコンポーネントで構成されます。小規模なシステムは 1 つのコンポーネントで構成されることもあります。

バージョン管理の対象の決定

何をバージョン管理の対象にするか決定する際に、対象をソース コードのファイルとディレクトリに限定してはなりません。構成管理にはプロジェクトの進展に応じて履歴を記録する機能があります。これにより、任意の時点でプロジェクトを迅速かつ容易に再生成することができます。プロジェクト全体を記録するには、関連するすべてのファイルとディレクトリを対象に含めます。主なものを以下に示します。

- ソース コードのファイルとディレクトリ
- Rational Rose® ファイルなどのモデル ファイル
- ライブラリ
- 実行可能ファイル
- インターフェイス
- テスト スクリプト
- プロジェクト計画
- コンパイラ、その他の開発ツール、システム ヘッダー ファイル
- システム文書とユーザー文書
- 要求文書

プロジェクトへのコンポーネントのマッピング

システム アーキテクチャを一連のコンポーネントにマッピングし、バージョン管理の対象にするすべてのファイルとディレクトリを指定した後で、使用するプロジェクトの数を決定する必要があります。一般に、プロジェクトは、特定のリリースに関する作業を行うプロジェクト チームのためのプロジェクト管理環境と考えることができます。チーム メンバーは共同で作業して、一連の関連するコンポーネントを開発、統合、テスト、リリースします。多くの場合、システムを 1 つのプロジェクトで開発することが可能です。いくつかのプロジェクトに分けて作業する場合もあります。プロジェクトの数を決定する際には、以下の要因を考慮します。

- 必要な統合の度合い
- 製品の複数バージョンを並行して開発し、リリースする必要があるかどうか

統合の度合い

さまざまなコンポーネント間の関係を判断します。高度に統合する必要のある関連コンポーネントは同じプロジェクトに含めます。これにより、コンポーネントを頻繁にビルドしてテストできるので、開発サイクルの後期にコンポーネントを統合した時点での問題発生を回避できます。

並行リリースの必要性

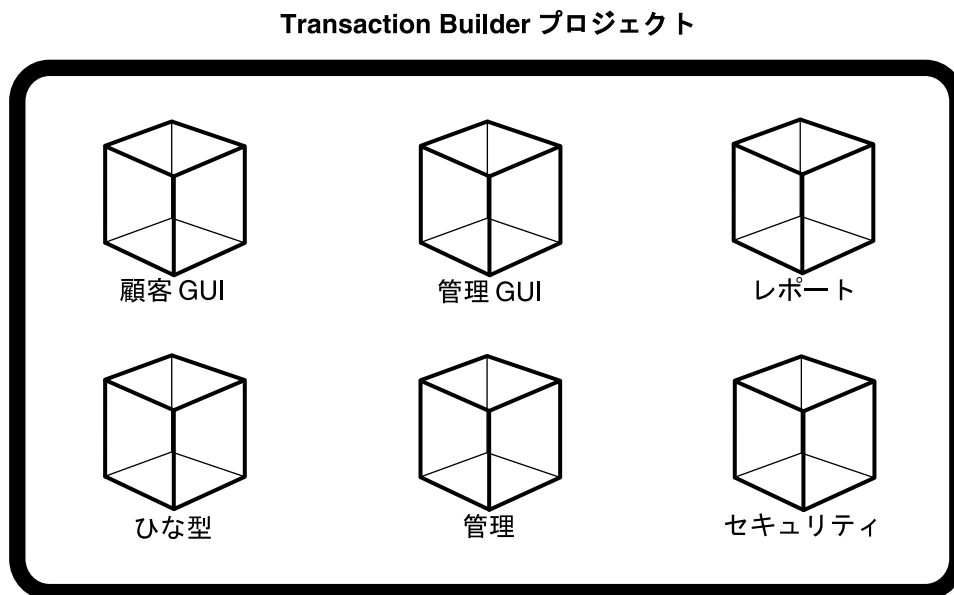
システムの複数のバージョンを並行して開発する必要がある場合、バージョンごとに個別のプロジェクトを使用することを検討します。たとえば、パッチ リリースと新しいリリースのために同時に作業する必要があるとします。その場合、両方のプロジェクトで使用するコンポーネントは大部分が同じです(複数のプロジェクトでは同じコンポーネントを変更できます)。パッチ リリース プロジェクトの作業が完了したら、その結果を新しいリリース プロジェクトに統合します。

チームが将来にわたりシステムのバージョンを数多く開発し、リリースしていくと予想される場合は、メインライン プロジェクトを作成できます。メインライン プロジェクトは、一定期間内に存続する関連プロジェクトを統合する際の、1 つの統合ポイントの役割を果たします。メインライン プロジェクトの使用方法的詳細については、「第 9 章 複数プロジェクトの並行リリースの管理」を参照してください。

例

図 9 に、Transaction Builder システム用に計画されている初期の一連のコンポーネントを示します。30 人の開発者から成るチームがこのシステムに取り組みます。コンポーネント間には高度な統合が必要で、大部分の開発者がいくつかのコンポーネントを対象に作業するので、プロジェクト マネージャーはすべてのコンポーネントを 1 つのプロジェクトで管理しています。

図 9 Transaction Builder プロジェクトで使用するコンポーネント



コンポーネントの編成

初期の一連のコンポーネントにシステム アーキテクチャをマッピングし、どのプロジェクトがそのコンポーネントにアクセスするかを決定したら、以下のタスクを実行して計画を詳細に設定します。

- 使用する VOB の数を決定する
- 必要な追加のコンポーネントを指定する
- コンポーネントのディレクトリ構造を定義する
- 読み取り専用のコンポーネントを把握する

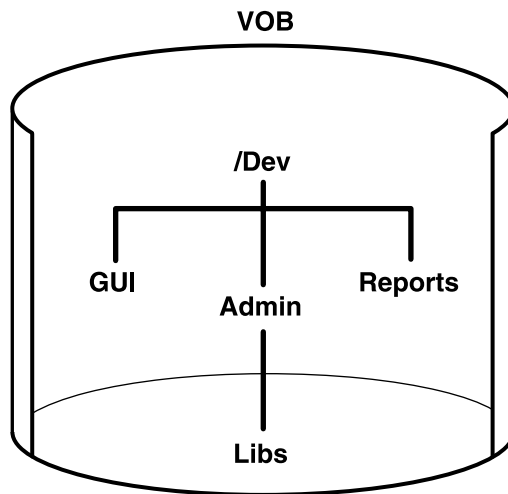
使用する VOB 数の決定

ClearCase では、1 つの VOB に複数のコンポーネントを保存できます。少数のコンポーネントしか使用しないプロジェクトでは、各コンポーネントごとに 1 つの VOB を使用することもできます。これに対し、多くのコンポーネントを使用する場合は、いくつかの VOB にそれぞれ複数のコンポーネントを保存することになります。1 つの VOB に複数のエレメントの複数のバージョンを保存できます。小さいコンポーネントを 1 つだけ保存するために VOB を使用するのとは効率的ではありません。

以下の制約事項に留意してください。

- コンポーネントのルートディレクトリは、VOB のルートディレクトリまたはその 1 レベル下のディレクトリである必要があります。コンポーネントには、そのルートディレクトリにあるすべてのディレクトリ要素とファイル要素が含まれます。たとえば、図 10 では、Libs をコンポーネントにすることはできません。
- コンポーネントをネストすることはできません。たとえば、図 10 で GUI、Admin、Reports がコンポーネントになるのは、Dev がコンポーネントではない場合だけです。
- VOB のルートディレクトリにコンポーネントを作成すると、その VOB に複数のコンポーネントを保存することはできなくなります。そのため、コンポーネントは VOB のルートディレクトリの 1 レベル下のディレクトリに作成することをお勧めします。そうすることによって、VOB にコンポーネントを追加していくことが可能になります。
- コンポーネントを VOB のルートディレクトリに作成する場合でも、その 1 レベル下のディレクトリに作成する場合でも、コンポーネント名はその PVOB 内で一意である必要があります。

図 10 VOB への複数コンポーネントの保存



追加のコンポーネントの指定

システムアーキテクチャを調べることで、必要なほとんどすべてのコンポーネントを把握することができます。しかし、以下のようなコンポーネントをいくつか見落とす可能性があります。

システム コンポーネント	システム レベルのファイルを格納するためのコンポーネントを 1 つ指定することをお勧めします。その中に、プロジェクト計画書、要求文書、システム モデル ファイル、そのほかのアーキテクチャ文書を含めます。
プロジェクト ベースライン コンポーネント	プロジェクトのすべてのコンポーネントからベースラインを選択する複 合ベースラインを使用する場合は、複合ベースラインを固有のコンポー ネントに保存することをお勧めします。詳細については、40 ページの 「プロジェクト ベースラインの指定」を参照してください。
テスト コンポーネント	システムのテストに関連するファイルは別のコンポーネントに格納する ことを検討します。その中に、テスト スクリプト、テスト結果、テスト ログ、テスト文書などを含めます。
展開 コンポーネント	開発サイクルの終わりに、システムと共に出荷または社内に展開するた めに生成されたファイルを格納する、別のコンポーネントが必要になり ます。これらのファイルには、実行可能ファイル、ライブラリ、イン ターフェイス、ユーザー文書が含まれます。
ツール コンポーネント	ソース ファイルだけでなく、チームの開発ツール (コンパイラなど) やシ ステム ヘッダー ファイルもバージョン管理の対象にすることをお勧めし ます。

ディレクトリ構造の定義

コンポーネントのリストを完成した後で、それらのコンポーネント内のディレクトリ構造を定義する必要があります。最初は表 1 に示すようなディレクトリ構造をお勧めします。その後、システムのニーズに合わせてディレクトリ構造を変更します。

表 1 で、コンポーネント_1 からコンポーネント_n は、システム アーキテクチャ内の一連の論理パッケージに対応するコンポーネントを指します。

表 1 コンポーネントの推奨ディレクトリ構造

コンポーネント	ディレクトリ	一般的な内容
システム	plans	プロジェクト計画、役割や責任の記述など
	requirements	要求文書
	models	Rose ファイル、そのほかのアーキテクチャ文書
	documentation	システム文書

表 1 コンポーネントの推奨ディレクトリ構造

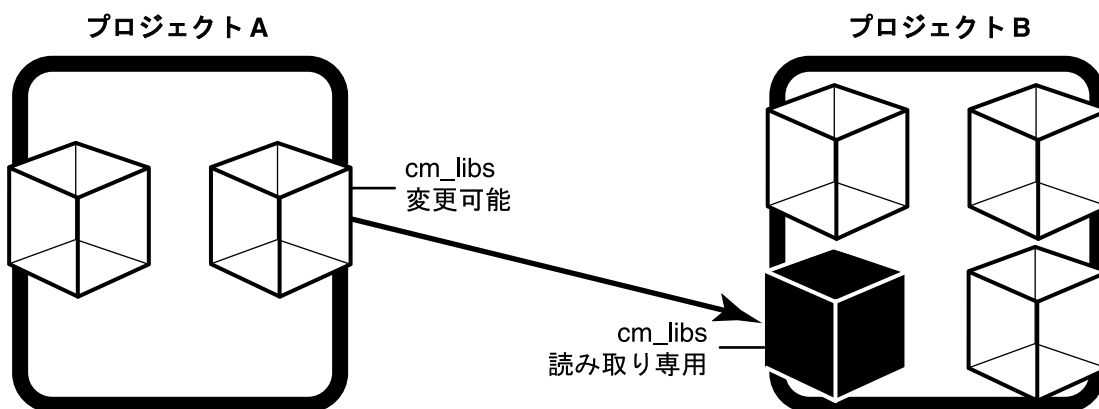
コンポーネント	ディレクトリ	一般的な内容
コンポーネント <u>1</u> から コンポーネント <u>n</u>	requirements	コンポーネントの要件
	models	コンポーネントのモデル ファイル
	source	コンポーネントのソース ファイル
	interfaces	コンポーネントのパブリック インターフェイス
	binaries	コンポーネントの実行可能 ファイルとその他のバイナリ ファイル
	libraries	コンポーネントで使用するラ イブラリ
	tests	コンポーネントのテスト スク リプトと関連文書
テスト	scripts	テスト スクリプト
	results	テスト結果とログ
	documentation	テスト文書
展開	binaries	展開する実行可能ファイル
	libraries	展開するライブラリ
	interfaces	展開するインターフェイス
	documentation	ユーザー文書
ツール	compilers	Visual InterDev や Rational Rose などの開発ツール
	headers	システム ヘッダー ファイル
プロジェクト ベースライン	none	プロジェクト内のすべての コンポーネントからベース ラインを選択する複合ベー スライン

読み取り専用コンポーネントの識別

プロジェクトの作成時に、そのプロジェクトのコンテキスト内で各コンポーネントを変更できるかどうかを選択する必要があります。ほとんどの場合、コンポーネントを変更可能にします。しかし、コンポーネントを読み取り専用にして、そのエレメントをプロジェクトチームのメンバーが変更できないようにする場合もあります。コンポーネントは複数のプロジェクトで使用することができます。したがって、1つのプロジェクトチームがコンポーネントを保守する責任を負い、ほかのプロジェクトチームはそのコンポーネントを使用してほかのコンポーネントの作成のみを行います。

たとえば、図 11 では、プロジェクト A チームのメンバーが一連のライブラリ ファイルを持っています。プロジェクト B チームのメンバーは、自分たちのコンポーネントをビルドするときに、それらのライブラリの一部を参照します。プロジェクト A ではコンポーネント `cm_libs` を変更可能です。プロジェクト B では同じコンポーネントが読み取り専用です。コンポーネント `cm_libs` に関しては、プロジェクト A とプロジェクト B は、「作成者と使用者」の関係になります。

図 11 読み取り専用コンポーネントの使用



ストリーム方針の選択

基本的な UCM プロセスでは、プロジェクトの唯一の共有作業空間としてインテグレーション ストリームを使用します。開発者は、インテグレーション ストリームの推奨ベースラインの内容を自分の開発ストリームに取り込むことによってプロジェクトに参加し、作業結果をインテグレーション ストリームにデリバーします。デリバーされた作業結果を統合担当者が新しいベースラインに取り込み、開発者は自分の開発ストリームを新しい推奨ベースラインにリベースします。プロジェクトのサイズと開発者の人数によっては、このプロセスがプロジェクトチームにとって適切な選択になります。

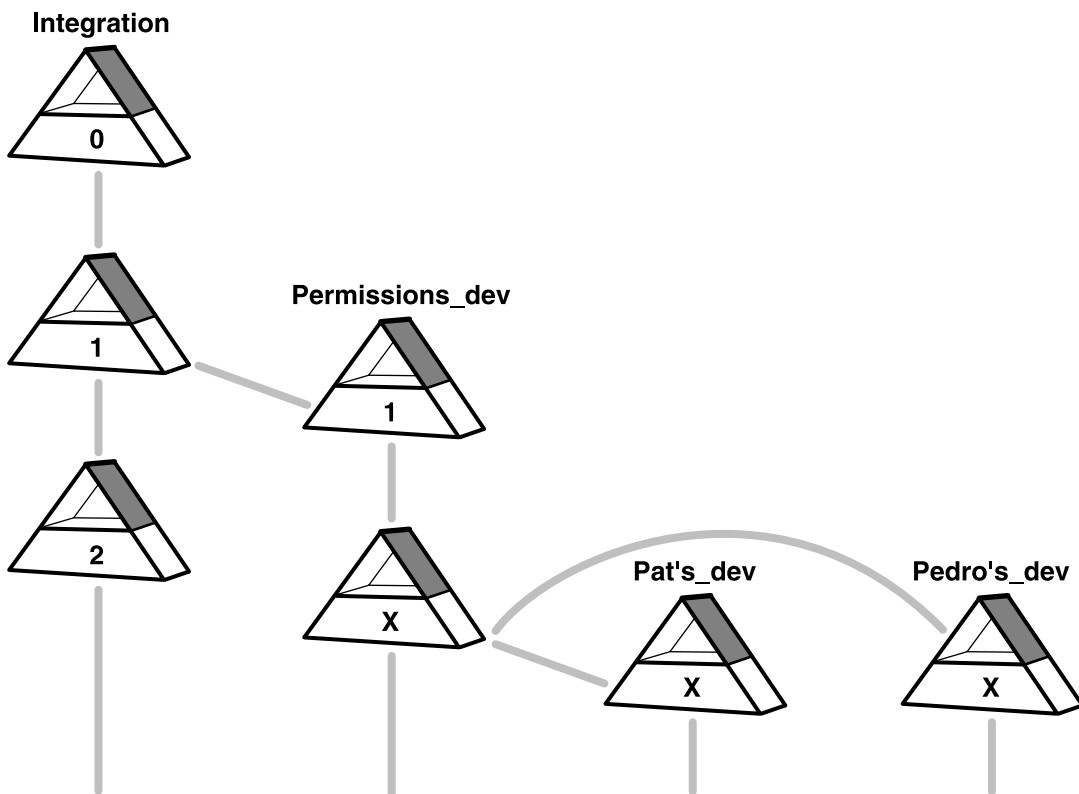
ストリーム階層

また、UCM の開発ストリーム階層機能を使用して、1 つのプロジェクトに複数の共有作業空間を作成することもできます。この方法では、小人数の開発者チームをいくつか結成し、それぞれのチームで特定の機能を開発するというプロジェクト編成が可能です。

たとえば、図 12 では、権限機能を担当する 2 人の開発者のためにプロジェクト マネージャーが **Permissions_dev** という開発ストリームを作成しました。開発者の **Pat** と **Pedro** は、インテグレーション ストリーム レベルではなく **Permissions_dev** レベルでプロジェクトに参加しています。彼らは作業結果を **Permissions_dev** ストリームにデリバリーします。統合担当者または **Permissions_dev** ストリームの管理を担当する開発者がデリバリーされた作業結果を定期的にまとめて新しいベースラインを作成し、開発者は自分の開発ストリームを新しいベースラインにリベースします。

権限機能の開発が完了したら、2 人の開発者は最後の作業結果を **Permissions_dev** ストリームにデリバリーします。統合担当者は、2 人の開発者がデリバリーした作業結果を最終的なベースラインにまとめ、それをインテグレーション ストリームにデリバリーします。

図 12 機能固有の開発ストリームの使用



単一ストリーム プロジェクト

ほとんどの開発チームでは、個人用作業空間と共有作業空間で構成される並行開発環境で十分です。ただし、複数の小規模な開発者チームが緊密な共同作業を行う場合は、直列開発環境の方が通常は適しています。UCM では、単一ストリーム プロジェクトを作成することによって、直列開発環境を実装できます。単一ストリーム プロジェクトとは、インテグレーション ストリーム 1 つだけからなるプロジェクトであり、ユーザーが開発ストリームを作成することはできません。開発者が単一ストリーム プロジェクトに参加するには、インテグレーション ストリームにアタッチされるビューを作成します。

開発者間でコードの迅速な共有が必要になる開発の初期の段階では、単一ストリーム プロジェクトを使用します。開発作業の規模が大きくなり、並行開発環境が必要になった場合には、単一ストリーム プロジェクトの最終的なベースラインを基にして複数ストリーム プロジェクトを作成します。

単一ストリーム プロジェクトの主な利点は以下のとおりです。

- ・ 開発者が動的ビューで作業している場合は、ある開発者がファイルをチェックインすると、ほかの開発者はすぐにその作業結果を見ることができます。開発者がスナップショットビューで作業している場合は、ある開発者がファイルをチェックインすると、ほかの開発者はビューを更新することによって、すぐにその作業結果を見ることができます。複数ストリーム プロジェクトでは、各開発者はデリバーとリベースの操作中のみほかの開発者の変更内容を見ることができます。
- ・ 開発者の作業環境が簡素化されます。すべての作業をインテグレーション ストリームで行うため、開発者は開発ストリームと 2 つのビュー (開発ストリームにアタッチされたビューとインテグレーション ストリームにアタッチされたビュー) を操作する必要がありません。さらに、開発者はデリバーやリベースを実行する必要もありません。
- ・ 統合担当者の仕事が簡素化されます。すべての開発者が同じストリームで作業し、ほかの開発者が行った変更をすぐに確認できるため、ベースラインを頻繁に作成する必要がなくなります。複数ストリーム プロジェクトとは異なり、開発者は作業結果を統合するためにベースラインを使用する必要がありません。単一ストリーム プロジェクトでのベースラインの第一の目的は、主要なマイルストーンを示すことです。

単一ストリーム プロジェクトの主な短所は以下のとおりです。

- ・ ファイル共有のサポートが制限されます。ClearCase では複数の開発者が同一ストリームのエレメントを同時にチェックアウトできますが、チェックアウトを予約できるのは 1 人の開発者だけです。予約済みチェックアウトによって、開発者がエレメントの新しいバージョンをチェックインする権利が保証されます。チェックアウトを予約した開発者以外は、エレメントを非予約としてチェックアウトしなければなりません。つまり、予約済みチェックアウトがチェックインまたはキャンセルされるまで自分のバージョンをチェックインすることはできません。非予約チェックアウトを行った開発者は、自分の変更内容を予約済みチェックアウトによって実行された変更とマージする必要があります。

- 開発者がファイルをチェックインすると変更がすぐに共有可能になるため、開発者は作業結果をテストし、プロジェクトにバグが組み込まれないように十分注意する必要があります。これに対し、複数ストリーム プロジェクトでは、統合担当者またはソフトウェア品質保証チームが新しいベースラインを専用のテスト ストリームで徹底的にテストし、合格した場合のみベースラインを推奨することができます。
- 開発者がファイルをチェックインすると、すぐに変更が共有可能になるため、ファイルをチェックアウトしている時間が複数ストリーム プロジェクトの場合より長くなる可能性があります。ビューが失われると、そのビューで行い、まだチェックインしていないすべての変更も失われます。そのため、単一ストリーム プロジェクトでは、ビューを頻繁にバックアップするよう ClearCase 管理者に依頼することをお勧めします。

読み取り専用ストリーム

プロジェクトの規模が大きくなってくると、何人かのユーザーにベースラインへのアクセスを与え、しかもそれらのユーザーがコンポーネントを変更できないようにしたい場合があります。このようなユーザーのために、読み取り専用の開発ストリームを作成することができます。読み取り専用ストリームにベースラインを作成したり、読み取り専用ストリームの下位に子ストリームを作成することはできません。読み取り専用ストリームには、派生オブジェクトなどのビュープライベート ファイルを作成できます。

読み取り専用ストリームの一般的な使用目的には次のようなものがあります。

- 品質保証チームは特定の構成をビルドし、テストする必要があります。
- カスタマ サポート チームは前のリリースのライブラリにアクセスする必要があります。
- リリース管理チームは、新旧のベースラインを組み合わせるリリースを作成する必要があります。

ベースライン ポリシーの指定

プロジェクトのコンポーネントを編成した後で、コンポーネントのベースラインの作成に関連するポリシーを決定します。以下の事項の定義が必要になります。

- プロジェクト ベースライン
- ベースラインを作成するポイント
- ベースラインの命名規則
- 一連のプロモーション レベル
- ベースラインのテスト方法

プロジェクト ベースラインの指定

統合担当者は、開発者がプロジェクトに参加するときや開発ストリームをリベースするときに、どのベースラインを使用すべきかを知らせます。各コンポーネントに対応するベースラインのリストを作成することもできます。ただし、複合ベースラインを使用してプロジェクト ベースラインを表す方が効率的です。複合ベースラインは、ほかのコンポーネントからベースラインを選択したものです。

たとえば、図 13 では、**プロジェクト A** が複合ベースライン **PA** を使用して **GUI** コンポーネントと **Admin** コンポーネントのベースラインを選択します。**プロジェクト B** でも複合ベースライン **PB** を使用しています。複合ベースラインによって選択されたベースラインを メンバといいます。

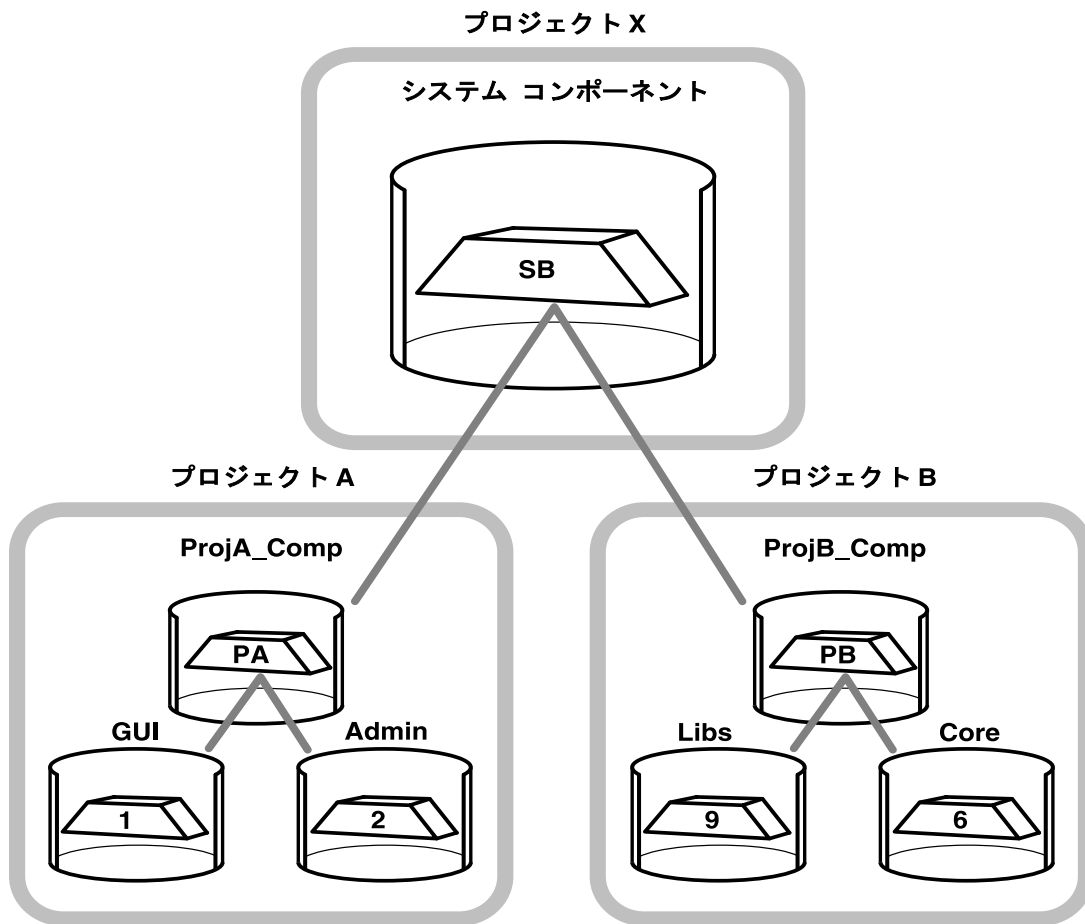
プロジェクト ベースラインを表す複合ベースラインを作成した場合、次にそのプロジェクト ベースラインを含むコンポーネントに対して実行するベースライン作成操作は再帰的に行われます。最新のベースラインが作成された後で複合ベースラインを構成するベースラインを含むコンポーネントが変更された場合は、そのコンポーネントに新しいベースラインが作成されます。たとえば、統合担当者がベースライン 1 を作成した後で、開発者が **GUI** コンポーネント内のファイルに変更を加えたとします。その後、新しいプロジェクト ベースラインを作成すると、変更されたファイルがある **GUI** コンポーネント内に新しいベースラインが作成され、新しいプロジェクト ベースラインはその **GUI** ベースラインを選択します。

複合ベースラインがほかの複合ベースラインを選択することもできます。たとえば、複数のプロジェクトからなる大規模なシステムの場合は、複合ベースラインを使用してシステム ベースラインを表現したい場合もあります。図 13 では、**SB** が複合ベースラインであり、**プロジェクト A** のベースライン **PA** と **プロジェクト B** のベースライン **PB** を選択します。

複合ベースラインを使用してプロジェクトを表すだけでなく、同じプロジェクト内で複数の複合ベースラインを使用することもできます。複数の複合ベースラインを扱う場合、2 つの複合ベースラインで同じコンポーネントの異なるベースラインが選択されることがあります。このような場合には、メンバベースラインのいずれか 1 つを選択し、競合を解決する必要があります。このような競合を回避するために、複合ベースラインを複雑な階層構造にするのではなく、単純なベースライン設計を選択することをお勧めします。ベースライン競合の詳細については、120 ページの「ベースラインの競合の解決」を参照してください。

通常のベースラインと同様、複合ベースラインもコンポーネントに属している必要があります。ただし、そのコンポーネントが固有のエレメントを含んでいる必要はありません。たとえば、図 13 のコンポーネント **System_component**、**ProjA_Comp**、**ProjB_Comp** は、それぞれの複合ベースラインだけで構成されています。複合ベースラインを含むことだけを目的としたコンポーネントを作成する場合は、**VOB** にルート ディレクトリを作成せずにコンポーネントを作成するオプションを選択できます。このようなコンポーネントには、固有のエレメントを追加することはできません。

図 13 システム レベルの複合ベースラインの使用



ベースラインを作成するポイント

プロジェクトの開始時に、新しい開発の開始点となる 1 つまたは複数のベースラインを指定する必要があります。プロジェクトの作業が進展するにつれて、新しいベースラインを定期的に作成する必要があります。

初期ベースラインの指定

既存のプロジェクトの新しいバージョンを作成する場合、既存のプロジェクトのコンポーネントに属す最新の推奨ベースラインから作業を開始するようお勧めします。たとえば、Transaction Builder プロジェクトのバージョン 3.2 の作業を開始する場合、バージョン 3.1 のコンポーネントのリリース済みまたは実稼動しているバージョンを指定します。

ベース ClearCase 構成をプロジェクトに変換する場合、既存のラベル付きのバージョンからベースラインを作成することができます。最新の安定したバージョンにラベルが付けられているかどうかをチェックします。まだラベルが付けられていない場合、ラベル タイプを作成して、プロジェクトの対象に含めるバージョンにそのラベルを適用する必要があります。ラベルタイプの作成と適用の詳細については、94 ページの「ラベルからのベースラインの作成」を参照してください。

後続のベースライン

開発者が新しいプロジェクトの作業を開始し変更を始めた後は、インテグレーション ストリームと機能固有の開発ストリームに、定期的に (毎晩または毎週など) ベースラインを作成します。この作業には以下の利点があります。

- 開発者の作業結果の同期が保たれる

優れた構成管理には、開発者が個人用作業空間を保持し、一連のファイルを対象に独立して作業できることが非常に重要です。しかし、独立して作業している期間が長くなると、問題が発生します。デリバーされた変更が新しいベースラインに組み込まれ、各自の開発ストリームをリベースするまで、開発者はほかの開発者の作業結果を知ることはできません。

- バージョンのマージに必要な時間が最少化される

開発ストリームをリベースしたときに、新しいベースラインに選択されているファイルと各自の個人用作業空間内の作業結果との間にマージの矛盾があれば、開発者はそれを解決する必要があります。ベースラインを頻繁に作成していれば、含まれる変更は少ないので、開発者がバージョンのマージに費やす時間が少なくて済みます。

- 統合上の問題が早期に検出される

ベースラインを作成する際には、前回のベースライン作成以後にデリバーされた作業結果を組み込んで、まずプロジェクトをビルドしてテストします。ベースラインの作成を頻繁に行うことにより、開発者が間違っってプロジェクトに持ち込む可能性のある重大な問題を検出できる可能性が高まります。問題を早期に検出できれば、問題の特定と修正に要する作業量を最少化することができます。

単一ストリーム プロジェクトで作業を行っている場合は、ベースラインを頻繁に作成する必要はありません。開発者はほかの開発者が行った変更をファイルのチェックイン後すぐに確認でき、最新の推奨ベースラインにリベースする必要はありません。単一ストリーム プロジェクトにおけるベースラインの第一の目的は、反復の終了やベータ リリースなどの主要なプロジェクトマイルストーンを示すことです。

命名規則の定義

ベースラインはプロジェクト管理の重要なツールであるため、わかりやすい命名規則を使用してください。ベースライン名には、以下の項目を含めます (すべての項目を使用することもできます)。

- プロジェクト名
- 開発スケジュールのマイルストーンまたはフェーズ
- 作成日

例: V4.0TRANS_BL2_June12.

UCM には、ベースライン命名規則を実装する際に使用できるテンプレートがいくつか用意されています。詳細については、86 ページの「ベースライン命名テンプレートの設定」を参照してください。

開発の状態を反映するプロモーション レベルの設定

プロモーション レベルとは、ベースラインの品質または安定性を示すために使用する、ベースラインの属性です。ClearCase には以下のデフォルトのプロモーション レベルが用意されています。

- Rejected
- Initial
- Built
- Tested
- Released

デフォルトのプロモーション レベルの一部または全部を使用することも、独自に定義することもできます。レベルは最低から最高の品質への移行を反映するように配列します。プロモーション レベルは、開発者に対してベースラインを推奨する際に役立ちます。[推奨ベースライン] ダイアログ ボックスには、プロモーション レベルが指定したレベル以上のベースラインが表示されます。プロモーション レベルによって、このダイアログ ボックスに表示されるベースラインのリストをフィルタすることができます。プロジェクトに適用する一連のプロモーション レベルと各レベルの設定基準を定義する必要があります。

ベースラインのテスト方法の計画

ソフトウェア開発チームは通常、何レベルかのテストを行います。検証テストと呼ばれる初期のテストでは、ソフトウェアにエラーがなく正しく機能することをチェックします。リグレッションテストのようなさらに総合的なテストは長い時間がかかり、通常はソフトウェア品質保証技術者のチームによって実行されます。

新しいベースラインを作成するときには、開発者が別の変更をデリバーしないように、インテグレーション ストリームをロックする必要があります。それによって、一連の静的なファイルをビルドしてテストすることができます。検証テストはそれほど時間がかからないので、インテグレーション ストリームを長時間ロックする必要はありません。しかし、広範なテストを行う場合には、範囲に応じて長い時間がかかります。

インテグレーション ストリームを長時間ロックすることは、よい方法ではありません。ロック中に開発者が完成した作業結果がデリバーされなくなるからです。この問題の 1 つの解決方法は、広範なテストに専用の開発ストリームを作成することです。検証テストに合格済みの新しいベースラインを作成した後で、テスト チームは指定されたテスト用の開発ストリームを新しいベースラインに合わせて、リベースします。ベースラインが次のテスト レベルに合格したら、そのベースラインをプロモートします。ベースラインが十分に安定していると判断した場合、そのベースラインを推奨ベースラインとして、開発者が開発ストリームをリベースできるようにします。

テスト用の開発ストリームの作成の詳細については、105 ページの「ベースライン テスト用開発ストリームの作成」を参照してください。ストリームのテストの詳細については、117 ページの「ベースラインのテスト」を参照してください。

PVOB の計画

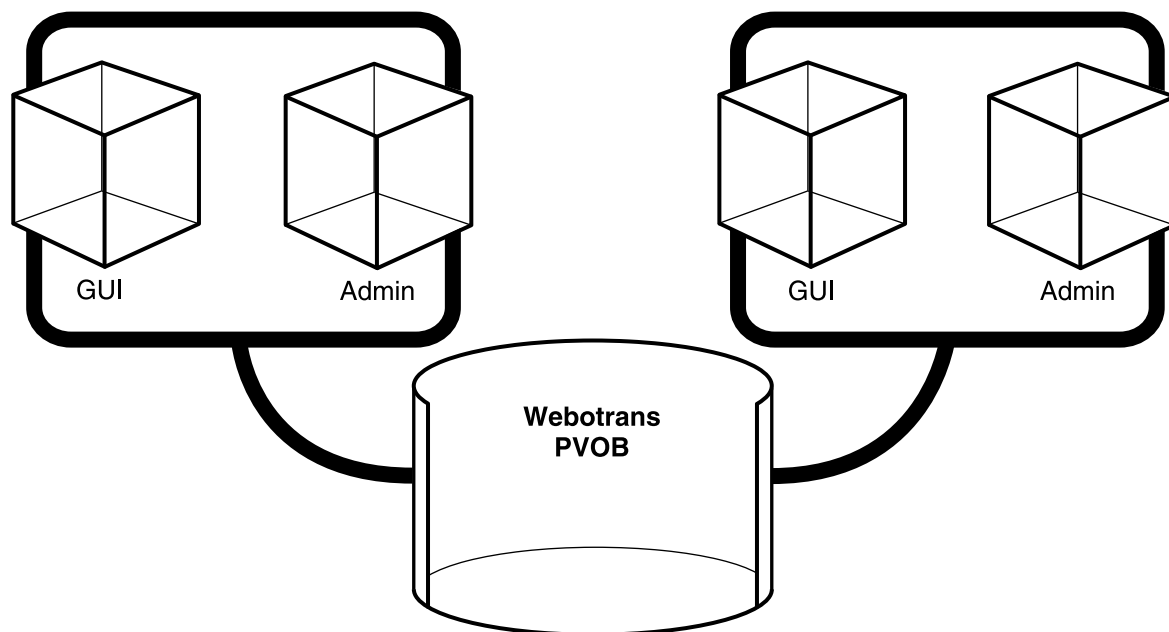
ClearCase ではプロジェクト、ストリーム、アクティビティ、変更セットなどの UCM のオブジェクトをプロジェクト VOB (PVOB) に格納します。PVOB を管理 VOB として使用することもできます。システムでいくつの PVOB を使用するか、PVOB の管理機能を使用するかどうかを決定する必要があります。

使用する PVOB 数の決定

製品メモ: この項の内容は Rational ClearCase LT には当てはまりません。Rational ClearCase LT では各サーバーで 1 つの PVOB しか使用できないからです。

同じ PVOB を使用する複数のプロジェクトからは、同じ一連のコンポーネントにアクセスすることができます。異なるプロジェクトに属する開発者が共に同じコンポーネントの一部を対象に作業する必要がある場合、それらのプロジェクトに 1 つの PVOB を使用します。たとえば、図 14 は Webotrans 製品の 2 つのバージョンを並行開発する様子を示しています。プロジェクト チームの大部分のメンバーはあるプロジェクト内のリリース 4.0 を対象に作業していますが、少数のメンバーは別のプロジェクト内のリリース 4.0.1 を対象に作業しています。両方のプロジェクトで同じコンポーネントを使用するので、同じ PVOB を使用しています。

図 14 関連プロジェクトでの PVOB の共有



PVOB の容量が問題になるほどプロジェクトの規模が大きい場合のみ、複数の PVOB の使用を検討してください。

管理 VOB の役割の理解

管理 VOB にはグローバル タイプの定義を格納します。**AdminVOB** ハイパーリンクを通じて管理 VOB に結合された VOB は、各 VOB 内で定義することなく、同じタイプ定義を共有します。たとえば、管理 VOB 内でエレメントタイプ、属性タイプ、ハイパーリンクタイプなどを定義することができます。その管理 VOB にリンクされたすべての VOB は、それらのタイプ定義を使用して、エレメント、属性、ハイパーリンクを作成することができます。

現時点で管理 VOB を使用している場合は、それを PVOB と関連付けることができます。この関連付けを行うには、管理 VOB と PVOB の間に **AdminVOB** ハイパーリンクを作成します。**Windows** では、VOB 作成ウィザードを実行すると自動的に **AdminVOB** ハイパーリンクが作成されます。**UNIX** では、**cleartool mkhlink** コマンドを使用して **AdminVOB** ハイパーリンクを作成します。その後、コンポーネントを作成すると、そのコンポーネントのルートディレクトリがある VOB と管理 VOB の間に **AdminVOB** ハイパーリンクが作成され、そのコンポーネントで管理 VOB のグローバルタイプ定義を使用できるようになります。

現時点で管理 VOB を使用していない場合は、管理 VOB を作成する必要はありません。コンポーネントを作成すると、そのコンポーネントのルート ディレクトリがある VOB と管理 PVOB の間に AdminVOB ハイパーリンクが作成され、PVOB が管理 VOB の役割を果たすようになります。

管理 VOB とグローバル タイプの詳細については、『Rational ClearCase 管理ガイド』を参照してください。

複数の PVOB の使用法

すべてのプロジェクトで 1 つの PVOB を使用することをお勧めしますが、組織によっては複数の PVOB を使用する場合もあります。ある PVOB 内のプロジェクトで別の PVOB 内のコンポーネントを変更する必要がある場合は、ClearCase 管理者がいずれか 1 つの PVOB を PVOB とコンポーネント VOB の共通管理 VOB として指定します。

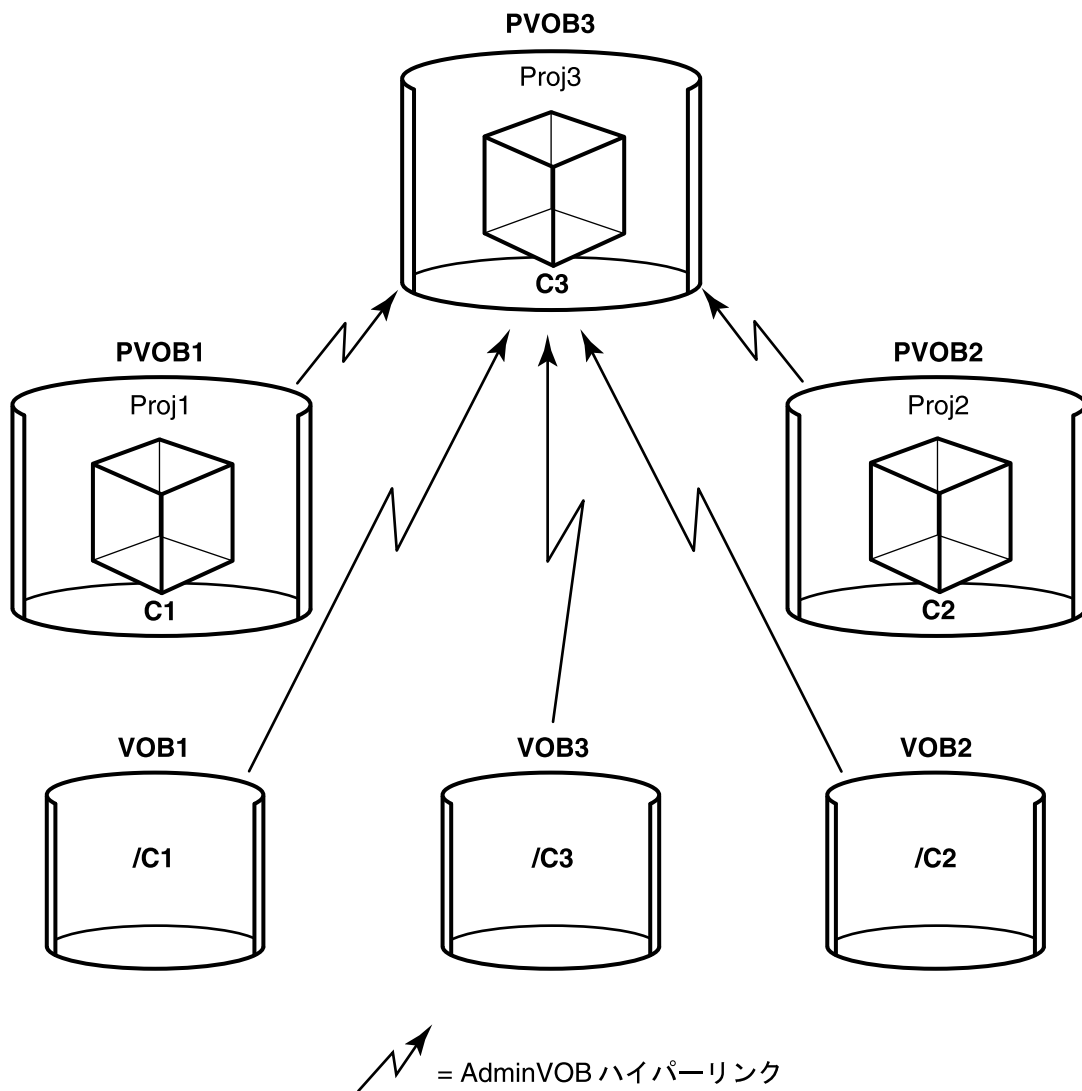
図 15 では、PVOB1 と PVOB2 が PVOB3 を管理 VOB として使用しています。PVOB とコンポーネント VOB から出ている矢印は、PVOB3 への AdminVOB ハイパーリンクを表します。コンポーネント VOB と PVOB は共通管理 VOB を共有しているため、すべてのプロジェクトが 3 つのコンポーネントすべてを変更できます。

PVOB が共通管理 VOB を共有していない場合は、あるプロジェクトで別の PVOB からコンポーネントを選択することはできますが、そのプロジェクトでは選択したコンポーネントが読み取り専用になります。

図 15 に示す PVOB は管理 VOB として使用されています。PVOB とコンポーネント VOB を管理 VOB にリンクするという方法もあります。この方法が適しているケースとしては、開発チームがベース ClearCase から UCM へ移行しており、現在、管理 VOB を使用している場合があります。

複数の PVOB を使用する場合は、管理 VOB として使用する PVOB を最初に作成します。残りの PVOB を作成するときには、最初に作成した PVOB を管理 VOB として指定します。

図 15 ある 1 つの PVOB を複数の PVOB の管理 VOB として使用する



特殊なエレメント タイプの指定

エレメント タイプの概念を利用すると、ClearCase でのエレメントの処理をクラスごとに變えることができます。エレメント タイプはファイル エレメントのクラスです。ClearCase には定義済みのエレメント タイプが用意されています (`file` や `text_file` など)。ユーザーが独自にエレメント タイプを定義することもできます。UCM プロジェクトで使用するためにエレメント タイプを作成するとき、`mergetype` 属性を指定することができます。この属性は、デリバリーとリベースの操作でそのエレメント タイプのファイルをマージする方法を決定します。

ClearCase でデリバリー操作またはリベース操作の間にマージが必要になると、ClearCase はそのエレメントのバージョンのマージを試みます。ClearCase がバージョン間の相違を調整できない場合にのみ、ユーザーの介入を要求されます。あるファイル タイプに関しては、異なるマージ操作を行う場合もあります。

非マージ エレメント

ファイル タイプの中にはマージの必要がないものがあります。そのようなファイルに関しては、だれかが誤ってマージしないよう注意が必要です。たとえば、展開またはステージング用のコンポーネントには、顧客に出荷するかまたは社内にインストールする実行可能ファイルが含まれます。これらのファイルの開発は終わっており、プロジェクト サイクルの開発フェーズの成果物と考えられます。このようなタイプのファイルに関しては、エレメント タイプを作成して、マージ動作に `never` を指定することができます。

メモ: このようなエレメントに `never` マージ動作を指定しなければ、開発者が作業結果をプロジェクトのインテグレーション ストリームへデリバリーするときに問題が発生する可能性があります。開発者はデリバリーする前に、実行可能ファイルをビルドしてテストします。それらのファイルが派生オブジェクトとしてバージョン管理されている場合、それらは現在のアクティビティの変更セットに含められます。デリバリー操作の間に、ClearCase はそれらの実行可能ファイルをインテグレーション ストリームにマージしようとします。ただし、そのファイルがマージ動作に `never` が指定されているエレメント タイプに属していれば、マージは行われません。

非自動マージ エレメント

ファイル タイプの中には、ClearCase が自動的にマージするのではなく、手作業でマージした方がよいものがあります。その一例として、Visual Basic のフォーム ファイルが挙げられます。これは生成されたテキスト ファイルです。Visual Basic は、開発者が Visual Basic GUI 中に作成したフォームに基づいて、フォーム ファイルを生成します。ClearCase のマージ操作によって自動的にフォーム ファイルを変更するよりも、Visual Basic GUI からフォーム ファイルを生成し直す方が便利です。

このタイプのファイルに関しては、エレメント タイプを作成して、マージ動作に `user` を指定することができます。エレメント タイプの作成については、「第 15 章 エレメント タイプを使用したファイル エレメントの処理のカスタマイズ」と『Rational ClearCase リファレンス ガイド』の `mkeltype` リファレンス ページを参照してください。

エレメント タイプの適用範囲の定義

エレメント タイプを定義するとき、その適用範囲を通常またはグローバルにすることができます。デフォルトの設定では、エレメント タイプの適用範囲は通常です。つまり、そのエレメント タイプはそれが含まれる VOB 内でのみ利用可能です。管理 VOB 内でエレメント タイプを作成しその適用範囲をグローバルと定義すると、その管理 VOB に AdminVOB ハイパーリンクされているほかの VOB もそのエレメント タイプを使用できます。グローバルなエレメント タイプを定義する必要がある、現在は管理 VOB を使用していない場合は、PVOB 内でエレメント タイプを定義します。

UCM と ClearQuest の統合の使用法の計画

UCM と ClearQuest の統合の設定を行う前に、いくつかの事項を決定する必要があります。それらは以下の 2 つの一般的なカテゴリに分けられます。

- PVOB を ClearQuest のユーザー データベースにマッピングする方法
- ClearQuest のユーザー データベースに適用するスキーマ

ClearQuest のユーザー データベースへの PVOB のマッピング

ここでは、ClearQuest のユーザー データベースにリンクするプロジェクトで使用する PVOB の数を決定する際に考慮すべき 3 つの点について説明します。

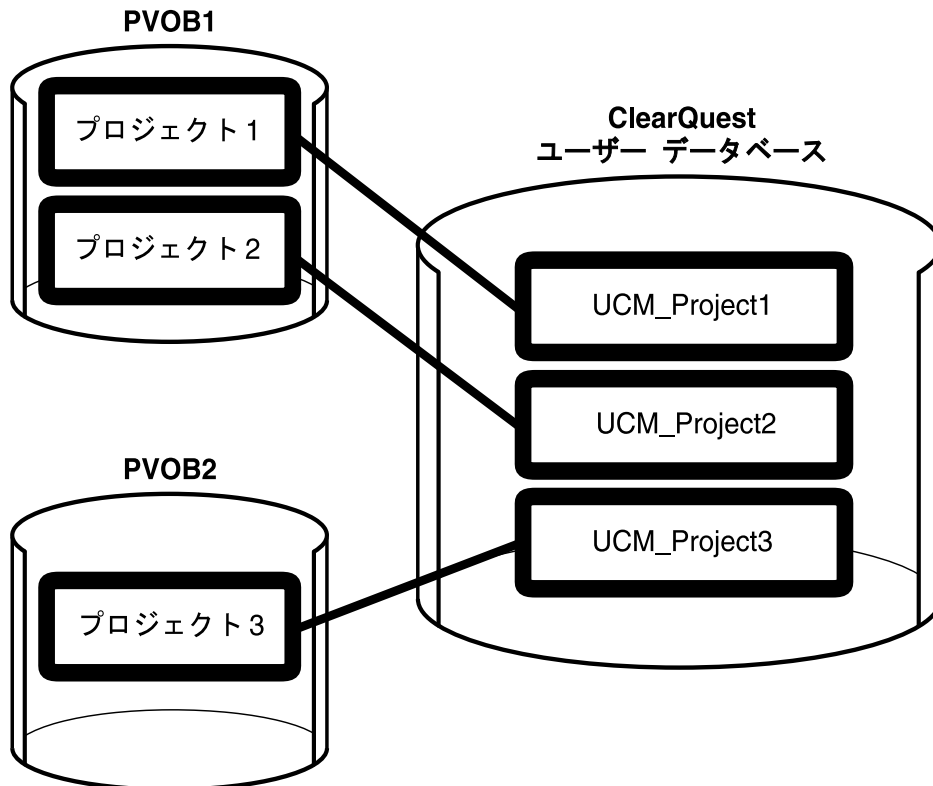
MultiSite の要件

ClearCase MultiSite を使用する場合、PVOB のすべてのレプリカが ClearQuest のユーザー データベースにアクセスできる必要があります。複数の PVOB を管理 VOB または管理 VOB の役割をする PVOB にリンクしている場合は、階層内のすべての PVOB と管理 VOB のレプリカをすべてのサイトに作成する必要があります。

同じデータベースにリンクされるプロジェクトには一意の名前を付ける

UCM では、個別の PVOB 内に格納されているプロジェクトに同じ名前を付けることが許されています。しかし、それらのプロジェクトを同じ ClearQuest のユーザー データベースにリンクすることはできません。この命名の要件を図 16 に示します。

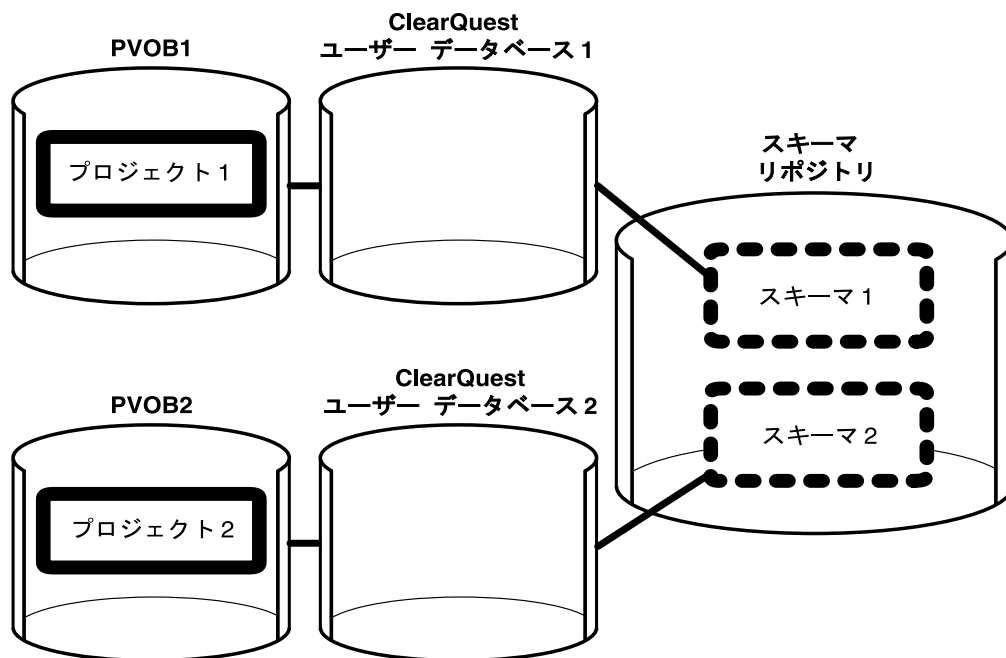
図 16 同一の ClearQuest データベースにリンクされた複数の PVOB 内のプロジェクト



リンクしているデータベースへの 1 つのスキーマ リポジトリの適用

チームに属する開発者の中に複数のプロジェクトの担当者がある場合、それらのプロジェクトにリンクされる ClearQuest のユーザー データベース用のスキーマを 1 つのスキーマ リポジトリに格納するようにお勧めします。その様子を図 17 に示します。このようにすると、開発者はプロジェクトを容易に切り替えることができます。スキーマを個別のスキーマ リポジトリに格納した場合、プロジェクトを切り替える際に別のスキーマ リポジトリに接続するには、開発者は ClearQuest メンテナンス ツールを使用する必要があります。

図 17 複数の ClearQuest データベースへの同スキーマ リポジトリの適用



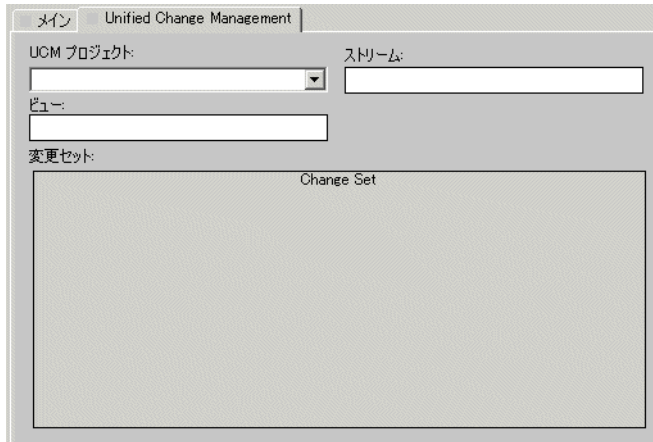
適用するスキーマの決定

UCM と ClearQuest の統合を使用するには、UCM に適用可能なスキーマに基づいた ClearQuest のユーザー データベースを作成またはアップグレードする必要があります。UCM に適用可能なスキーマとは以下の要件を満たすものです。

- スキーマには **UnifiedChangeManagement** パッケージが適用されています。パッケージには特定の機能を定義するレコード、フィールド、状態などのメタデータが含まれています。スキーマにパッケージを適用すると、機能を素早く追加することができ、機能を最初から開発する必要はありません。
- 少なくとも 1 つのレコード タイプに **UnifiedChangeManagement** パッケージが適用されています。このパッケージを使用すると、レコード タイプにフィールドとスクリプトを追加し、レコード タイプのフォームに [Unified Change Management] タブを追加することができます。図 18 に [Unified Change Management] タブを示します。
- スキーマには **UCMPolicyScripts** パッケージが適用されています。このパッケージには ClearQuest の 3 つの開発ポリシーが含まれています。それらを実施することができます。

ClearQuest には UCM に適用可能なスキーマがあらかじめ 2 つ用意されています。具体的には、UnifiedChangeManagement と Enterprise です。このどちらかを使用することによって、UCM と ClearQuest の統合の使用を直ちに開始することができます。または、ClearQuest Designer と ClearQuest パッケージ ウィザードを使用して、独自のスキーマまたは定義済みの別のスキーマを UCM に適用することができます。UCM に適用可能な定義済みのスキーマのどれか 1 つを開始点として使用して、後で必要に応じて変更することもできます。

図 18 UCM に適用可能なレコード タイプ用のレコード フォームの [Unified Change Management] タブ



UnifiedChangeManagement スキーマの概要

UnifiedChangeManagement スキーマには以下のレコード タイプがあります。

- **BaseCMActivity**
軽量のレコード タイプで、追加のフィールドを必要としないアクティビティに関する情報の格納に使用できます。BaseCMActivity レコード フォームの [メイン] タブを図 19 に示します。このレコード タイプを開始点として使用して、後でフィールドと状態を追加することができます。
- **Defect**
このレコード タイプは DefectThis のほかの定義済みスキーマ中の同じ名前のレコード タイプと基本的に同じです。ただし、ClearQuest に適用できます。Defect レコード タイプには BaseCMActivity レコード タイプよりも多くのフィールドとフォーム タブがあり、詳細な情報を記録できます。
- **UCMUtilityActivity**
このレコード タイプは、通常は使用しません。アクティビティが含まれるプロジェクトを ClearQuest のユーザー データベースにリンクしたときのように、UCM と ClearQuest の統合のためにレコードを生成する必要がある場合に、このレコード タイプを使用します。このレコード タイプをユーザーが変更することはできません。

図 19 BaseCMActivity レコードタイプ用のレコードフォームの[メイン]タブ

The screenshot shows a web-based record form for 'BaseCMActivity'. The form is titled 'Unified Change Management' and has a 'メイン' (Main) tab selected. The form contains several input fields: 'ID:' with a text box, '所有者:' (Owner) with a dropdown menu, '状態:' (Status) with a text box, '見出し:' (Header) with a text box, and '説明:' (Description) with a large text area.

UCM へのスキーマの適用

UCM に適用可能な定義済みのスキーマを使用しないことに決めた場合、スキーマを UCM に適用できるようにするために追加の作業が必要になります。その前に、以下の質問に答える必要があります。

- どのレコードタイプを UCM に適用可能にするか。スキーマ内のすべてのレコードタイプを適用可能にする必要はありません。UCM に適用可能なレコードタイプのレコードだけをアクティビティにリンクします。
- UCM に適用可能な各レコードタイプについて
 - 各状態をどの状態タイプにマッピングするか。各状態を 4 つの UCM の状態タイプのどれかにマッピングする必要があります。具体的には、**Waiting**、**Ready**、**Active**、**Complete** のどれかにマッピングします。71 ページの「状態タイプの設定」を参照してください。
 - ある状態から別の状態への遷移レコードにどのデフォルトのアクションを適用しているか。72 ページの「レコードタイプに関する状態遷移のデフォルトアクションの要件」を参照してください。
 - どのポリシーを実施するか。統合時に使用したポリシーを選択し設定して、開発作業で実施することができます。ポリシーのスクリプトを編集して、その内容を変更することもできます。詳細については、「第 4 章 ポリシーの設定」を参照してください。

UCM には開発ポリシーがいくつか用意されています。その中から選択して、プロジェクト内の開発作業で実施することができます。それらのポリシーの中には、プロジェクトに **Rational ClearQuest** を適用する場合にのみ利用可能なものもあります。UCM に用意されたポリシーに加え、UCM 操作に設定されたトリガを使用して独自のポリシーを作成することもできます。トリガの詳細については、「第 8 章 トリガを使用した開発ポリシーの実施」を参照してください。

コンポーネントとベースライン

ここでは、コンポーネントとベースラインに関連するポリシーについて説明します。

変更可能なコンポーネント

ほとんどの場合、コンポーネントは変更可能にします。コンポーネントを読み取り専用にした方がよい場合については、36 ページの「読み取り専用コンポーネントの識別」を参照してください。

ベースラインを推奨する際のデフォルトのプロモーション レベル

推奨ベースラインとは、プロジェクト チームのメンバーが自分たちの開発ストリームをリベースする際に使用する、一連のベースラインです。開発者がプロジェクトに参加したとき、その開発作業空間が推奨ベースラインで初期化されます。ベースラインを推奨するときに使用する [推奨ベースライン] ダイアログ ボックスには、プロモーション レベルがベースラインを推奨する際のデフォルトのプロモーション レベル以上である最新のベースラインが表示されます。

デフォルトのビュー タイプ

プロジェクトに参加したときに、開発者はプロジェクトへの参加ウィザードを使用して、自分たちの開発ビュー、インテグレーション ビュー、開発ストリームを作成します。開発者は開発ビューと開発ストリームを使用して、プロジェクト チームのほかのメンバーから独立して作業を行います。ビルドした作業結果ををほかの開発者によってインテグレーション ストリームまたは機能固有の開発ストリームにデリバーされた最新の作業結果に基づいてテストするときには、インテグレーション ビューを使用します。

Rational ClearCase には、動的ビューとスナップショット ビューの 2 種類のビューがあります。どちらのビューを開発ビューとインテグレーション ビューのデフォルトとして使用するかを決定します。開発者がプロジェクトに参加するときには、デフォルトのビュー タイプを受け付けるか、拒否するかを選択できます。開発者が最初にプロジェクトのビューを作成するときには、プロジェクトに参加ウィザードではデフォルト値が使用されます。それ以降は、このウィザードでは開発者が最後に選択したビューがデフォルトのビュー タイプになります。

製品メモ: Rational ClearCase LT はスナップショット ビューのみサポートします。

動的ビューでは、ClearCase のマルチバージョン ファイル システム (MVFS) を使用して、VOB 内に格納されているファイルとディレクトリに対する迅速で透過的なアクセスを提供します。Windows では、ClearCase は動的ビューを Windows エクスプローラ内のドライブ文字にマッピングします。スナップショット ビューは VOB 内のファイルとディレクトリをユーザーのコンピュータ上のディレクトリにコピーしたものです。

インテグレーション ビューのデフォルトのビュー タイプには動的ビューを使用するようお勧めします。動的ビューを使用すると、開発者はインテグレーション ストリームまたは機能固有の開発ストリームに作業結果をデリバリーする際に、前回のベースラインが作成された後ではかの開発者によってデリバリーされた最新の作業結果に基づいて、ビルドした作業結果をテストすることができます。スナップショット ビューを使用する場合は、デリバリーされた最新のファイルとディレクトリを開発者が自分のコンピュータにコピーする必要があります (スナップショット更新操作)、この操作は忘れがちです。

プロジェクトとストリームを変更する権限

ここでは、プロジェクトとストリームのオブジェクトを変更できるユーザーを決定するポリシーについて説明します。

すべてのユーザーにプロジェクトの変更を許可する

デフォルトでは、このポリシーは無効に設定されています。つまり、プロジェクト オブジェクトに対して変更を加えることができるのは、プロジェクトの所有者、PVOB の所有者、権限を持つユーザーだけです。すべてのユーザーにプロジェクト オブジェクトの変更を許可するには、このポリシーを有効にします。

すべてのユーザーにストリームとストリームのベースラインの変更を許可する

デフォルトでは、このポリシーは無効に設定されています。つまり、ストリームとストリーム内のベースラインに対して変更を加えることができるのは、ストリームの所有者、PVOB の所有者、権限を持つユーザーだけです。すべてのユーザーにストリームとそのベースラインの変更を許可するには、このポリシーを有効にします。このポリシーは、プロジェクト内のすべてのストリームに適用することもできれば、ストリームごとに設定することも可能です。

デリバー操作

ここでは、デリバー操作に関連するポリシーについて説明します。このポリシーは、プロジェクト内のすべてのストリームに適用することもできれば、ストリームごとに設定することも可能です。開発者がデリバー操作を開始すると、ターゲット ストリームとプロジェクトのポリシー設定がチェックされます。ターゲット ストリームとプロジェクトのポリシー設定が異なる場合は、プロジェクトの設定が優先されます。

未処理のチェックアウトがあるストリームからのデリバーを許可

このポリシーでは、チェックアウトしたままのファイルが開発ストリームにあっても、開発者は作業結果をターゲット ストリームにデリバーできます。このポリシーを設定していない場合、開発者はソース ストリーム内のすべてのファイルをチェックインしてからでなければ、作業結果をデリバーすることはできません。以下の状況を避けるために、開発者にファイルをチェックインするように要求します。

- 1 開発者がアクティビティの作業を完了した後、そのアクティビティに関連するファイルをチェックインし忘れます。
- 2 その開発者がほかのアクティビティの作業を行います。
- 3 いくつかのアクティビティを完了した後で、開発者はそれらをターゲット ストリームにデリバーします。最初のアクティビティに関連するファイルはチェックアウトされたままなので、それらはデリバー操作に含まれません。開発者が開発作業空間内で変更を正しくビルドしてテストしても、ターゲット ストリームにデリバーされた変更はエラーを発生させる可能性があります。チェックアウトしたファイルが含まれていないからです。

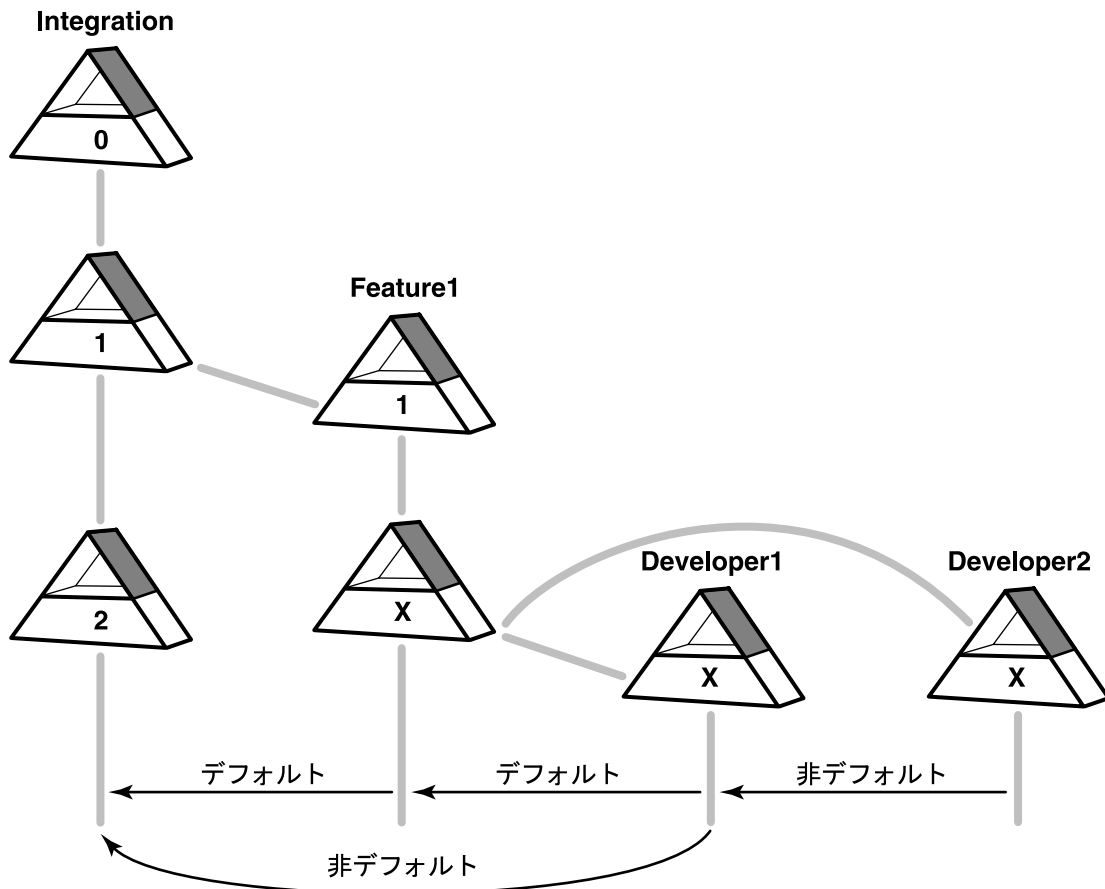
デリバー前のリベース

このポリシーでは、開発者は自分の作業結果をターゲット ストリームにデリバーする前に、ターゲット ストリームの現在の推奨ベースラインに合わせて自分のソース ストリームをリベースする必要があります。このポリシーの目的は、開発者が開発作業空間で作業結果をビルドし、最新の安定したベースラインに含まれている作業結果に基づいてテストしてから、ターゲット ストリームにデリバーすることです。この方法を取ると、開発者がデリバー操作を行うときに必要なマージの作業量が最小限で済みます。

デフォルト以外のターゲットへのデリバー操作

図 20 に示すように、開発ストリームの階層を作成することができます。このような階層によって、特定の機能を担当する開発者の共有作業空間として開発ストリームを指定することができます。その機能を担当する開発者は、作業結果を機能固有の開発ストリームにデリバーします。

図 20 ストリーム階層におけるデフォルトとデフォルト以外のデリバリー ターゲット



ストリーム階層では、ストリーム間に祖先と子孫の関係が確立されます。図 20 では、インテグレーションストリームと開発ストリーム Feature1 が開発ストリーム Developer1 と Developer2 の祖先です。Feature1、Developer1、Developer2 は、インテグレーション ストリームの子孫です。ストリームの直接の祖先は親ストリームです。ストリームの直接の子孫は子ストリームです。

プロジェクトに複雑な開発ストリーム階層がいくつも存在する場合、作業結果が開発ストリームから多くのターゲット ストリームへとデリバリーされることがあります。さらに、ストリームから別のプロジェクトのストリームへと作業結果がデリバリーされる場合もあります。開発ストリームからのデリバリー操作のデフォルト ターゲットは、同じストリームの親ストリームです。開発者がデフォルト以外のターゲット ストリームに作業結果をデリバリーすることもあります。図 20 内の矢印はデフォルトとデフォルト以外のデリバリー ターゲットを示しています。以下のポリシーはデフォルト以外のターゲット ストリームにのみ適用されます。

別プロジェクトのプロジェクトまたはストリームへのデリバーを許可する

このポリシーは、ストリームがほかのプロジェクトのストリームからのデリバーを受け付けるかどうかを指定します。別のプロジェクトに作業結果をデリバーする必要があるケースについては、「第 9 章 複数プロジェクトの並行リリースの管理」を参照してください。

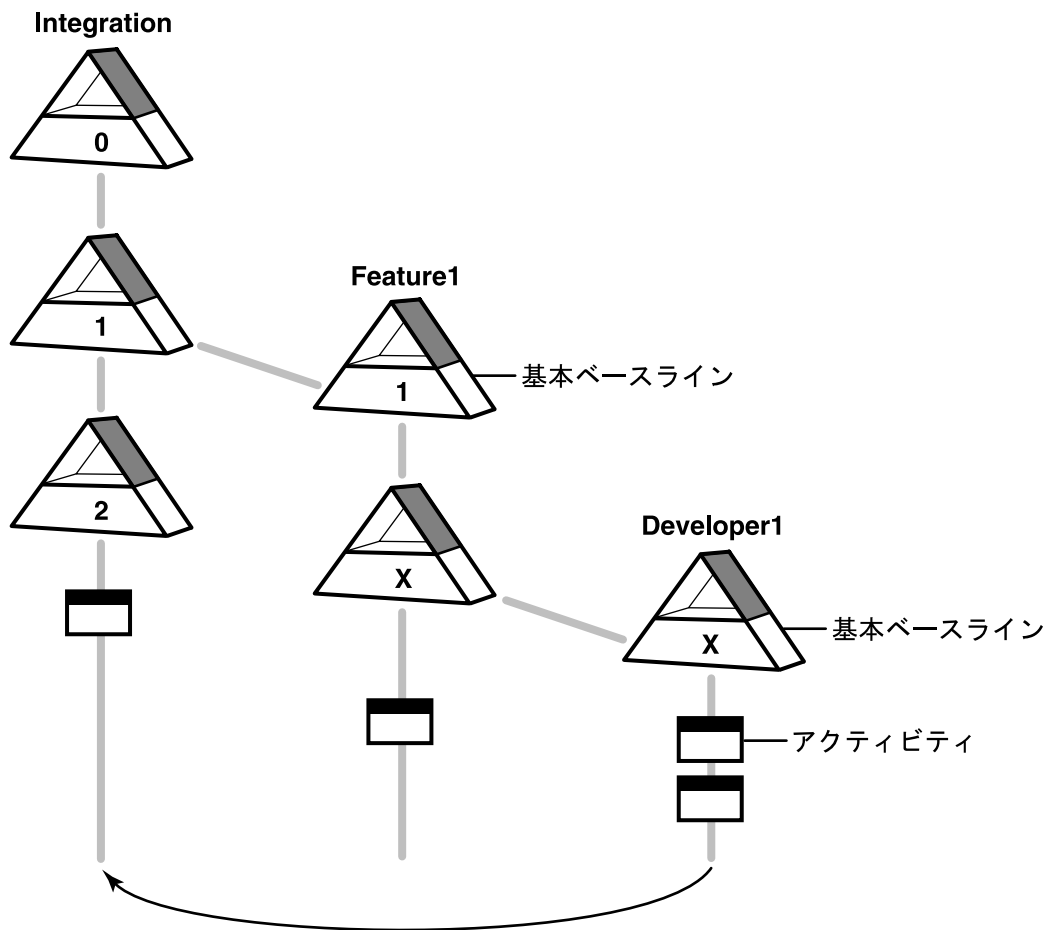
ストリームからの変更と基本ベースラインからの変更をデリバーする

UCM では、基本ベースラインを使用してストリームのビューを構成します。ストリームにタッチされたビューには、そのストリームの基本ベースラインに示されたエレメント バージョンに加え、そのストリームに作成されたアクティビティに関連付けられたエレメント バージョンが表示されます。たとえば、図 21 では、1 が開発ストリーム **Feature1** の基本ベースラインです。ベースライン **X** は開発ストリーム **Developer1** の基本ベースラインです。

Developer1 ストリームで作業している開発者が作業結果をインテグレーション ストリームにデリバーする場合には、**Developer1** ストリームで作成されたアクティビティと基本ベースライン **X** に示されたファイルがデリバーの対象になります。このインテグレーション ストリームを担当する統合担当者は、場合によっては **Developer1** ストリームで作業している開発者の作業結果を受け取る必要があります。その場合、統合担当者はデリバーされた作業結果に **X** ベースラインで加えられた変更が含まれていることに気づかない可能性があります。必要であれば、このポリシーを [無効] に設定し、ターゲット ストリームがソース ストリームの基本ベースラインに加えられた変更を含むデリバー操作を受け付けないようにします。

UCM では、このポリシーに 2 つのバージョンがあります。1 つはプロジェクト間のデリバー操作、もう 1 つはプロジェクト内のデリバー操作です。

図 21 基本ベースラインで行われた変更のデリバー



ターゲット ストリームがソース ストリームに存在するコンポーネントを含んでいない場合でもデリバーを許可する

このポリシーは、ストリームがターゲット ストリーム以外のコンポーネントに加えられた変更を含むデリバーを受け付けるかどうかを指定します。このポリシーを [有効] に設定すると、デリバー操作は可能ですが、欠落したコンポーネントに加えられた変更はデリバーに含まれません。UCM では、このポリシーに 2 つのバージョンがあります。1 つはプロジェクト間のデリバー操作、もう 1 つはプロジェクト内のデリバー操作です。

ソース ストリームの変更可能なコンポーネントがターゲット ストリームでは変更可能でない場合でもデリバーを許可する

このポリシーは、ストリームがターゲット ストリームのプロジェクトで変更不可能なコンポーネントに加えられた変更を含むプロジェクト間デリバーを受け付けるかどうかを指定します。このポリシーを [有効] に設定すると、デリバー操作は可能ですが、変更不可能なコンポーネントに加えられた変更はデリバーに含まれません。

UCM と ClearQuest の統合

ここでは、プロジェクトに ClearQuest を適用するようにした場合にのみ利用可能なポリシーについて説明します。ここで説明するポリシーにはカスタマイズが可能なものもあります。ClearQuest では、スクリプトを使用してカスタマイズ可能なポリシーを実装します。そのスクリプトを編集することによって、ポリシーの動作を変更することができます。74 ページの「ClearQuest のプロジェクト ポリシーのカスタマイズ」を参照してください。

作業前に ClearQuest のアクションを実行

このポリシーは、開発者がアクティビティを設定しようとしたときに呼び出されます。デフォルトのポリシー スクリプトでは、開発者のユーザー名が ClearQuest レコードの [Owner] フィールドの名前と一致しているかどうかを確認されます。両者が一致する場合、その開発者はそのアクティビティの作業を行うことができます。一致しない場合、設定したアクティビティアクションはエラーになります。

このポリシーの意図は、開発者がアクティビティの作業を開始する前に、すべての基準が満たされていることを確認することです。このポリシーを変更して、チェックの対象にする基準を追加することもできます。

デリバー前に ClearQuest のアクションを実行

このデフォルトのポリシー スクリプトはプレースホルダーであり、実際には何もしません。UCM に ClearQuest を使用可能なプロジェクト内で開発者がアクティビティをデリバーしようすると、ClearCase によってこのポリシーが呼び出されます。このスクリプトを変更して承認プロセスを実装し、デリバー操作を管理することをお勧めします。たとえば、アクティビティのレコードタイプに [Approved] チェック ボックスを追加して、プロジェクト マネージャーがそれを選択してからでなければ開発者がアクティビティをデリバーできないようにすることもできます。

2 つの ClearQuest 対応プロジェクト間でのデリバーの詳細については、65 ページの「UCM と ClearQuest の統合におけるプロジェクト間デリバーの処理」を参照してください。

アクティビティ変更前に ClearQuest のアクションを実行

このデフォルトのポリシー スクリプトはプレースホルダーであり、実際には何もしません。このポリシーは、開発者がアクティビティを終了しようとしたときに呼び出されます。アクティビティを終了すると、そのアクティビティの変更セットに所属するすべてのファイルがチェックインされ、設定したポリシーに基づいて、アクティビティの **Complete** 状態への遷移などの ClearQuest アクションが実行されます。アクティビティ終了操作は、単一ストリーム プロジェクト内または複数ストリーム プロジェクトのインテグレーション ストリームで実行した場合、複数ストリーム プロジェクトでのデリバー操作と似たような結果になります。どちらの操作でも、チームのほかのメンバーが変更を共有できます。このスクリプトを変更して承認プロセスを実装し、アクティビティ終了操作を制御することをお勧めします。たとえば、アクティビティのレコード タイプに [Approved] チェック ボックスを追加して、プロジェクト マネージャーがそれを選択してからでなければ開発者がアクティビティを終了できないようにすることもできます。

デリバー後に ClearQuest のアクションを実行

デリバー操作に含まれる各アクティビティごとのデリバー操作の終わりに、このポリシーが ClearCase によって呼び出されます。このデフォルトのポリシー スクリプトはプレースホルダーであり、実際には何もしません。このスクリプトを変更してデリバー後の開発作業を実装することもできます。たとえば、このスクリプトを使用して、デリバー操作が終了したことをプロジェクト内のすべての開発者に知らせる電子メール メッセージを送信することも可能です。

2 つの ClearQuest 対応プロジェクト間でのデリバーの詳細については、65 ページの「UCM と ClearQuest の統合におけるプロジェクト間デリバーの処理」を参照してください。

アクティビティ変更後に ClearQuest のアクションを実行

このポリシーは、開発者がアクティビティを終了しようとしたときに呼び出されます。[アクティビティ変更前に ClearQuest のアクションを実行] ポリシーを設定すると、このポリシーが最初に呼び出されます。デフォルトのポリシー スクリプトの動作は以下のようになります。

- 単一ストリーム プロジェクトで作業している開発者と複数ストリーム プロジェクトのインテグレーション ストリームで作業している開発者に対しては、アクティビティ終了操作を許可します。
- 開発ストリームで作業している開発者に対しては、アクティビティの変更セットに所属するすべてのファイルをチェックインしますが、ClearQuest のアクションは実行しません。

このスクリプトを変更してアクティビティ終了後の開発作業を実装することもできます。たとえば、このスクリプトを使用して、ある開発者がファイルをチェックインしてアクティビティを終了したことをプロジェクト内のすべての開発者に知らせる電子メール メッセージを送信することも可能です。

デリバー後に完了に遷移

デリバー操作に含まれる各アクティビティごとのデリバー操作の終わりに、このポリシーが **ClearCase** によって呼び出されます。このポリシーでは、アクティビティのデフォルト アクションを使用して、アクティビティが **Complete** 状態に遷移します。デフォルトのアクションではアクティビティのレコードのいくつかのフィールドにデータが入力されている必要があり、フィールド空の場合は、このスクリプトによってエラーが返され、デリバー操作は **uncompleted** 状態になります。それによって、開発者は別のデリバー操作を行うことができなくなりますが、現在の操作には影響しません。また、バージョンのマージ中に行われた変更はロールバックされません。

エラーから復帰するには、開発者は、アクティビティのレコード内の必須フィールドを設定し、デリバー操作を再開する必要があります。開発者がデリバー操作を GUI から呼び出した場合は、**ClearQuest** レコード フォームが表示されるので、そのフィールドを設定できます。

このポリシーはカスタマイズできません。

2 つの **ClearQuest** 対応プロジェクト間でのデリバーの詳細については、65 ページの「UCM と **ClearQuest** の統合におけるプロジェクト間デリバーの処理」を参照してください。

アクティビティ変更後に完了に遷移

このポリシーは、アクティビティ終了操作の終了時に呼び出されます。このポリシーでは、アクティビティのデフォルト アクションを使用して、アクティビティが **Complete** 状態に遷移します。デフォルトのアクションで、アクティビティのレコードの特定のフィールドに入力が必要だった場合、このフィールドが空の場合、このポリシーはエラーを返します。このエラーから復帰するには、開発者がアクティビティのレコード内の必須フィールドを設定する必要があります。

開発者がインテグレーション ストリームで作業しているかどうかに基づいて、アクティビティを **Complete** 状態に遷移することができます。

- 単一ストリーム プロジェクト内または複数ストリーム プロジェクトのインテグレーション ストリームで作業している開発者についてのみアクティビティの状態を遷移するには、このポリシーと [アクティビティ変更後に **ClearQuest** のアクションを実行] ポリシーを設定します。
- 開発者がどのストリームで作業しているかにかかわらずアクティビティの状態を遷移するには、このポリシーを設定し、[アクティビティ変更後に **ClearQuest** のアクションを実行] ポリシーの設定を解除します。

このポリシーはカスタマイズできません。

デリバー前に ClearQuest のマスターシップを転送

デリバー操作が正常に完了すると、[デリバー後に完了に遷移] プロジェクト ポリシーによって、アクティビティが **Complete** 状態に遷移します。このポリシーが **MultiSite** 環境で正しく動作するためには、デリバーされるアクティビティをマスター登録したレプリカがターゲット ストリームをマスター登録したレプリカと同じでなければなりません。必ずこのようにマスター登録するためには、[デリバー前に **ClearQuest** のマスターシップを転送] ポリシーを設定します。

[デリバー前に **ClearQuest** のマスターシップを転送] ポリシーの動作は、デリバー操作がローカルかリモートかによって異なります。デリバー操作がローカルの場合、つまりターゲット ストリームがローカル **PVOB** レプリカによってマスター登録されている場合、すべてのアクティビティがローカルにマスター登録されなければ、このポリシーによってデリバー操作が失敗します。

リモート **PVOB** レプリカによってマスター登録されているターゲット ストリームは、リモート デリバー操作の対象になります。開発者がデリバー操作を開始しても、その操作は **posted** 状態のままになります。リモート サイトの統合担当者が、デリバー操作を完了します。

リモート デリバー操作は、[デリバー前に **ClearQuest** のマスターシップを転送] ポリシーにより、次のように動作します。

- デリバー操作に含まれるすべてのアクティビティがリモート レプリカによってマスター登録されている場合は、**ClearCase** でデリバー操作を進行できます。
- ローカル レプリカによってマスター登録されたアクティビティがデリバー操作に含まれている場合、**MultiSite** により、それらのアクティビティのマスターシップがリモート レプリカに転送されます。リモート サイトの統合担当者が必要なマージを実行し、デリバー操作を完了した後で、それらのアクティビティのマスターシップがローカル レプリカへ戻されるようにするには、[デリバー後に **ClearQuest** のマスターシップを転送] プロジェクト ポリシーも設定します。
- リモート、ローカル以外のレプリカによってマスター登録されたアクティビティがデリバー操作に含まれる場合、デリバー操作は失敗します。

このポリシーはカスタマイズできません。

2 つの **ClearQuest** 対応プロジェクト間でのデリバーの詳細については、65 ページの「**UCM** と **ClearQuest** の統合におけるプロジェクト間デリバーの処理」を参照してください。

デリバー後に ClearQuest のマスターシップを転送

このポリシーは、必ず [デリバー前に ClearQuest のマスターシップを転送] ポリシーと共に使用します。[デリバー前に ClearQuest のマスターシップを転送] ポリシーを設定すると、リモートデリバー操作の対象となるアクティビティのマスターシップがローカル レプリカからリモートレプリカに転送されます。リモートサイトの統合担当者がデリバー操作を完了した後で、これらのアクティビティのマスターシップを元の (ローカル) レプリカへ戻したい場合に、このポリシーを設定します。

このポリシーはカスタマイズできません。

2 つの ClearQuest 対応プロジェクト間でのデリバーの詳細については、65 ページの「UCM と ClearQuest の統合におけるプロジェクト間デリバーの処理」を参照してください。

UCM と ClearQuest の統合におけるプロジェクト間デリバーの処理

UCM と ClearQuest を統合した場合、ClearQuest 対応のプロジェクトから別のプロジェクトへデリバーするときには、以下のポリシーは適用されません。ただし、これには 1 つだけ例外があります。

- デリバー前に ClearQuest のアクションを実行
- デリバー後に ClearQuest のアクションを実行
- デリバー後に完了に遷移
- デリバー前に ClearQuest のマスターシップを転送
- デリバー後に ClearQuest のマスターシップを転送

UCM と ClearQuest を統合した場合、ソース ストリームのプロジェクトに設定されたポリシーが適用されるのは、ソース ストリームとターゲット ストリームがインテグレーション ストリームであり、ターゲット ストリームがソース ストリームのデフォルト デリバー ターゲットである場合だけです。

ClearQuest ユーザー データベースの設定

5

本章では、プロジェクトに UCM と ClearQuest を統合して使用するために、ClearQuest のユーザー データベースを設定する方法について説明します。本章に示す手順は、通常 ClearQuest のデータベース管理者によって実行されます。Rational ClearQuest では、UCM にすぐに適用できるように、定義済みのスキーマを用意しています。カスタム スキーマまたは別の定義済みのスキーマを UCM に適用することもできます。UCM と ClearQuest の統合を設定する前に決定すべき事項については、49 ページの「UCM と ClearQuest の統合の使用法の計画」を参照してください。

定義済みの UCM に適用可能なスキーマの使用

定義済みの UCM スキーマには UnifiedChangeManagement と Enterprise があります。これらのスキーマには、UCM プロジェクトを処理するために必要なレコード タイプ、フィールド、フォーム、状態、その他の定義が含まれています。ClearQuest のユーザー データベースを UCM で使用できるように設定するには、次の手順を実行します。

- 1 UCM に適用可能な定義済みのスキーマのどれか 1 つに関連するユーザー データベースを作成します。ClearQuest Designer 内で [データベース]、[新規データベースの作成] をクリックして、新規データベースの作成ウィザードを起動します。
- 2 ウィザードの案内に従って、手順を最後まで実行します。手順 4 では、新しいデータベースに関連付けるスキーマを選択するよう要求するメッセージが表示されます。スキーマ名のリストをスクロールして、新しいスキーマを選択します。その様子を図 22 に示します。
- 3 [完了] をクリックします。

図 22 UCM に適用可能なスキーマとのユーザー データベースの関連付け



UCM に適用するためのスキーマの設定

定義済みの UCM スキーマを使用すると、UCM と ClearQuest の統合を直ちに使用することができます。しかし、プロジェクトのアクティビティと変更依頼を把握するために、スキーマのカスタマイズや、別の定義済みスキーマの使用が必要になることもあります。スキーマを UCM にも適用するには、次の手順を実行します。

- 1 スキーマに **UCM_Project** という名前のレコードタイプが含まれていないことを確認します。このレコードタイプは UCM と ClearQuest の統合によって予約されています。
- 2 ClearQuest Designer 内で、[パッケージ]、[パッケージ ウィザード] をクリックして、パッケージ ウィザードを起動します。
- 3 必要に応じてパッケージ ウィザードを使用し、**BaseCMActivity** パッケージをスキーマに適用します。**BaseCMActivity** パッケージを使用すると、スキーマに **BaseCMActivity** レコードが追加されます。**BaseCMActivity** レコードタイプは軽量のアクティビティ レコードタイプです。最初は **BaseCMActivity** レコードタイプを使用し、その後でフィールドや状態などを追加することをお勧めします。**BaseCMActivity** レコードタイプの名前を変更する場合は、必ずこのタイプのレコードを作成する前に行ってください。
- 4 スキーマに **UnifiedChangeManagement** パッケージを適用します。
[UnifiedChangeManagement] を選択して [次へ] をクリックします。
- 5 ウィザードの 2 ページ目で、スキーマを選択します。[次へ] をクリックします。

- 6 ウィザードの 3 ページ目で、スキーマのレコードタイプを指定するように案内するメッセージが表示されます。有効にしたいレコードタイプのチェックボックスをオンにします。[次へ]をクリックします。選択したすべてのレコードタイプが 70 ページの「カスタムレコードタイプを有効にするための要件」にリストされている要件を満たしている必要があります。
- 7 ウィザードの 4 ページ目で、有効にするように選択した各レコードタイプの状態に状態タイプを割り当てます。各状態について、隣の状態タイプのセルをクリックして、利用可能な状態タイプから 1 つを選択します。その様子を図 23 に示します。別のレコードタイプを有効にするには、レコードタイプのリスト内の矢印をクリックして、利用可能なレコードタイプを表示します。4 つの状態タイプとそれらを設定する規則については、71 ページの「状態タイプの設定」を参照してください。

処理を終えたら、[完了]をクリックして、スキーマをチェックアウトします。

メモ: ウィザードの [状態のタイプ名の設定] ページが表示されないことがあります。その場合、状態に状態タイプを割り当てるには、[パッケージ]、[状態のタイプ名の設定] をクリックします。

図 23 レコードタイプの状態への状態タイプの割り当て

パッケージ ウィザード - 状態のタイプ名の設定

パッケージ (P): UnifiedChangeManagement-2.0

レコード タイプ (R): Defect

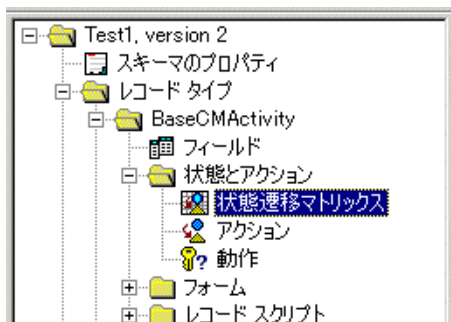
状態のタイプを選択 (S):

状態	状態のタイプ
2 Assigned	Waiting
3 Opened	Active
4 Resolved	Active
5 Closed	Active
6 Duplicate	Waiting
	Ready
	Active
	Complete

< 戻る (B)

- 8 有効にした各レコードタイプの状態に関するデフォルトのアクションを設定してからでないと、スキーマをチェックインすることはできません。デフォルトのアクションとは、開発者がアクティビティの作業またはデリバリーを開始したときに、ClearQuest によって行われる状態遷移のアクションです。ClearQuest Designer の作業空間内で、レコードタイプの状態遷移マトリックスに移動します。その様子を図 24 に示します。

図 24 レコードタイプの状態遷移マトリックスへの移動



[状態遷移マトリックス] をダブルクリックして、マトリックスを表示します。状態列の見出しを右クリックして、ショートカットメニューの [プロパティ] をクリックします。[デフォルト アクション] タブをクリックします。デフォルトのアクションを選択します。デフォルトのアクションの要件については、72 ページの「レコード タイプに関する状態遷移のデフォルト アクションの要件」を参照してください。デフォルトのアクションを設定する前に、レコード タイプにアクションを追加する場合は、[アクション] をダブルクリックしてアクション グリッドを表示し、[編集]、[アクションの追加] をクリックします。

- 9 ClearQuest Designer ワークスペース内でレコードタイプの [動作] まで移動します。[動作] をダブルクリックして、[動作] グリッドを開きます。[Headline] フィールドが [Mandatory] に設定されていることを確認します。[Owner] フィールドがすべてのアクティブな状態タイプで [Mandatory] に設定されていることを確認します。
- 10 [ファイル]、[確認] をクリックして、スキーマの変更を検証します。表示されるエラーを修正し、[ファイル]、[チェックイン] をクリックしてスキーマをチェックインします。
- 11 ユーザー データベースをアップグレードし、[データベース]、[データベースのアップグレード] をクリックしてスキーマの UCM に適用可能なバージョンに関連付けられるようにします。別の方法として、スキーマの UCM に適用可能なバージョンに基づくユーザー データベースを新規に作成することもできます。

カスタム レコード タイプを有効にするための要件

UnifiedChangeManagement パッケージをカスタム レコード タイプに適用する場合、そのレコード タイプは以下の要件を満たしている必要があります。

- SHORT_STRING として定義されている [Headline] という名前のフィールドと ClearQuest の users レコード タイプへの REFERENCE として定義されている [Owner] という名前のフィールドが含まれていること。[Headline] フィールドは長さが 120 文字以上である必要があります。

- 以下の名前のフィールドを含まないこと。
 - ucm_vob_object
 - ucm_stream
 - ucm_stream_object
 - ucm_view
 - ucm_project
- タイプが Modify である **Modify** という名前のアクションが含まれること。
- Submitted という名前の状態が含まれること。この状態の名前は、UnifiedChangeManagement パッケージを適用した後で変更できます。

状態タイプの設定

UCM と ClearQuest の統合では、アクティビティの進捗状況のモニターを支援するために、状態遷移モデルを使用します。このモデルを実装するために、UCM と ClearQuest の統合では UCM に適用可能なスキーマに状態タイプを追加します。表 2 に 4 つの状態タイプを示します。各状態を状態タイプに割り当てる必要があります。各状態タイプについて必ず 1 つの状態を定義する必要があります。

表 2 UCM に適用可能なスキーマの状態タイプ

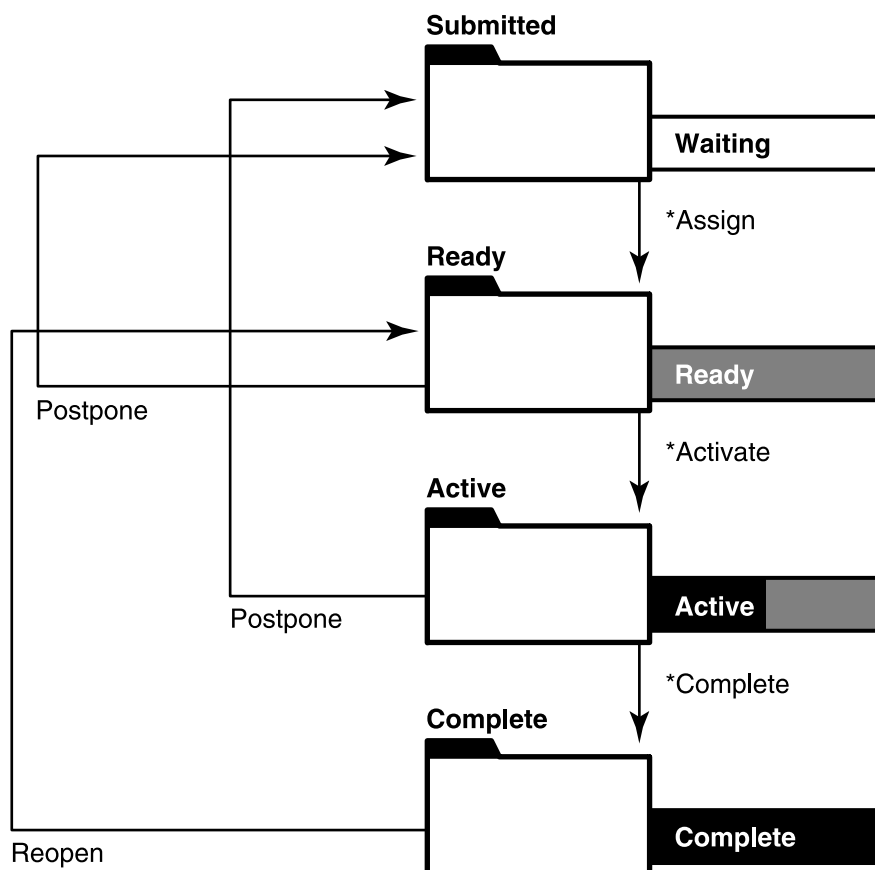
状態タイプ	説明
Waiting	アクティビティは、割り当てられていないか、依存関係が満足されていないため、作業できる状態になっていません。
Ready	アクティビティが割り当てられ、すべての依存関係が満足されて作業できる状態になっています。
Active	開発者はアクティビティの作業を開始しましたが、まだ完了していません。
Complete	開発者はアクティビティの作業を完了したか、作業せずにアクティビティを断念しました。

レコードタイプに関する状態遷移のデフォルトアクションの要件

レコードタイプには多数の状態定義を含めることができます。しかし、UCM に適用可能なレコードタイプには、状態タイプの間を Waiting、Ready、Active、Complete の順に遷移するパスが必ず 1 つ必要です。ある状態から次の状態への遷移はデフォルトのアクションで行われる必要があります。

図 25 に例を示します。これは、定義済み UCM スキーマ内の UCM に適用可能な BaseCMActivity レコードタイプに定義されている状態の間のアクションとデフォルトのアクションです。デフォルトのアクションにはアスタリスク (*) を付けてあります。状態は、Submitted、Ready、Active、Complete です。対応する状態タイプを状態の右側に示してあります。

図 25 UCM に適用可能な BaseCMActivity レコードタイプの状態遷移図



この単一パスの要件に加えて、状態は以下の規則に適合する必要があります。

- すべての **Waiting** 状態に対するデフォルトのアクションは、別の **Waiting** 状態、**Ready** 状態、**Active** 状態のどれかに遷移する必要がある。
- **Ready** 状態に対するアクションの中に **Waiting** 状態に直接戻るものが存在する場合、その **Waiting** 状態に対するデフォルトのアクションは **Ready** 状態に直接遷移する必要がある。
- すべての **Ready** 状態に対するデフォルトのアクションは別の **Ready** 状態または **Active** 状態に遷移する必要がある。
- すべての **Ready** 状態に対しては、**Waiting** 状態に直接遷移するアクションが必ず 1 つ必要である。
- **BaseCMAActivity** レコード タイプの初期の状態は **Waiting** である必要がある。

スキーマの最新 UCM パッケージへのアップグレード

UCM に適用可能な ClearQuest スキーマが ClearQuest の以前のリリースのものである場合は、スキーマを **UnifiedChangeManagement** パッケージの最新バージョンでアップグレードすると、新しい機能を使用できるようになります。スキーマをアップグレードするには、次の手順を実行します。

- 1 **ClearQuest Designer** 内で、[パッケージ]、[インストール済みパッケージのアップグレード] をクリックします。インストール済みパッケージのアップグレード ウィザードが起動します。
- 2 このウィザードの最初のページにアップグレードが必要なパッケージがあるすべてのスキーマが表示されます。アップグレードするスキーマを選択し、[次へ] をクリックします。
- 3 ウィザードの 2 ページ目には、アップグレードするパッケージが表示されます。[アップグレード] をクリックして変更を承認します。
- 4 アップグレードする **UnifiedChangeManagement** パッケージがリリース 3.0 より前の場合は、UCM に適用可能な各レコード タイプについて、状態を状態タイプに割り当てる必要があります。
- 5 [ファイル]、[確認] をクリックして、スキーマの変更を検証します。表示されるエラーを修正し、[ファイル]、[チェックイン] をクリックしてスキーマをチェックインします。
- 6 ユーザー データベースをアップグレードし、[データベース]、[データベースのアップグレード] をクリックしてスキーマの新しいバージョンに関連付けます。

ClearQuest のプロジェクト ポリシーのカスタマイズ

プロジェクトポリシーを実装するために、UCM と ClearQuest の統合では以下のスクリプトのペアを UCM に適用可能なスキーマに追加します。

- UCM_ChkBeforeDeliver と UCM_ChkBeforeDeliver_Def
- UCM_ChkBeforeWorkOn と UCM_ChkBeforeWorkOn_Def
- UCM_CQActAfterDeliver と UCM_CQActAfterDeliver_Def
- UCM_CQActBeforeChact と UCM_CQActBeforeChact_Def
- UCM_CQActAfterChact と UCM_CQActAfterChact_Def

各ポリシーにはスクリプトが 2 つあります。具体的には、ベース スクリプトとデフォルト スクリプトです。デフォルト スクリプトは名前に `_Def` が付いていて、`UnifiedChangeManagement` パッケージによってインストールされます。ベース スクリプトは `UCMPolicyScripts` パッケージによってインストールされ、UCM と ClearQuest の統合によって呼び出されます。ベース スクリプトは対応するデフォルト スクリプトを呼び出します。そのデフォルト スクリプトにデフォルトの動作のロジックが組み込まれています。ポリシーの動作を修正するには、ベース スクリプトからのデフォルト スクリプトの呼び出しを削除し、新しい動作のロジックをベース スクリプトに追加します。ベース スクリプトに記述されていた規則に従います。

各スクリプトには Visual Basic バージョンと Perl バージョンがあります。Visual Basic スクリプトには UCM プレフィックスが付きます。Perl スクリプトには UCU プレフィックスが付きます。Windows NT 上の ClearQuest クライアントには Visual Basic のスクリプトが使用されます。UNIX 上の ClearQuest クライアントには Perl のスクリプトが使用されます。両方のプラットフォームの ClearQuest クライアントを使用している場合、ポリシーの動作を修正するときに、Visual Basic バージョンと Perl バージョンの両方のポリシー スクリプトに同じ修正を加えるようにします。そうしないと、UNIX と Windows NT で ClearQuest クライアントの動作が異なります。

これらのポリシーの詳細については、61 ページの「UCM と ClearQuest の統合」を参照してください。

親アクティビティ レコードと子アクティビティ レコードの関連

プロジェクト マネージャーは、大規模なタスクに関する多くのアクティビティを開発者に割り当てます。開発者はアクティビティを検討して、1 つの大きなアクティビティを完成するために、アクティビティをいくつかに分けて実行することがあります。

たとえば、「顧客検証機能の追加」というアクティビティでは、製品の GUI、コマンド行インタフェース、ライブラリに相当の作業が必要になる可能性があります。アクティビティの進捗を高い精度で把握するには、そのアクティビティを 3 つに分解することができます。

ClearQuest 内で親/子コントロールを使用することによって、分解と子アクティビティの親アクティビティへの関連付けを行うことができます。

親/子コントロールの使用

ClearQuest では、レコード フォーム内にフィールドを表示するために、コントロールを使用します。親/子コントロールを参照または参照リスト フィールドと併用すると、関連するレコードをリンクすることができます。UCM に適用可能なレコード タイプのレコード フォームに親/子コントロールを追加することによって、チームの開発者は親アクティビティをいくつかの子アクティビティに分解することができます。

すべての子アクティビティが完了したときに、親アクティビティの状態を ClearQuest が変更するには、フック プロシージャを記述する必要があります。このようなフックの例は、『Rational ClearQuest 管理ガイド』を参照してください。

ユーザーの作成

プロジェクト チームに属する開発者にアクティビティを割り当てる前に、ClearQuest 内の各開発者のユーザー アカウント プロファイルを設定する必要があります。このためには、以下を実行します。

- 1 ClearQuest Designer で [ツール] メニューの [ユーザー管理] をクリックします。
- 2 [ユーザー アクション]、[ユーザーの追加] をクリックします。
- 3 [ユーザーの追加] ダイアログ ボックスに必要事項を入力します。

ユーザー プロファイルを作成する方法の詳細については、『Rational ClearQuest 管理ガイド』と ClearQuest Designer のヘルプを参照してください。

環境の設定 (UNIX)

ClearQuest のユーザー データベースを UCM プロジェクトにも適用するには、まず表 3 に示す 2 つの環境変数を定義する必要があります。UCM と ClearQuest を統合して使用する開発者もこれらの変数を自分のマシンに定義する必要があります。

ClearQuest のインストール先ディレクトリにある C シェル スクリプト `cq_setup.csh` を実行すると、環境変数を設定することができます。たとえば、次のように指定します。

```
% source ClearQuest-install-directory/cq_setup.csh
```

表 3 統合に必要な環境変数

変数	設定
\$CQ_HOME	ClearQuest-install-directory/releases/ ClearquestClient
\$LD_LIBRARY_PATH (HP-UX では \$SHLIB_PATH)	以下の 2 つを含む必要があります。 ClearCase-install-directory/shlib と ClearQuest-install-directory/releases/ ClearquestClient/architecture/shlib

複数の ClearQuest スキーマ リポジトリがある場合、\$SQUID_DBSET 環境変数を使用して、使用するスキーマ リポジトリの名前を指定する必要があります。

プロジェクトの セットアップ

6

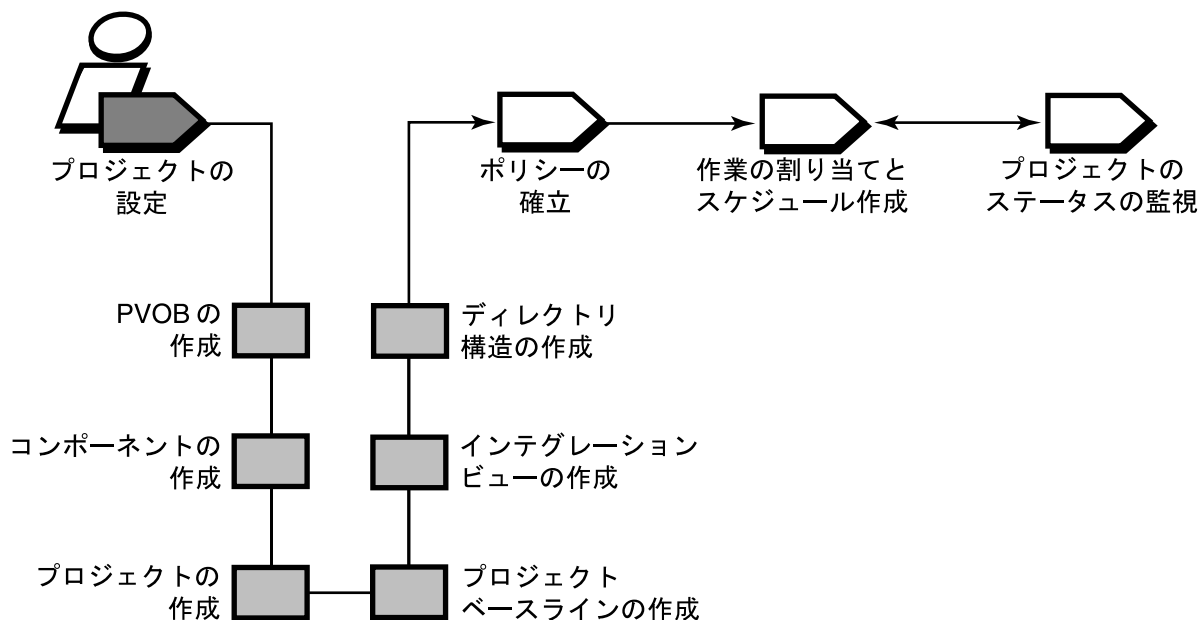
この章では、統一変更管理機能 (UCM) 環境で開発チームが作業できるようにプロジェクトをセットアップする方法について説明します。プロジェクトをセットアップする前に、プロジェクトの計画を立てます。構成管理計画の内容の詳細については、「第3章 プロジェクトの計画」を参照してください。

この章では次のシナリオについて説明します。

- プロジェクトの新規作成
- 既存のベース ClearCase 構成に基づいたプロジェクトの作成
- 既存のプロジェクトに基づいたプロジェクトの作成
- プロジェクトで UCM - ClearQuest 統合を使用可能にする
- Rational Suite を使用した作業
- 新しいベースラインのテスト専用開発ストリームの作成
- 機能固有の開発ストリームの作成

プロジェクトの新規作成

プロジェクト
マネージャー



この項では、既存のプロジェクトや既存の ClearCase VOB セットを使用せずにプロジェクトを新規に作成、セットアップする方法について説明します。

プロジェクト VOB の作成 (Windows)

製品メモ: このタスクは ClearCase LT ユーザーには適用されません。インストール時に ClearCase 管理者が PVOB を作成します。

PVOB を作成するには

- 1 [スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[管理]、[VOB の作成] をクリックします。VOB 作成ウィザードが表示されます。
- 2 VOB 作成ウィザードのステップ 1 で、PVOB の名前を入力します。PVOB の目的を説明するコメントを入力します。[この VOB は UCM コンポーネントを含む] チェックボックスをオフのままにします。1 つの VOB を PVOB として使用し、1 つのコンポーネントを使用することができます。ただし、プロジェクトが非常に小規模であり、かつ今後もプロジェクトの規模が拡大しない場合を除き、VOB をこのように使用しないことをお勧めします。[UCM プロジェクト VOB として作成] チェックボックスをオンにします。

- 3 ステップ 2 で PVOB の記憶ディレクトリを指定します。PVOB 記憶ディレクトリとは、PVOB の内容のリポジトリとして機能するディレクトリ ツリーです。PVOB の記憶ディレクトリには、VOB の記憶ディレクトリと同じサブディレクトリが含まれています (VOB 記憶ディレクトリ構造の詳細については、『Rational ClearCase 管理ガイド』を参照)。推奨される場所を 1 つ選択するか、別の場所を示す UNC (汎用命名規則) パスを入力します。ネットワーク上の共有リソースの場所を検索するには、[参照] をクリックします。
- 4 ステップ 3 では、この PVOB に関連付ける管理 VOB を選択するよう要求されます。リストの先頭までスクロールし、[なし] を選択します。これは現在プロジェクトを新規に作成しており、管理 VOB を使用していないためです。コンポーネントを作成すると、ClearCase によって、そのコンポーネントと PVOB との間に AdminVOB ハイパーリンクが設定され、PVOB が管理 VOB の役割を果たすようになります。

複数の PVOB を作成するときに、今後これらの PVOB のプロジェクトが同一コンポーネントを変更する必要がある場合には、管理 PVOB として機能する PVOB を 1 つ選択し、この PVOB を最初に作成します。ほかの PVOB を作成するときに、ウィザードのステップ 3 で管理 VOB として機能するこの PVOB を指定します。コンポーネントを作成すると、管理 VOB として機能する PVOB とコンポーネントの間に AdminVOB ハイパーリンクが ClearCase により作成されます。複数の PVOB の使用法の詳細については、44 ページの「PVOB の計画」を参照してください。

プロジェクト VOB の作成 (UNIX)

製品メモ: このタスクは ClearCase LT ユーザーには適用されません。インストール時に ClearCase 管理者が PVOB を作成します。

PVOB を作成するには

- 1 cleartool mkvob コマンドを実行します。たとえば、次のように指定します。

```
cleartool mkvob -tag /vobs/myproj2_pvob -nc -ucmproject ¥  
/usr/vobstore/myproj2_pvob.vbs
```

-ucmproject オプションは、VOB ではなく PVOB が作成されることを示します。
/usr/vobstore/myproj2_pvob.vbs パスにより、PVOB の記憶ディレクトリの場所が指定されます。PVOB 記憶ディレクトリとは、PVOB の内容のリポジトリとして機能するディレクトリ ツリーです。PVOB の記憶ディレクトリには、VOB の記憶ディレクトリと同じサブディレクトリが含まれています。VOB 記憶ディレクトリ構造の詳細については、『Rational ClearCase 管理ガイド』を参照してください。

MVFS 環境で開発者が動的ビューを使用する場合には、次の 2 つの手順を実行します。

- 2 PVOB タグに対応する PVOB マウント ポイントを作成します。たとえば、次のように指定します。

```
mkdir /vobs/myproj2_pvob
```

- 3 PVOB をマウントします。たとえば、次のように指定します。

```
cleartool mount /vobs/myproj2_pvob
```

PVOB は管理 VOB の役割を果たします。コンポーネントの作成時に、ClearCase はコンポーネントと PVOB の間に AdminVOB ハイパーリンクを自動的に作成します。

複数の PVOB を作成するときに、今後これらの PVOB のプロジェクトが同一コンポーネントを変更する必要がある場合には、管理 PVOB として機能する PVOB を 1 つ選択し、この PVOB を最初に作成します。ほかの PVOB を作成するときに `cleartool mkhlink` コマンドを使用し、管理 VOB として機能する PVOB と各 PVOB との間に AdminVOB ハイパーリンクを作成します。コンポーネントを作成すると、管理 VOB として機能する PVOB とコンポーネントの間に AdminVOB ハイパーリンクが ClearCase により作成されます。複数の PVOB の使用法の詳細については、44 ページの「PVOB の計画」を参照してください。

プロジェクト ベースラインを保存するコンポーネントの作成

このタスクは省略できますが、実行することを強くお勧めします。プロジェクトを表す方法としては、コンポーネントごとに 1 つのベースラインを追跡する方法よりも、複合ベースラインを使用する方法の方が簡単です。複合ベースラインとエレメントを同じコンポーネントに保存できますが、プロジェクト ベースラインを 1 つの専用コンポーネントに保存すると整然とします。ほかのユーザーがこのコンポーネントにエレメントを作成しないようにするには、VOB ルート ディレクトリのないコンポーネントを作成します。VOB ルート ディレクトリのないコンポーネントには、コンポーネント自体のエレメントを保存できません。VOB ルート ディレクトリのないコンポーネントを作成するには、次の手順を実行します。

- 1 Windows の場合、ClearCase エクスプローラで [UCM] をクリックし、[プロジェクト エクスプローラ] をクリックします。UNIX の場合、コマンド `clearprojexp` を入力してプロジェクト エクスプローラを起動します。プロジェクト エクスプローラは、プロジェクトに関する情報を作成、管理、表示するためのグラフィカル ユーザー インターフェイス (GUI) です。
- 2 プロジェクト エクスプローラの左側のペインには、ローカル ClearCase ドメインのすべての PVOB のフォルダのリストが表示されます。各 PVOB にはそれぞれ独自のルート フォルダがあります。ClearCase は PVOB の名前を使用してルート フォルダを作成します。作成した PVOB に移動します。
- 3 ClearCase は、[コンポーネント] というフォルダも作成します。このフォルダには、PVOB の各コンポーネントのエントリが保存されます。[コンポーネント] フォルダを右クリックし、ショートカットメニューの [新規] をポイントし、[VOB ルートなしのコンポーネント] をクリックします。
- 4 [VOB ルートなしのコンポーネントの作成] ダイアログ ボックスで、コンポーネントの名前と説明を入力します。[OK] をクリックします。

プロジェクトで複数の複合ベースラインを使用することがあります。複数の複合ベースラインを使用する場合には、直接、またはほかの複合ベースラインを介して間接的に、プロジェクト内のすべてのコンポーネントのベースラインを選択する最上位の複合ベースラインを1つ使用することをお勧めします。

エレメントを保存するコンポーネントの作成

この項では、チームが開発したファイルを保存するコンポーネントの作成方法を説明します。

製品メモ: エレメントを保存するコンポーネントの作成手順は、Rational ClearCase と Rational ClearCase LT とでは多少異なります。

コンポーネントを作成するときに、コンポーネントのディレクトリー ツリーを保存する VOB を指定する必要があります。1 つの VOB に複数のコンポーネントを保存するか、1 つのコンポーネントだけを保存する VOB を作成できます。1 つの VOB に複数のコンポーネントを保存する方法の詳細については、32 ページの「使用する VOB 数の決定」を参照してください。

複数のコンポーネントを保存する VOB の作成 (Windows)

ClearCase で複数のコンポーネントを保存できる VOB を作成するには

- 1 VOB 作成ウィザードを起動します。
- 2 ステップ 1 で VOB の名前を入力します。VOB の目的を説明するコメントを入力します。
[この VOB は UCM コンポーネントを含む] チェックボックスをオンにします。
- 3 ステップ 2 で、[この VOB が複数のコンポーネントを含むことを可能にする] オプション ボタンと [次のコンポーネントを VOB に含める] チェックボックスをオンにします。
[ビュー] リストからビューを選択し、[追加] をクリックします。

[コンポーネントの追加] ダイアログ ボックスにコンポーネントの名前とルート ディレクトリを入力し、[OK] をクリックします。ウィザードのリストにコンポーネントが表示されます。別のコンポーネントを作成するには、[追加] をクリックします。コンポーネントの名前は、その PVOB 内で固有でなければなりません。
- 4 ステップ 3 で、VOB を保存する場所を指定します。推奨されている場所を 1 つ選択するか、別の場所の UNC パスを入力できます。ネットワーク上の共有リソースの場所を検索するには、[参照] をクリックします。
- 5 ステップ 4 では、コンポーネントに関するプロジェクト情報を保存する PVOB を指定するよう要求されます。矢印をクリックすると、選択可能な PVOB のリストが表示されます。作成した PVOB を選択します。

ClearCase LT で複数のコンポーネントを保管できる VOB を作成するには

- 1 VOB 作成ウィザードを起動します。
- 2 ステップ 1 で VOB の名前を入力します。VOB の目的を説明するコメントを入力します。
- 3 ステップ 2 で、[この VOB が複数のコンポーネントを含むことを可能にする] オプション ボタンと [次のコンポーネントを VOB に含める] チェックボックスをオンにします。
[ビュー] リストからビューを選択し、[追加] をクリックします。

[コンポーネントの追加] ダイアログ ボックスにコンポーネントの名前とルート ディレクトリを入力し、[OK] をクリックします。ウィザードのリストにコンポーネントが表示されます。別のコンポーネントを作成するには、[追加] をクリックします。コンポーネントの名前は、その PVOB 内で固有でなければなりません。
- 4 ステップ 3 で、VOB を保存する場所を指定します。ウィザードのこのページには、ClearCase 管理者が作成した VOB の保存場所のリストが表示されます。VOB の保存場所が 1 つだけの場合、このステップは省略され、その VOB 保存場所が使用されます。

複数のコンポーネントを保存する VOB の作成 (UNIX)

ClearCase で複数のコンポーネントを保存できる VOB を作成するには

- 1 `cleartool mkvob` コマンドを使用します。たとえば、次のように指定します。

`cleartool mkvob -nc -tag /vobs/testvob13 /usr/vobstore/testvob13.vbs`

MVFS 環境で開発者が動的ビューを使用する場合には、次の 2 つの手順を実行します。
- 2 VOB タグに対応する VOB マウント ポイントを作成します。たとえば、次のように指定します。

`mkdir /vobs/testvob13`
- 3 VOB をマウントします。たとえば、次のように指定します。

`cleartool mount /vobs/testvob13`

ClearCase LT で複数のコンポーネントを保存できる VOB を作成するには、`cleartool mkvob` コマンドを使用します。たとえば、次のように指定します。

`cleartool mkvob -nc -tag /testvob13 -stgloc vobstore`

コンポーネントを作成して VOB に保存するには

- 1 ClearCase プロジェクト エクスプローラで [コンポーネント] フォルダを右クリックし、ショートカット メニューの [新規] をポイントし、[VOB 内のコンポーネント] をクリックします。
- 2 [VOB 内でのコンポーネントの作成] ダイアログ ボックスの [VOB] リストから、このコンポーネントを保存する VOB を選択します。コンポーネントの名前とルート ディレクトリを入力します。[OK] をクリックします。

VOB あたり 1 つのコンポーネントの作成 (Windows)

ClearCase で VOB とそのコンポーネントを 1 つ作成するには

- 1 VOB 作成ウィザードを起動します。
- 2 ステップ 1 でコンポーネントの名前を入力します。コンポーネントの名前は、その PVOB 内で固有でなければなりません。コンポーネントの目的を説明するコメントを入力します。
[この VOB は UCM コンポーネントを含む] チェックボックスをオンにします。
- 3 ステップ 2 で、[VOB を単一の VOB レベルのコンポーネントとして作成する] チェックボックスをオンにします。ウィザードには、操作を実行するビューが必要です。[ビュー] リストからビューを選択します。
- 4 ステップ 3 で、コンポーネントを保存する場所を指定します。推奨されている場所を 1 つ選択するか、別の場所の UNC パスを入力できます。ネットワーク上の共有リソースの場所を検索するには、[参照] をクリックします。
- 5 ステップ 4 では、コンポーネントに関するプロジェクト情報を保存する PVOB を指定するよう要求されます。矢印をクリックすると、選択可能な PVOB のリストが表示されます。作成した PVOB を選択します。

ClearCase によりコンポーネントが作成されます。このコンポーネントには、コンポーネントのルート ディレクトリの `¥main¥0` バージョンを指すベースラインがあります。

ClearCase LT で VOB とそのコンポーネントを 1 つ作成するには

- 1 [スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase LT] の順にポイントし、[ClearCase]、[VOB の作成] をクリックします。VOB 作成ウィザードが表示されます。
- 2 ステップ 1 でコンポーネントの名前を入力します。コンポーネントの名前は、その PVOB 内で固有でなければなりません。コンポーネントの目的を説明するコメントを入力します。
- 3 ステップ 2 で、[VOB を単一の VOB レベルのコンポーネントとして作成する] チェックボックスをオンにします。ウィザードには、操作を実行するビューが必要です。[ビュー] リストからビューを選択します。
- 4 ステップ 3 で、VOB の記憶ディレクトリを保存できる場所を 1 つ選択します。ウィザードのこのページには、ClearCase 管理者が作成した VOB の保存場所のリストが表示されます。VOB の保存場所が 1 つだけの場合、このステップは省略され、その VOB 保存場所が使用されます。

VOB あたり 1 つのコンポーネントの作成 (UNIX)

ClearCase で VOB とそのコンポーネントを 1 つ作成するには

- 1 **cleartool mkview** コマンドを使用してビューを作成します。動的ビューを作成する場合には、**cleartool setview** コマンドも実行します。たとえば、次のように指定します。

```
cleartool mkview -tag myview /net/host2/view_store/myview.vws  
cleartool setview myview
```

- 2 **cleartool mkvob** コマンドを使用して VOB を作成します。たとえば、次のように指定します。

```
cleartool mkvob -nc -tag /vobs/testvob1 /usr/vobstore/testvob1.vbs
```

MVFS 環境で開発者が動的ビューを使用する場合には、次の 2 つの手順を実行します。

- 3 VOB タグに対応する VOB マウント ポイントを作成します。たとえば、次のように指定します。

```
mkdir /vobs/testvob1
```

- 4 VOB をマウントします。たとえば、次のように指定します。

```
cleartool mount /vobs/testvob1
```

- 5 **cleartool mkcomp** コマンドを実行します。たとえば、次のように指定します。

```
cleartool mkcomp -nc -root /vobs/testvob1 testcomp1@/vobs/myproj2_pvob
```

この例では、**mkcomp** コマンドにより **testvob1** という VOB に基づいてコンポーネント **testcomp1** が作成されます。この例では VOB とコンポーネントにそれぞれ異なる名前が使用されていますが、同じ名前を使用することもできます。コンポーネントの名前は、その PVOB 内で固有でなければなりません。このコマンドを実行する前に VOB と PVOB をマウントする必要があります。**myproj2_pvob** PVOB を使用するすべてのプロジェクトは、**testcomp1** コンポーネントにアクセスできます。

ClearCase LT でコンポーネントを作成するには

- 1 ビューを作成し、この作成したビューに変更します。たとえば、次のように指定します。

```
cleartool mkview -stgloc dev_views ~/chris_snap_view  
cd ~/chris_snap_view
```

- 2 **cleartool mkvob** コマンドを使用して VOB を作成します。たとえば、次のように指定します。

```
cleartool mkvob -nc -tag /testvob1 -stgloc vobstore
```

- 3 **cleartool mkcomp** コマンドを実行します。たとえば、次のように指定します。

```
cleartool mkcomp -nc -root testvob1 testcomp1@/myproj2_pvob
```

cleartool mkcomp コマンドを使用する方法の代わりに、**ClearCase** プロジェクト エクスプローラを使用して既存の **VOB** をコンポーネントに変換する方法があります。詳細については、93 ページの「**VOB からコンポーネントへの変換**」を参照してください。

プロジェクトの作成

この項では、プロジェクト エクスプローラと新規プロジェクト ウィザードを使用してプロジェクトを作成する方法について説明します。コマンド行インターフェイス (CLI) からプロジェクトを作成する方法の詳細については、**cleartool mkproject**、**mkstream**、**mkfolder** の各リファレンス ページを参照してください。プロジェクトを作成するには、次の手順を実行します。

- 1 **Windows** の場合、**ClearCase** エクスプローラの左側のペインで **[UCM]** をクリックし、**[プロジェクト エクスプローラ]** をクリックします。**UNIX** の場合、コマンド行で **clearprojexp** と入力します。
- 2 プロジェクト エクスプローラの左側のペインには、ローカル **ClearCase** ドメインのすべての **PVOB** のルート フォルダのリストが表示されます。各 **PVOB** にはそれぞれ独自のルート フォルダがあります。**ClearCase** は **PVOB** の名前を使用してルート フォルダを作成します。

ClearCase は、**[コンポーネント]** というフォルダも作成します。このフォルダには、**PVOB** の各コンポーネントのエントリが保存されます。フォルダにはプロジェクトとその他のフォルダを保存できます。プロジェクト情報を保存する **PVOB** のルート フォルダを選択します。

- 3 **[ファイル]** の **[新規]** をポイントし、**[フォルダ]** をクリックして、プロジェクト フォルダを作成します。プロジェクト フォルダを作成する必要はありませんが、作成しておくと便利です。プロジェクト数の増加にともない、関連するプロジェクトをまとめる上でプロジェクト フォルダが役立ちます。
- 4 左側のペインで、プロジェクト フォルダまたはルート フォルダを選択します。**[ファイル]** の **[新規]** をポイントし、**[プロジェクト]** をクリックします。新規プロジェクトの ウィザードが表示されます。
- 5 新規プロジェクト ウィザードのステップ 1 で、分かりやすいプロジェクト名を入力し、このプロジェクトの目的を説明するコメントを入力します。プロジェクトのインテグレーション ストリームの名前を入力するか、デフォルト名 (**project name_Integration**) を受け入れます。作成するプロジェクトのタイプを選択します。従来の並行開発プロジェクトでは、ユーザーが複数のストリームを作成できます。これにより、開発者は専用の作業空間と共有の作業空間を使用できます。シングル ストリーム プロジェクトでは、1 つのインテグレーション ストリーム だけが使用されます。シングル ストリーム プロジェクトでは、ユーザーは複数の開発ストリームを作成できません。シングルストリーム プロジェクトと複数ストリーム プロジェクトの詳細については、36 ページの「ストリーム方針の選択」を参照してください。

- 6 ステップ 2 では、既存のプロジェクトに基づいてプロジェクトを作成するかどうかを指定します。プロジェクトを新規に作成するため、[いいえ] をクリックします。
- 7 ステップ 3 では、プロジェクトが使用するベースラインを選択するように要求されます。

[追加] をクリックします。[ベースラインの追加] ダイアログ ボックスが開きます。

[コンポーネント] リストで、作成したコンポーネントを 1 つ選択します。Windows の場合、[変更] をポイントし、[すべてのストリーム] をクリックします。UNIX の場合、[元のストリーム] ボックス下部にある矢印をクリックし、[すべてのストリーム] を選択します。コンポーネントの初期ベースラインが [ベースライン] リストに表示されます。ベースラインを選択します。[OK] をクリックします。ステップ 3 のリストにベースラインが表示されます。すべての基本ベースライン (プロジェクトの複合ベースラインが保存されているコンポーネントのベースラインを含む) がすべてプロジェクトに含まれるまで、[ベースラインの追加] ダイアログ ボックスを使用してベースライン追加操作を行います。
- 8 ステップ 4 では、このプロジェクトに対して実施する開発ポリシーを指定するよう要求されます。実施するポリシーのチェックボックスをオンにします。各ポリシーの詳細については、「第 4 章 ポリシーの設定」を参照してください。
- 9 ステップ 5 では、ClearQuest 統合を使用できるようにプロジェクトを構成するかどうかを選択します。プロジェクトで Rational ClearQuest で使用可能にするには、[はい] をクリックし、リストから ClearQuest ユーザー データベースを選択します。統合の詳細については、98 ページの「プロジェクトで UCM と ClearQuest の統合を使用可能にする」を参照してください。

ベースライン命名テンプレートの設定

UCM では、プロジェクト内のベースライン命名規則を実装するためのテンプレートを定義できます。このテンプレートには、以下のトークンを組み込むことができます。

- プロジェクト
- コンポーネント
- ストリーム
- 日付
- 時刻
- ユーザー
- ホスト
- ベース名

ベース名は、ユーザーが指定する名前を参照します。このテンプレートを設定するには、`-blname_template` オプションを指定した `cleartool mkproject` コマンドまたは `chproject` コマンドを実行します。たとえば、次のように指定します。

```
cleartool chproject -blname_template project,component,date mck_proj1
```

この例では、**mck_proj1** プロジェクトで作成されるすべてのベースラインの名前にプロジェクト名、コンポーネント名、日付を使用するテンプレートが設定されます。コマンド行に入力するトークンを区切るには、カンマを使用します。ベースラインを作成すると、カンマがアンダースコアに置き換えられます。

ベースライン命名テンプレートを指定しないと、**ClearCase** はベース名を使用して新しいベースラインに名前を付けます。必要に応じて、ベースライン名が一意の名前になるように、ベースライン名に数字識別子が追加されます。

デリバリー操作実行時に、ソース ストリームにベースラインが作成されます。このベースラインの名前を付けるときに、ベース名トークンの代わりに **deliverbl.source-stream-name.date.unique-identifier** が使用されます。

chproject と **mkproject** の使用法の詳細については、『**Rational ClearCase リファレンス ガイド**』を参照してください。

プロモーション レベルの定義

ClearCase には 5 つのベースライン プロモーション レベルがあります。プロモーション レベルの一部またはすべてを保持できます。また、独自のプロモーション レベルを定義できます。プロジェクトで使用するプロモーション レベルを定義するには、次の手順を実行します。

- 1 プロジェクト エクスプローラで、プロジェクトの保存先 **PVOB** ルート フォルダを選択し、**[ツール]** をポイントして **[プロモーション レベルの定義]** をクリックします。この **PVOB** を使用するすべてのプロジェクトは、同じプロモーション レベルのセットにアクセスできます。
- 2 **[プロモーション レベルの定義]** ダイアログ ボックスが開きます。既存のプロモーション レベルを削除するには、削除するプロモーション レベルを選択し、**[削除]** をクリックします。プロモーション レベルの順序を変更するには、プロモーション レベルを選択して、**[上へ移動]** ボタンまたは **[下へ移動]** ボタンをクリックします。
- 3 新しいプロモーション レベルを追加するには、**[追加]** をクリックします。**[プロモーション レベルの追加]** ダイアログ ボックスを表示します。新しいプロモーション レベルの名前を入力し、**[OK]** をクリックします。**[プロモーション レベルの定義]** ダイアログ ボックスのプロモーション レベル リストに、新しいプロモーション レベルが表示されます。新しいプロモーション レベルを適切な位置に移動します。
- 4 プロモーション レベルの設定と順序付けが完了したら、新しいベースラインの初期プロモーション レベルにするものを選択します。初期プロモーション レベルは、ベースラインの作成時にデフォルトで割り当てられるレベルです。

CLI からプロモーション レベルを定義する方法については、**cleartool setplevel** のリファレンス ページを参照してください。

インテグレーション ビューの作成

プロジェクトを作成すると、そのプロジェクトのインテグレーション ストリームが ClearCase により作成されます。プロジェクトの共有エレメントを確認、変更する場合にインテグレーション ビューが必要です。インテグレーション ビューを作成するには、次の手順を実行します。

- 1 プロジェクト エクスプローラで、オブジェクト階層内を下方へ移動し、インテグレーション ストリームへ移動します。
 - a ルート フォルダ
 - b プロジェクト フォルダ
 - c プロジェクト
 - d ストリーム

図 26 にこの階層を示します。

図 26 プロジェクト エクスプローラでのインテグレーション ストリームへの移動



- 2 インテグレーション ストリームを選択し、[ファイル] の [新規] をポイントし、[ビュー] をクリックします。
- 3 Windows の場合、ビュー作成ウィザードが表示されます。UNIX の場合、[ビューの作成] ダイアログ ボックスが表示されます。デフォルト値を受け入れ、インテグレーション ストリームに関連付けるインテグレーション ビューを作成します。デフォルトでは、ビュー作成ウィザードと [ビューの作成] ダイアログ ボックスではインテグレーション ビュー名には命名規則 (username_project-name_int) が使用されます。

ClearCase では次の 2 種類のビューを利用できます。

- 動的ビュー。動的ビューでは ClearCase のマルチバージョン ファイル システム (MVFS) を使用して、VOB 内に保存されているファイルとディレクトリに迅速かつ透過的にアクセスできます。Windows の場合、ClearCase は動的ビューを Windows エクスプローラ内のドライブ文字にマッピングします。
- スナップショット ビュー。スナップショット ビューは VOB からユーザーのコンピュータのディレクトリにファイルとディレクトリをコピーします。

製品メモ: Rational ClearCase LT はスナップショットビューのみをサポートします。

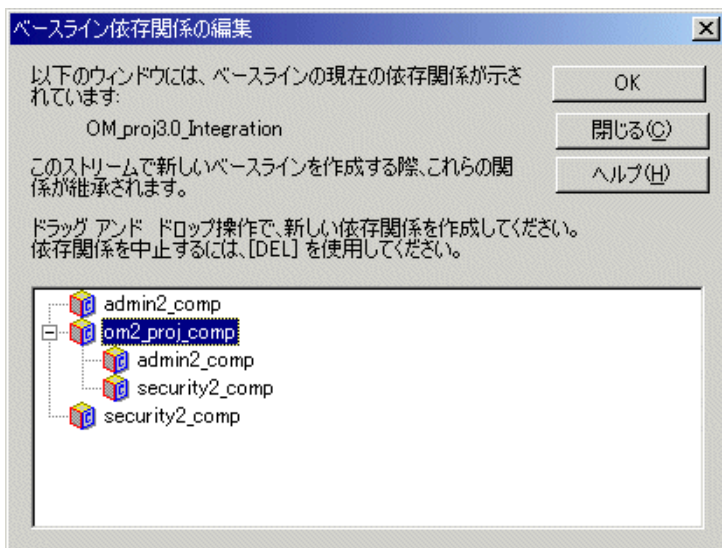
インテグレーション ビューと動的ビューを作成し、開発者がインテグレーション ストリームにデリバリーするファイルとディレクトリの正しいバージョンを常に確認できるようにすることをお勧めします。スナップショット ビューでは、更新操作を実行して、デリバリーされた最新のファイルとディレクトリをご使用のコンピュータへコピーする必要があります。動的ビューとスナップショット ビューの詳細については、『Rational ClearCase ソフトウェア開発ガイド』を参照してください。

プロジェクトを示す複合ベースラインの作成

プロジェクトで使用される各コンポーネントから最新ベースラインを選択して、このプロジェクトを示す複合ベースラインを作成するには、次の手順を実行します。

- 1 プロジェクト エクスプローラ で、プロジェクトのインテグレーション ストリームを右クリックしてショートカット メニューを表示します。[ベースライン依存関係の編集] をクリックします。
- 2 [ベースライン依存関係の編集] ダイアログ ボックスに、このプロジェクトで使用されるすべてのコンポーネントのリストが表示されます。複合ベースラインが保存されるコンポーネントを確認します。複合ベースラインが保存されるコンポーネントにそのほかのコンポーネントをドラッグします。たとえば図 27 では、om2_proj_comp コンポーネントに複合ベースラインが保存されます。複合ベースラインは、admin2_comp コンポーネントと security2_comp コンポーネントからベースラインを選択します。
- 3 [OK] をクリックします。[ベースライン依存関係の作成] ダイアログ ボックスが表示されます。
- 4 [ベース名] フィールド (Windows) または[ベースライン タイトル] フィールド (UNIX) に、UCM がこれらのコンポーネントに対して作成するベースラインの名前を入力します。プロジェクトにベースライン命名テンプレートが設定されている場合には、[テンプレート名] フィールドに使用される名前が表示されます。テンプレートに[ベース名] トークンがない場合、またはテンプレートが設定されていない場合には、[ベースライン依存関係の作成] ダイアログ ボックスには[ベース名] フィールドまたは[ベースライン] フィールドが表示されません。
- 5 [OK] をクリックします。

図 27 [ベースライン依存関係の編集] ダイアログ ボックスの使用法



アクティビティの作成と設定 (UNIX のみ)

インテグレーション ストリームにエレメントを追加する前に、アクティビティを作成して設定する必要があります。

- 1 インテグレーション ビューが動的ビューの場合には、このインテグレーション ビューを設定します。たとえば、次のように指定します。

```
cleartool setview kmt_Integration
```

インテグレーション ビューがスナップショット ビューの場合には、このインテグレーション ビューに変更します。

- 2 `cleartool mkactivity` コマンドを実行します。たとえば、次のように指定します。

```
cleartool mkactivity -headline "Create Directories" create_directories
```

ClearCase GUI ツールは、`-headline` で指定された名前を使用してアクティビティを識別します。最後の引数 `create_directories` は、アクティビティ セクタです。`cleartool` コマンドを実行するときにアクティビティ セクタを使用します。

- 3 `cleartool mkactivity` コマンドを使用してアクティビティを作成すると、デフォルトではビューにそのアクティビティが設定されます。1 つのコマンド行で複数のアクティビティを作成する場合や、`-in` オプションを使用してストリームを指定する場合にはアクティビティはビューに設定されません。インテグレーション ビューにアクティビティを設定する必要がある場合には、`cleartool setactivity` コマンドを使用します。たとえば、次のように指定します。

```
cleartool setactivity create_directories
```

ディレクトリ構造の作成

計画フェーズで定義するディレクトリ構造を実装するため、プロジェクトのコンポーネント内にディレクトリ エlementを作成する必要があります。これは、現在プロジェクトを新規に作成しているためです。34 ページの「ディレクトリ構造の定義」を参照してください。

Windows 環境の場合

ディレクトリ エlementをコンポーネントに追加するには

- 1 Windows エクスプローラで、インテグレーション ビューへ移動します。コンポーネントをダブルクリックして、その内容を表示します。複数コンポーネントを保存するために作成した VOB の中にコンポーネントがある場合には、そのコンポーネントは VOB の下にフォルダとして表示されます。
- 2 新規フォルダを作成します。
- 3 フォルダを右クリックしてショートカット メニューを表示します。[ClearCase] をポイントし、[ソース管理に追加] をクリックします。
- 4 指示に従い、追加する新しいディレクトリ エlementに関連付けるアクティビティを指定します。

UNIX 環境の場合

ディレクトリ エlementをコンポーネントに追加するには

- 1 アクティビティが設定されているインテグレーション ビューで、コンポーネントへ移動します。複数コンポーネントを保存するために作成した VOB の中にコンポーネントがある場合には、そのコンポーネントは VOB の下にディレクトリとして表示されます。たとえば、次のように指定します。

```
cd /vobs/testvob13/libs
```

- 2 コンポーネントのルート ディレクトリをチェックアウトします。たとえば、次のように指定します。

```
cleartool co -nc .
```

- 3 cleartool mkelem コマンドを実行します。たとえば、次のように指定します。

```
cleartool mkelem -nc -eltype directory design
```

この例は、**design** というディレクトリ エlementを作成します。デフォルトでは、**mkelem** コマンドはエlementをチェックアウトした状態のままにします。サブディレクトリなどのエlementをディレクトリ エlementに追加するには、ディレクトリ エlementをチェックアウトした状態のままにしておく必要があります。

- 4 新しいディレクトリへのエレメントの追加が完了したら、このエレメントをチェックインします。次に例を示します。

```
cleartool ci -nc design
```

- 5 ディレクトリ エレメントの作成が完了したら、コンポーネントのルートディレクトリをチェックインします。たとえば、次のように指定します。

```
cleartool ci -nc .
```

ディレクトリ エレメントとファイル エレメントの作成方法の詳細については、『Rational ClearCase ソフトウェア開発ガイド』と `mkelem` のリファレンス ページを参照してください。

ClearCase 外部からのディレクトリとファイルのインポート

ClearCase バージョン管理下に多数のファイルとディレクトリを配置する場合には、コマンド行ユーティリティ `clearexport` と `clearimport` を使用すると、迅速に処理できます。この 2 つのユーティリティでは、別のバージョン管理ソフトウェア システム (SourceSafe、RCS、PVCS など) から既存のディレクトリとファイルのセットを ClearCase に移行できます。

ソース ファイルをコンポーネントに移行するには

- 1 `clearexport` を実行し、ソース ファイルからデータ ファイルを生成します。
- 2 非 UCM ビューを作成、設定します。Windows の場合、ビュー作成ウィザードを使用します。ビュー作成ウィザードを開始するには、ClearCase エクスプローラで [ベース ClearCase] をポイントし、[ビューの作成] をクリックします。UNIX の場合、`cleartool mkview` コマンドと `setview` コマンドを使用します。
- 3 ビュー内から `clearimport` を実行し、データ ファイルのファイルとディレクトリをコンポーネントに移入します。
- 4 コンポーネントで、ラベルが付けられた一連のバージョン からベースラインを作成します。ベースラインに含めるバージョンにラベルが付いていない場合には、ラベル タイプを作成し、バージョンに適用します。詳細については、94 ページの「ラベルからのベースラインの作成」を参照してください。

あるいは、VOB に対して `clearexport` と `clearimport` を実行してから、VOB をコンポーネントに変換します。VOB をコンポーネントへ変換する方法の詳細については、93 ページの「既存の ClearCase 構成に基づいたプロジェクトの作成」を参照してください。

現在バージョン管理下でないディレクトリとフラット ファイルを移行するには、コマンド行ユーティリティ `clearfsimport` を使用します。ディレクトリとファイルを直接ストリームにインポートするには、UCM ビュー内から `clearfsimport` を実行します。次に、バージョンにラベルを付けずにストリームにベースラインを作成できます。詳しくは、`clearfsimport` のリファレンス ページを参照してください。

clearexport と clearimport の使用法の詳細については、『Rational ClearCase 管理ガイド』と、clearexport と clearimport のリファレンス ページを参照してください。

Windows 版 ClearCase には、インポート ウィザードが用意されています。このウィザードは、clearexport コマンドと clearimport コマンドに相当する機能を提供する GUI です。開始ウィザードに続けてインポート ウィザードを開始できます。

ベースラインの作成と推奨

ディレクトリ構造を作成してファイルをインポートしたら、これらのディレクトリ エLEMENT とファイル ELEMENT を選択する新しいベースラインを作成し、推奨します。プロジェクトに参加する開発者は、推奨ベースラインにより示されるバージョンを各自の開発ストリームに移入します。ベースラインの作成方法の詳細については、115 ページの「新規ベースラインの作成」を参照してください。ベースラインの推奨方法の詳細については、119 ページの「ベースラインの推奨」を参照してください。

既存の ClearCase 構成に基づいたプロジェクトの作成

既存の VOB がある場合には、これらの VOB とそのディレクトリをコンポーネントに変換できます。これにより、プロジェクトに VOB とディレクトリを含めることができます。この項では、既存の VOB に基づいてプロジェクトをセットアップする方法について説明します。

PVOB の作成

Windows の場合、78 ページの「プロジェクト VOB の作成 (Windows)」に説明されている手順に従い、VOB 作成ウィザードを使用して PVOB を作成します。管理 VOB を現在使用している場合には、ステップ 3 でリストから管理 VOB を選択します。PVOB と管理 VOB との間に AdminVOB ハイパーリンクが作成されます。作成されたコンポーネントは、既存の管理 VOB を使用します。管理 VOB を現在使用していない場合には、[なし] を選択します。

UNIX の場合、79 ページの「プロジェクト VOB の作成 (UNIX)」で説明されている手順に従い cleartool mkvob コマンドを使用します。管理 VOB を現在使用している場合には、cleartool mkhlink コマンドを使用して PVOB と管理 VOB の間に AdminVOB ハイパーリンクを作成します。作成されたコンポーネントは、既存の管理 VOB を使用します。

VOB からコンポーネントへの変換

VOB をコンポーネントに変換するには

- 1 プロジェクト エクスプローラで、PVOB を選択します。Windows の場合、[ツール] の [インポート] をポイントし、[VOB をコンポーネントに] をクリックします。UNIX の場合、[ツール]、[インポート] をポイントし、[VOB のインポート] をクリックします。[VOB のインポート] ダイアログ ボックスが表示されます。

- 2 [使用可能な VOB] リストで、コンポーネントに変換する VOB を選択します。[追加] をクリックし、VOB を [インポート対象の VOB] リストに移動します。[インポート対象の VOB] リストにはさらに VOB を追加できます。追加した VOB を元に戻す場合は、[インポート対象の VOB] リストで該当する VOB を選択し、[削除] をクリックします。VOB が [使用可能な VOB] リストに戻ります。操作を完了したら、[インポート] をクリックします。

VOB の内容を複数のコンポーネントに分けることができます。VOB のディレクトリ ツリーをコンポーネントに変換するには、次の手順を実行します。

- 1 プロジェクト エクスプローラで PVOB フォルダを右クリックし、[インポート] をポイントして [VOB ディレクトリをコンポーネントに] を選択します。
- 2 [VOB ディレクトリをコンポーネントにインポート] ダイアログ ボックスで、[ビュー] リストからビューを選択し、[VOB] リストからディレクトリが含まれている VOB を選択します。[ルート ディレクトリ] リストからディレクトリを選択し、コンポーネント名を指定します。

新しいコンポーネントには、ディレクトリと、そのすべてのサブディレクトリとファイルが含まれています。コンポーネントのルート ディレクトリは VOB ルート ディレクトリと同レベルまたはその直下に作成する必要があります。コンポーネントのルート ディレクトリが VOB ルート ディレクトリと同レベルの場合には、その VOB には複数のコンポーネントを保存できません。

ラベルからのベースラインの作成

既存の VOB またはそのディレクトリ ツリーの 1 つをコンポーネントに変換した後、このコンポーネントの中のディレクトリとファイルへアクセスするには、ラベル タイプにより識別されるバージョンからベースラインを作成する必要があります。ベースラインを作成するには、次の手順を実行します。

- 1 Windows の場合、使用するバージョンのセットにラベルが付いていない場合には、ラベル適用ウィザードを使用してラベル タイプを作成、適用します。ラベル適用ウィザードを開始するには、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[ラベル適用ウィザード] をクリックします。あるいは、コマンド プロンプトに「clearapplywizard」と入力します。

UNIX の場合、使用するバージョンのセットにラベルが付いていない場合には、cleartool mklbtype コマンドと mklabel コマンドを使用してラベル タイプを作成、適用します。たとえば、次のように指定します。

```
% cleartool mklbtype -c "label for release 2" REL2
Created label type "REL2".
```

```
% cleartool mklable -recurse REL2 .
Created label "REL2" on "." version "/main/5".
Created label "REL2" on "./src" version "/main/6".
Created label "REL2" on "./src/Makefile" version "/main/2".
```

-recurse オプションは、ClearCase に対し、現在の作業ディレクトリと同レベルまたはそれ以下のすべてのバージョンにラベルを適用するよう指示します。

- 2 プロジェクト エクスプローラで PVOB を選択します。Windows の場合、[ツール]、[インポート] をポイントし、[ラベルをベースラインに] をクリックします。UNIX の場合、[ツール]、[インポート] をポイントし、[ラベルのインポート] をクリックします。ラベルのインポート ウィザードのステップ 1 が表示されます。
- 3 [使用可能なコンポーネント] リストで、ベースラインを作成するラベルのあるコンポーネントを選択します。[追加] をクリックし、[選択されたコンポーネント] リストにコンポーネントを移動します。追加したコンポーネントを元に戻す場合は、[選択されたコンポーネント] リストで該当するコンポーネントを選択し、[削除] をクリックします。コンポーネントが [使用可能なコンポーネント] リストに戻ります。
- 4 手順 2 で、インポートするラベル タイプを選択し、このラベル タイプにより識別されるバージョンに対し作成するベースラインの名前を入力します。次に、ベースラインのプロモーション レベルを選択します。

メモ: グローバル ラベル タイプ定義からラベル タイプをインポートすることはできません。

プロジェクトの作成

85 ページの「プロジェクトの作成」で説明されている手順に従い、新規プロジェクト ウィザードを使用してプロジェクトを作成します。

インテグレーション ビューの作成

88 ページの「インテグレーション ビューの作成」で説明されている手順に従い、インテグレーション ビューを作成します。

既存のプロジェクトに基づいたプロジェクトの作成

新しいプロジェクトを作成するときに、場合によっては既存のプロジェクトの新しいバージョンを作成する必要があります。たとえば、Webotrans プロジェクトのバージョン 3.0 をリリースしており、バージョン 3.1 の計画を立てているとします。バージョン 3.1 ではバージョン 3.0 コンポーネントと同じコンポーネントを使用する予定であるため、バージョン 3.0 コンポーネントの最新ベースラインを基本ベースラインとして使用してバージョン 3.1 を開発します。

複合ベースラインを使用した最終ベースラインの取得

既存のプロジェクトに多数のコンポーネントが含まれている場合には、新しいプロジェクトを作成する前に、これらのコンポーネントの最終ベースラインを選択する複合ベースラインを作成できます。この複合ベースラインは、既存プロジェクトの最終承認済みベースラインから作業を開始するチームが使用できる共通の複合ベースラインです。このような複合ベースラインを作成するには、次の手順を実行します。

- 1 VOB にルート ディレクトリのないコンポーネントを作成します。80 ページの「プロジェクト ベースラインを保存するコンポーネントの作成」を参照してください。
- 2 コンポーネントの初期ベースラインをインテグレーション ストリームに追加します。109 ページの「コンポーネントの追加」を参照してください。
- 3 コンポーネントで、プロジェクトのそのほかのすべてのコンポーネントのベースラインを選択する複合ベースラインを作成します。89 ページの「プロジェクトを示す複合ベースラインの作成」を参照してください。
- 4 複合ベースラインを推奨します。119 ページの「ベースラインの推奨」を参照してください。

既存の PVOB とコンポーネントの再使用

プロジェクトではPVOB を新規に作成しないでください。これは、プロジェクトが既存のプロジェクトの新しいバージョンであり、既存のプロジェクトと同じコンポーネントを使用するためです。引き続き既存の PVOB を使用します。

プロジェクトの作成

85 ページの「プロジェクトの作成」で説明されている手順に従い、新規プロジェクト ウィザードを起動してプロジェクトを作成します。

このウィザードのステップ 2 で [はい] を選択して、既存のプロジェクトのベースラインからプロジェクトを開始することを指定します。次に、これらのベースラインが含まれているプロジェクトに移動します。図 28 に示す新しいプロジェクトは、OM_proj1.0_Integration ストリームのベースラインに基づいています。

図 28 新規プロジェクト ウィザードのステップ 2



ステップ 3 には、ステップ 2 で選択したプロジェクトの最新ベースラインが表示されます。既存のプロジェクトの最終承認済みベースラインを取り込んだ複合ベースラインを作成した場合には、その複合ベースラインを選択します。既存のプロジェクトに含まれていないコンポーネントのベースラインを追加するには、[追加] をクリックして [ベースラインの追加] ダイアログ ボックスを表示します。同様に、ベースラインを削除するには、ベースラインを選択して [削除] をクリックします。

85 ページの「プロジェクトの作成」で説明されている手順に従い、ウィザードの残りのステップを完了します。

インテグレーション ビューの作成

新しいプロジェクトを作成すると、新しいインテグレーション ストリームが ClearCase により作成されます。したがって、このインテグレーション ストリームのエレメントへアクセスするために新しいインテグレーション ビューを作成する必要があります。88 ページの「インテグレーション ビューの作成」で説明されている手順に従い、インテグレーション ビューを作成します。

プロジェクトで UCM と ClearQuest の統合を使用可能にする

プロジェクトを ClearQuest ユーザー データベースに接続する前に、UCM で使用可能なスキーマを使用できるようにこのデータベースをセットアップする必要があります。「第 5 章 ClearQuest ユーザー データベースの設定」を参照してください。

プロジェクトが ClearQuest ユーザー データベースを使用できるようにするには

- 1 プロジェクト エクスプローラの左側のペインでプロジェクトを右クリックして、ショートカット メニューを表示します。[プロパティ] をクリックして、そのプロパティ シートを表示します。
- 2 [ClearQuest] タブをクリックし、[プロジェクトで ClearQuest を使用可能にする] チェックボックスをオンにします。プロジェクトにリンクするユーザー データベースを選択します。プロジェクトを初めて使用可能にすると、そのプロジェクトの [ログイン] ダイアログ ボックスが表示されます。ユーザー名、パスワード、プロジェクトのリンク先データベース名を入力します。
- 3 実施する開発ポリシーを選択します。開発ポリシーの詳細については 61 ページの「UCM と ClearQuest の統合」を参照してください。操作が完了したら、[OK] をクリックします。

新しいプロジェクトを作成する場合は、図 29 に示すように、新規プロジェクト ウィザードのステップ 5 で [はい。次の ClearQuest データベースを使用します] をオンにし、ユーザー データベースを選択します。

UCM プロジェクトで ClearQuest ユーザー データベースを使用可能に設定したら、このプロジェクトを別のユーザー データベースにリンクすることもできます。アクティビティが作成されていない場合には、プロジェクトのプロパティ シートの [ClearQuest] タブで別のデータベースを選択すると、そのデータベースに切り替わります。

アクティビティの移行

ClearQuest データベースを使用可能にしたプロジェクトにアクティビティが含まれていると、UCMUtilityActivity レコード タイプを使用して各アクティビティのレコードが作成されます。プロジェクトのすべてのアクティビティを別のレコード タイプのレコードに保存するには、プロジェクトを作成した時点、つまりチーム メンバーがアクティビティを作成する前に、プロジェクトを使用可能にします。移行完了後に作成されるアクティビティは、UCM で使用可能なレコード タイプのレコードにリンクできます。

図 29 プロジェクトで ClearQuest ユーザー データベースを使用可能にする



プロジェクト ポリシーの設定

UCM で使用可能なスキーマには、ClearCase または ClearQuest で設定できる 4 つのポリシーが含まれます。

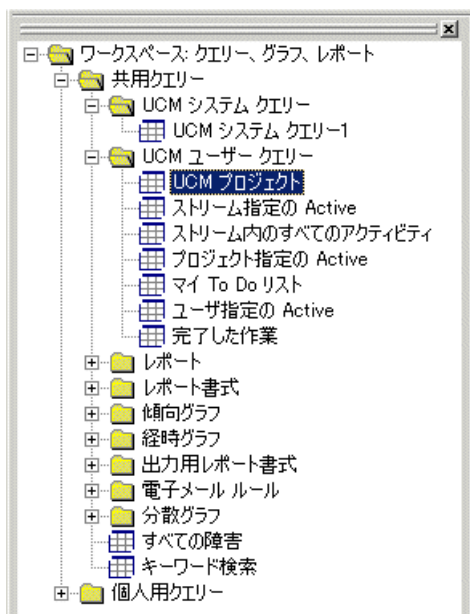
ClearCase では、図 29 に示すようにプロジェクトのプロパティ シートの [ClearQuest] タブのチェックボックスをオンにすることでポリシーを設定します。

ClearQuest からアクティビティを設定するには

- 1 Windows の場合、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearQuest] の順にポイントし、[Rational ClearQuest] をクリックして、ClearQuest クライアントを開始します。UNIX の場合、シェル プロンプトに `clearquest` と入力して、ClearQuest クライアントを開始します。ClearQuest クライアントワークスペースで、図 30 に示すように [UCM プロジェクト] クエリーに移動します。
- 2 クエリーをダブルクリックし、UCM で使用可能なプロジェクトをすべて表示します。
- 3 [結果] セットからプロジェクトを選択します。プロジェクトのフォームが表示されます。
- 4 フォームで [アクション] をクリックし、[Modify] を選択します。設定するポリシーのチェックボックスをオンにします。

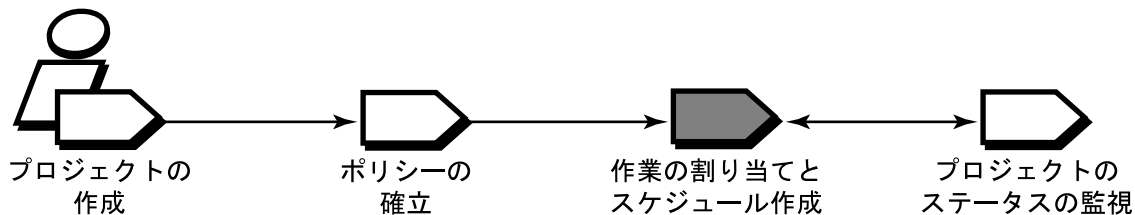
これらのポリシーの詳細については、61 ページの「UCM と ClearQuest の統合」を参照してください。

図 30 UCMProjects クエリーへの移動



アクティビティの割り当て

プロジェクト
マネージャー



ClearQuest でアクティビティの作成と割り当てを行うには

- 1 ClearQuest クライアントを開始し、プロジェクトに接続しているユーザー データベースにログオンします。
- 2 [アクション] の [新規作成] をクリックします。[レコードタイプの選択] ダイアログボックスが表示されます。UCM で使用可能なレコードタイプを選択して、[OK] をクリックします。

- 3 登録フォームが表示されます。各タブのボックスに情報を入力します。ボックスへの入力
が完了したら、[OK] をクリックします。ClearQuest によりレコードが作成され、このレ
コードの状態が Submitted 状態になります。
- 4 クエリーを実行し、このレコードを選択します。たとえばレコードのタイプが Defect の
場合、[すべての障害] クエリーを実行できます。
- 5 [アクション] の [Assign] をクリックし、[所有者] リストから所有者を選択します。
[適用] をクリックします。

ClearQuest で、アクティビティを割り当てる開発者のユーザー アカウント プロファイルが作成
されている必要があります。ユーザー アカウント プロファイルの作成方法の詳細については、
75 ページの「ユーザーの作成」を参照してください。

プロジェクトと ClearQuest ユーザー データベース間のリンクの無効化

場合によっては、プロジェクトと ClearQuest ユーザー データベースの間のリンクを無効にする
必要があります。リンクを無効にするには、次の手順を実行します。

- 1 プロジェクトのプロパティ シートの [ClearQuest] タブで、[プロジェクトで ClearQuest を
使用可能にする] チェックボックスをオンにします。
- 2 [ClearQuest] タブで [OK] をクリックします。プロジェクトと ClearQuest データベースの
間のリンクが無効になります。また、アクティビティとそれぞれのアクティビティに対応
する ClearQuest レコードの間のリンクも削除されます。
- 3 プロジェクトを別のユーザー データベースにリンクする場合には、プロジェクトのプロパ
ティ シートを再度表示し、[プロジェクトで ClearQuest を使用可能にする] チェックボッ
クスをオンにし、別のユーザー データベースを選択します。

メモ: リンクを解除したユーザー データベースを選択すると、プロジェクトのアクティビ
ティに対して新しい ClearQuest レコードが作成されます。この場合、アクティビティは以前
リンクされていた ClearQuest レコードにリンクされません。

リンクされているアクティビティとリンク解除されているアクティビティが混在 するプロジェクトの修正

プロジェクトで ClearQuest を使用可能にした後でも、プロジェクトの一部のアクティビティが
ClearQuest レコードにリンクされない状態で残ることがあります。同様に、プロジェクトと
ClearQuest の間のリンクを無効にしても、一部のアクティビティがリンクされた状態で残るこ
とがあります。プロジェクトがこのような整合性のない状態になる可能性のあるシナリオを次に示
します。

- 有効化操作または無効化操作の実行中にネットワーク障害またはシステム クラッシュが発生し、アクティビティ移行処理が中断された。
- **ClearQuest** ユーザー データベースが壊れたため、ユーザー データベースのバックアップバージョンを強制的に復元した。ユーザー データベースのバックアップバージョンは、このユーザー データベースにリンクされているプロジェクトが含まれている **PVOB** と同期していません。
- **cleartool** コマンド行インターフェイスを使用して、アクティビティまたはプロジェクトの名前を変更した。コマンド行インターフェイスからアクティビティまたはプロジェクトの名前を変更しても、対応する **ClearQuest** レコードは更新されません。この結果、**ClearCase** オブジェクトと **ClearQuest** オブジェクトの間に不整合が発生します。
- プロジェクトの **PVOB** が **ClearCase MultiSite** 構成に含まれており、**MultiSite** 同期操作によって、**ClearQuest** を使用可能なローカル **PVOB** プロジェクトに、リンクされていないアクティビティが追加された。

問題の検出

ClearQuest が使用可能に設定されているプロジェクトのリンクされていないアクティビティの修正などの操作を開発者が行おうとすると、エラーが表示され、その操作が禁止されます。

問題の修正

上記の 1 番目から 3 番目のシナリオのいずれかが原因で問題が発生している場合には、**-ucm** オプションを指定した **cleartool checkvob** コマンドを使用して、プロジェクトを整合性のある状態に復元します。このコマンドの使用法の詳細については、『**Rational ClearCase 管理ガイド**』と『**Rational ClearCase リファレンス ガイド**』を参照してください。

問題の原因が **MultiSite** にある場合は、リモート サイトで次の操作を実行します。

- 1 プロジェクト エクスプローラでプロジェクトのプロパティ シートを表示し、**[ClearQuest]** タブをクリックします。
- 2 [このレプリカでマスター登録されている、リンクされていないアクティビティをリンクします] をクリックします。プロジェクトのアクティビティがすべてチェックされ、リンクされていないアクティビティがすべてリンクされます。次の概要情報が表示されます。
 - リンクする必要があったアクティビティの数。
 - すでにリンクされていたアクティビティの数。
 - 現在の **PVOB** レプリカでマスター登録されていないためにリンクできなかったアクティビティの数。この場合、問題を修正するために [このレプリカでマスター登録されている、リンクされていないアクティビティをリンクします] 操作を実行する必要があるレプリカのリストも表示されます。
- 3 ステップ 2 のリストの各レプリカについて、ステップ 1 とステップ 2 を繰り返します。

UCM と ClearQuest の統合に対する MultiSite の影響

ClearCase MultiSite を使用して PVOB と ClearQuest MultiSite を複製し、UCM と ClearQuest の統合で使用される ClearQuest ユーザー データベースとスキーマ リポジトリを複製するには、各種要件を理解する必要があります。この項では、これらの要件について説明します。

レプリカと命名に関する要件

UCM と ClearQuest の統合をセットアップするときに、プロジェクトと ClearQuest ユーザー データベースの間にリンクを作成します。MultiSite を使用する場合には、次の要件が適用されます。

- リンクされているプロジェクトが保存されている PVOB レプリカがあるサイトごとに、プロジェクトのリンク先の ClearQuest ユーザー データベースのレプリカと、ユーザー データベースのスキーマ リポジトリを作成する必要がある。同様に、リンクされている ClearQuest ユーザー データベース レプリカがあるサイトごとに、そのユーザー データベースのリンク先であるプロジェクトが保存されている PVOB のレプリカが必要です。
- ClearQuest レプリカの名前は、同じサイトの PVOB レプリカ名に一致している必要がある。

プロジェクトのマスターシップの転送

プロジェクトで ClearQuest を使用可能にする前に、現在の PVOB レプリカでプロジェクトをマスター登録する必要があります。レプリカでプロジェクトがマスター登録されていない場合には、プロジェクトをマスター登録するレプリカで `multitool chmaster` コマンドを使用して、プロジェクトのマスターシップを転送します。

プロジェクトで ClearQuest を使用可能にすると、ClearQuest ユーザー データベースに対応するプロジェクト レコードが作成され、そのレコードのマスターシップが ClearQuest ユーザー データベースの現在のレプリカに割り当てられます。プロジェクトで ClearQuest を使用可能にしたときにプロジェクトと同名のプロジェクト レコードが ClearQuest ユーザー データベースに存在しており、そのプロジェクト レコードが現在のレプリカにマスター登録されていない場合には、プロジェクト レコードのマスターシップを現在のレプリカに転送します。

ClearQuest レコードへのアクティビティのリンク

アクティビティが含まれているプロジェクトで ClearQuest を使用可能にすると、アクティビティに対応する ClearQuest レコードが作成され、レコードとアクティビティがリンクされます。リモート レプリカにマスター登録されているアクティビティはリンクできません。リモート レプリカにマスター登録されているアクティビティのリンク方法の詳細については、102 ページの「問題の修正」を参照してください。

プロジェクト ポリシー設定の変更

ClearQuest からプロジェクトのポリシー設定を変更する前に、ClearQuest プロジェクト レコードをマスター登録する必要があります。同様に、ClearCase からプロジェクトのポリシー設定を変更する前に、プロジェクト オブジェクトをマスター登録する必要があります。現在のレプリカでプロジェクトのポリシー設定を変更しても、現在のレプリカをシブリング レプリカとを同期するまでは、シブリング レプリカのストリームには新しい設定は反映されません。レプリカの同期方法の詳細については、『Rational ClearCase MultiSite 管理ガイド』を参照してください。

プロジェクト名の変更

統合により、対応する ClearQuest プロジェクト レコードの名前フィールドにプロジェクト名がリンクされます。ClearCase GUI でプロジェクト名を変更すると、対応する ClearQuest プロジェクト レコードの名前フィールドにこの変更が反映されます。同様に、ClearQuest で名前を変更すると、ClearCase のプロジェクト名にこの変更が反映されます。MultiSite 環境でプロジェクト名を変更する前に、プロジェクト レコードとプロジェクト オブジェクトの両方をマスター登録する必要があります。

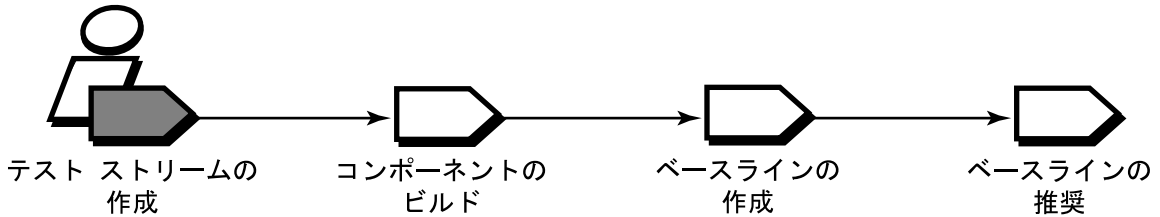
メモ: プロジェクト名を変更するには、必ず CleraCase プロジェクト エクスプローラなどの GUI を使用してください。コマンド行インターフェイスを使用してプロジェクト名を変更すると、対応するプロジェクト レコードにこの変更が反映されません。

Rational Suite を使用した作業 (Windows)

Rational Suite で UCM を使用している場合には、Rational RequisitePro プロジェクト、Rational Rose と XDE モデル、Rational Test データストアを UCM コンポーネントに保存し、これらをベースラインに組み込むことができます。この統合を使用可能にするには、Rational Administrator の GUI を使用して Rational プロジェクトを作成し、構成します。Rational プロジェクトにより、UCM プロジェクトに RequisitePro プロジェクト、Rose モデル、Rational Test データストアが関連付けられます。この統合のセットアップ方法の詳細については、『Rational Suite 統一変更管理 (UCM) ユーザーズ ガイド』を参照してください。

ベースライン テスト用開発ストリームの作成

統合担当者



新しいベースラインを作成するときには、静的なファイルセットを作成してテストできるように、インテグレーション ストリームをロックすることをお勧めします。インテグレーション ストリームをロックしないと、ファイルの作成、テスト中に、ほかの開発者が変更をデリバーして混乱が発生する可能性があります。インテグレーション ストリームの短時間ロックは妥当ですが、インテグレーション ストリームを数日間ロックすると、完了したがデリバーされていないアクティビティが蓄積されます。開発者を長期間ロックアウトすることを回避するため、開発ストリームを作成し、ベースラインの集中的なテストにこのストリームを使用できます。プロジェクトで機能固有の開発ストリームを使用している場合には、機能固有の開発ストリームごとにテスト ストリームを作成し、これらのストリームで作成されたベースラインをテストできます。

開発ストリームを作成するには

- 1 ClearCase プロジェクト エクスプローラでインテグレーション ストリームを右クリックし、ショートカット メニューの [子ストリームの作成] をクリックします。
[開発ストリームの作成] ダイアログ ボックスが表示されます。
- 2 テスト ストリームでの変更を許可しない場合には、[ストリームを読み取り専用にする] チェックボックスをオンにします。このオプションをオンにすると、このストリームのベースラインで検出された障害を修正できなくなります。代わりに、障害の責任者である開発者が各自の開発ストリームで修正し、修正内容をインテグレーション ストリームにデリバーします。
- 3 デフォルトでは、ClearCase による開発ストリームの作成時に、一連の推奨ベースラインが使用されます。新規ベースラインは広範にテストされていないので、推奨ベースラインに関連するレベルにプロモートされていない可能性があります。推奨ベースライン以外のベースラインを使用して開発ストリームを作成するには、[詳細設定オプション] をクリックします。

[ベースラインの変更] ダイアログ ボックスが表示されます。

- 4 [ベースラインの変更] ダイアログ ボックスで、テストするベースラインが含まれるコンポーネントを選択します。[変更] をクリックします。

2 番目の [ベースラインの変更] ダイアログ ボックスが表示されます。このダイアログ ボックスには、コンポーネントのすべてのベースラインが表示されます。
- 5 テストするベースラインを選択して、[OK] をクリックします。別のコンポーネントのベースラインをテストする必要がある場合には、1 番目の [ベースラインの変更] ダイアログ ボックスでそのコンポーネントを選択し、この手順を繰り返します。操作が完了したら、1 番目の [ベースラインの変更] ダイアログ ボックスで [OK] をクリックします。
- 6 [開発ストリームの作成] ダイアログ ボックスで、[このストリームに対するビュー作成を要求する] チェックボックスがオンであることを確認します。[OK] をクリックします。

ビュー作成ウィザード (Windows) または [ビューの作成] ダイアログ ボックス (UNIX) が表示されます。
- 7 開発ストリームのビューを作成するため、ビュー作成ウィザードのステップまたは [ビューの作成] ダイアログ ボックスのフィールドに情報を入力します。

新しいベースラインを作成、テストできるように開発ストリームが構成されました。また、ほかの開発者のベースラインの作成、テストの妨げになっている可能性について心配することなく、インテグレーション ストリームに変更をデリバーできます。ベースラインのテストの詳細については、117 ページの「ベースラインのテスト」を参照してください。

機能別開発ストリームの作成

基本 UCM プロセスでは、インテグレーション ストリームがプロジェクトの唯一の共有作業空間として使用されます。プロジェクトを複数の小さな開発者チームに分割して、各チームがそれぞれ固有の機能を開発することができます。このような編成での作業を支援するには、各開発者チームの共有作業空間として使用する開発ストリームを作成します。各機能を担当する開発者は、機能固有の開発ストリームの推奨ベースラインに基づいて各自の開発ストリームを作成できます。機能固有の開発ストリームの詳細については、36 ページの「ストリーム方針の選択」を参照してください。

機能固有の開発ストリームを作成するには

- 1 ClearCase プロジェクト エクスプローラでインテグレーション ストリームを右クリックし、ショートカット メニューから [子ストリームの作成] を選択します。

[開発ストリームの作成] ダイアログ ボックスが表示されます。
- 2 デフォルトでは、ClearCase による開発ストリームの作成時に、一連の推奨ベースラインが使用されます。推奨ベースライン以外のベースラインを使用して開発ストリームを作成するには、[詳細設定オプション] をクリックし、[ベースラインの変更] ダイアログ ボックスからベースラインを選択します。

- 3 [開発ストリームの作成] ダイアログ ボックスに、新しいストリームの名前と説明を入力します。[このストリームに対するビュー作成を要求する] チェックボックスがオンであることを確認します。[OK] をクリックします。

Windows の場合、ビュー作成ウィザードが表示されます。

UNIX の場合、[ビューの作成] ダイアログ ボックスが表示されます。

- 4 開発ストリームのビューを作成するため、ビュー作成ウィザードのステップまたは [ビューの作成] ダイアログ ボックスのフィールドに情報を入力します。
- 5 ClearCase プロジェクト エクスプローラで機能固有の開発ストリームを右クリックし、[推奨ベースライン] を選択します。
- 6 [推奨ベースライン] ダイアログで [追加] を選択します。[ベースラインの追加] ダイアログ ボックスが表示されます。この機能を担当する開発者に対して推奨するベースラインを選択します。開発者が各自の開発ストリームを作成する場合、これらの開発ストリームは推奨ベースラインに基づいています。ベースラインの選択が完了したら、[推奨ベースライン] ダイアログで [OK] をクリックします。

プロジェクトを作成し、設定が完了したら、開発者がプロジェクトに参加し、アクティビティを実行します。完了したアクティビティは、インテグレーション ストリームまたは機能固有の開発ストリームにデリバーします。統合担当者は、開発者が互いに作業を同期できるようにプロジェクトをメンテナンスする必要があります。本章では次のメンテナンス タスクについて説明します。

- コンポーネントの追加
- コンポーネントのビルド
- 新規ベースラインの作成
- ベースラインのテスト
- ベースラインの推奨
- プロジェクト状態の監視
- プロジェクトのクリーンアップ

コンポーネントの追加

通常、プロジェクトの規模は次第に拡張し、コンポーネントの追加が必要になる場合があります。ストリームにコンポーネントを追加するには、次の操作を実行します。

- 1 ClearCase エクスプローラの左側のペインで、[UCM] をクリックして、[プロジェクト エクスプローラ] をクリックします。

UNIX: `clearprojexp` と入力して、ClearCase プロジェクト エクスプローラを起動します。

- 2 プロジェクト エクスプローラの右側のペインでストリームを右クリックして、ショートカットメニューを表示します。[プロパティ] をクリックして、ストリームの [プロパティ] ダイアログ ボックスを開きます。

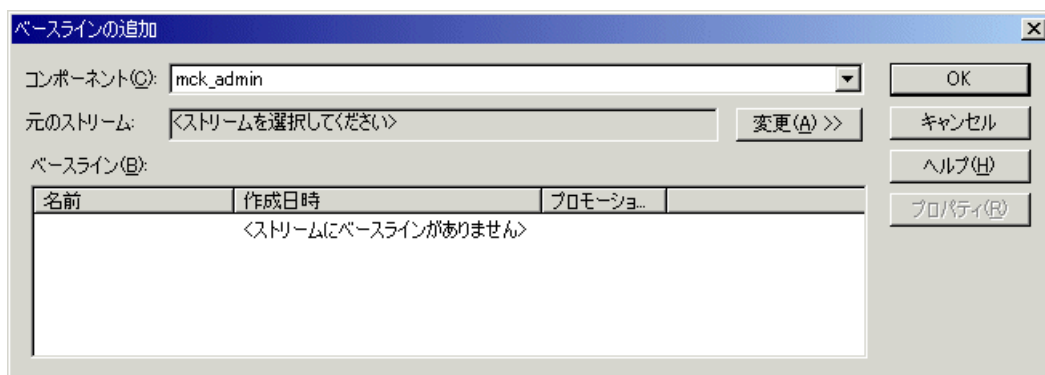
- 3 [構成] タブをクリックして、[追加] をクリックします。図 31 に示す [ベースラインの追加] ダイアログ ボックスが表示されます。

- 4 Windows: [変更]、[すべてのストリーム] をクリックするか、[変更]、[参照] をクリックして、追加するコンポーネント ベースラインがあるストリームを選択します。

UNIX: [元のストリーム] ボックスの端にある矢印をクリックし、ツリーからストリームを選択するか、[すべてのストリーム] をクリックします。

- 5 [コンポーネント] リストで、追加するコンポーネントを選択します。コンポーネントのベースラインが [ベースライン] リストに表示されます。

図 31 [ベースラインの追加] ダイアログ ボックス



- 6 [ベースライン] リスト内から、プロジェクトに追加するベースラインを選択します。
- 7 [OK] をクリックします。[ベースラインの追加] ダイアログ ボックスが閉じ、選択したベースラインが [構成] タブに表示されます。
- 8 [OK] をクリックして、ストリームの [プロパティ] ダイアログ ボックスを閉じます。
[リベースのプレビュー] ダイアログ ボックスが表示されます。新規の基本ベースラインを含むようにインテグレーション ストリームの構成を変更するため、UCM ではインテグレーション ストリームをリベースする必要があります。
- 9 [リベースのプレビュー] ダイアログ ボックスで [OK] をクリックします。
- 10 [完了] をクリックして、リベース操作を完了します。

コンポーネントを変更可能にする

デフォルトでは、プロジェクトに追加したコンポーネントは読み取り専用になります。コンポーネントをプロジェクト内で変更可能にするには、次の手順を実行します。

- 1 プロジェクト エクスプローラでプロジェクトを選択し、[ファイル]、[ポリシー] をクリックします。
- 2 [コンポーネント] タブでコンポーネントの隣のチェック ボックスをオンにします。
- 3 [OK] をクリックします。

ビューの同期

追加したコンポーネントにアクセスするためには、次の手順に従ってビューをストリームの新しい構成と同期する必要があります。

- 1 プロジェクト エクスプローラで、追加したコンポーネントがあるストリームを選択し、[ファイル]、[プロパティ] をクリックします。
- 2 [ビュー] タブをクリックします。ビューを選択し、[プロパティ] をクリックします。
- 3 [一般] タブを開き、[ストリームと同期] をクリックします。

子ストリームの同期

子ストリームがコンポーネントにアクセスできるようにするには、子ストリームをプロジェクトの新しい変更可能なコンポーネントと同期し、さらに子ストリームのビューもストリームの新しい構成と同期する必要があります。

子ストリームをプロジェクトの新しい変更可能なコンポーネントと同期するには、次の手順を実行します。

- 1 プロジェクト エクスプローラでストリームを選択し、[ファイル]、[プロパティ] をクリックします。
- 2 [一般] タブを開き、[プロジェクトと同期] をクリックします。

子ストリームのビューをストリームの新しい構成と同期するには、次の手順を実行します。

- 1 プロジェクト エクスプローラでストリームを選択し、[ファイル]、[プロパティ] をクリックします。
- 2 [ビュー] タブをクリックします。ビューを選択し、[プロパティ] をクリックします。
- 3 [一般] タブを開き、[ストリームと同期] をクリックします。

スナップショット ビューのロード規則の更新

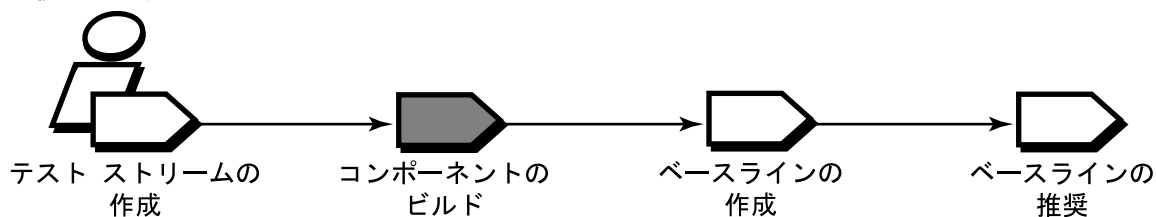
ビューがスナップショット ビューである場合、ストリームに追加したコンポーネントが表示の対象となるように、ビューのロード規則を編集する必要があります。スナップショット ビューのロード規則は、ClearCase がどのコンポーネントをビューにロードするかを指定するものです。ビューのロード規則を編集するには、次の手順を実行します。

- 1 プロジェクト エクスプローラ で、ストリームを選択して、[ファイル]、[プロパティ] をクリックし、ストリームのプロパティ シートを表示します。
- 2 プロパティ シートの [ロード規則] タブをクリックします。
- 3 ストリームに追加したコンポーネント (複数可) を選択します。
- 4 [追加] をクリックします。[OK] をクリックして、プロパティ シートを閉じます。

また、プロジェクトで作業している開発者で、開発ビューとしてスナップショットビューを使用している開発者がいるか知る必要があります。スナップショットビューを使用している開発者が新規コンポーネントのあるベースラインをリベースする場合、ClearCase はスナップショットビューの構成仕様を更新しますが、ビューのロード規則は更新しません。このため、コンポーネントを追加した場合には、スナップショットビューを使用している開発者に対し、開発ストリームを新規ベースラインにリベースした後で開発ビューのロード規則を更新する必要があることを通知します。

コンポーネントのビルド

統合担当者



ストリームに新しいベースラインを作成するときには、まず現在のベースラインとその作成以降に開発者がそのストリームにデリバーした作業結果を使用して、コンポーネントをビルドします。コンポーネントのビルドが完了したら、デリバーされた最新の作業結果を選択するベースラインを作成できます。コンポーネントをビルドするには、以下のタスクを実行します。

- ストリームのロック
- リモートのデリバー操作の検出
- リモートのデリバー操作の実行
- 不正なデリバー操作の取り消し
- コンポーネントのビルドとテスト

インテグレーション ストリームのロック

インテグレーション ストリームまたは機能固有のストリームにコンポーネントをビルドする前に、そのストリームをロックし、開発者が作業結果をデリバーできないようにします。これにより、一連のファイルを実際に静的な状態で扱うことができます。

- 1 プロジェクト エクスプローラで、ストリームを選択します。
- 2 [ファイル]、[プロパティ] をクリックして、ストリームのプロパティ シートを表示します。
- 3 [ロック] タブをクリックします。
- 4 [ロック済み] をクリックして、[OK] をクリックします。

メモ: ストリームをロックするときに、開発者のデリバー操作が進行中の場合もあります。その場合、アクティビティに関連するファイルがチェックインされず、それがもとでビルド操作が不完全なものとなり、不正なベースラインが作成されてしまうことがあります。進行中のデリバー操作がないかどうかをチェックするスクリプトを作成し、ストリームをロックする前に実行することをお勧めします。

デリバーできる作業結果の検索

コンポーネントをビルドする前に、デリバー操作の完了が必要になる場合があります。ほとんどの場合は、開発者がデリバー操作を完了します。しかし、ターゲット ストリームのマスターが開発者のソース ストリームではなく別のレプリカである **MultiSite** 構成では、開発者がデリバー操作を完了することはできません。このようなストリームのマスタースhip状況が検出されると、デリバー操作がリモート デリバー操作になります。

リモート デリバー操作では、**ClearCase** はデリバー操作を開始しますが、それをポスト済み状態のままにします。ポスト済み状態のデリバー操作を見つけ、完了するのは統合担当者の作業です。ポスト済み状態のデリバー操作を行った開発者は、統合担当者がそのデリバー操作を完了するかキャンセルするまで、ソース開発ストリームからのデリバーもソース開発ストリームのリベースもできません。

製品メモ: Rational ClearCase LT は ClearCase MultiSite をサポートしていません。

ポスト済み状態のすべてのデリバー作業を検索するには

- 1 プロジェクト エクスプローラで、プロジェクトを選択します。
- 2 [ツール]、[ポスト済みデリバーの検索] をクリックします。プロジェクト内にポスト済みデリバーがある場合は、[ポスト済みデリバーの検索] ダイアログ ボックスが開き、そこにポスト済み状態のデリバー操作を含むすべてのストリームが表示されます。それぞれのポスト済みデリバー操作にソース ストリームとターゲット ストリームが表示されます。

特定のターゲット ストリームのポスト済みデリバー操作を検索するには、ストリームを選択し、[ツール]、[ポスト済みデリバーの検索] をクリックします。[ポスト済みデリバーの検索] ダイアログ ボックスに、選択したターゲット ストリームを対象とするポスト済みデリバー操作のソース ストリームが表示されます。[ポスト済みデリバーの検索] ダイアログ ボックスには、選択したターゲット ストリームの直接の子であるソース ストリームからのポスト済みデリバー操作だけが表示されます。

リモート デリバー操作の完了

開発ストリームのリモート デリバー操作を完了するには

- 1 [ポスト済みデリバーの検索] ダイアログ ボックス内の一覧から、開発ストリームを選択します。
- 2 [デリバー] をクリックします。[デリバー] ダイアログ ボックスが表示されます。デリバー操作を再開するには [再開] をクリックします。デリバー操作をキャンセルするには [キャンセル] をクリックします。デリバー操作を完了するための詳細については、『Rational ClearCase ソフトウェア開発ガイド』を参照してください。

デリバー操作の取り消し

開発者は、デリバー操作を完了する前ならいつでも、作業をとりやめ、変更内容を元に戻すことができます。ただし、インテグレーション ビューにバージョンをチェックインする場合、開発者が変更を元に戻すことは簡単ではありません。このような場合、プロジェクト マネージャーが `cleartool rmver -xhlink` コマンドを使用して、チェックインされたバージョンを削除する必要があります。

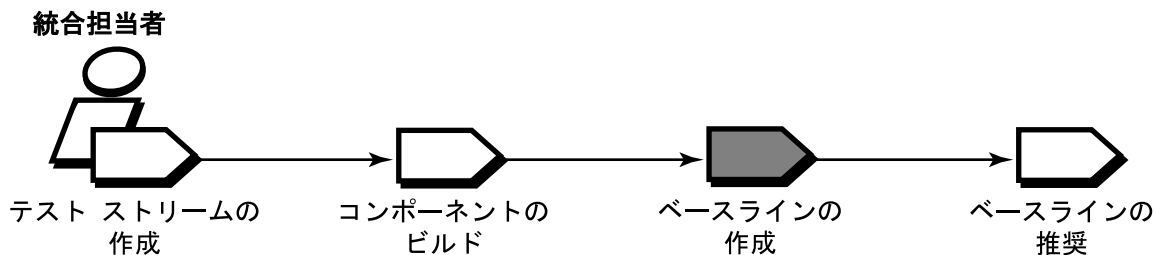
メモ: `rmver` コマンドは開発チームの開発履歴の一部を消去するため、予期しない結果を招くことがあります。そのため、このコマンドを使用する場合、特に `-xhlink` オプションと共に使用する場合は、注意が必要です。詳細については、『Rational ClearCase リファレンス ガイド』の `rmver` リファレンス ページを参照してください。

バージョンを削除しても、必ず変更が無効になるわけではありません。後続バージョンが作成されていたり、バージョンを削除する前にバージョンがマージされていた場合は、変更は残ります。この場合、ファイルをチェックアウトし、編集して変更を削除し、ファイルを再度チェックインする必要があります。

コンポーネントのビルドとテスト

ストリームをロックし、未処理のデリバー操作を完了したら、プロジェクトの実行可能ファイルをビルドし、それをテストすることによって、前回ベースラインを作成してから開発者がデリバーした変更の中にバグがないかどうかを確認できます。ビルドの実行に関する詳細については、『Rational ClearCase ソフトウェア ビルドガイド』を参照してください。ストリーム内でビルドとテストを実行するときにはストリームをロックするため、新規ベースラインの詳細なテストを行う場合は別の開発ストリームを使用することをお勧めします。現在のストリームでは、長時間にわたってロックせずに済むように、簡単な検証テストのみを行います。新規ベースラインをテストする開発ストリームの使用については、117 ページの「ベースラインのテスト」を参照してください。

新規ベースラインの作成



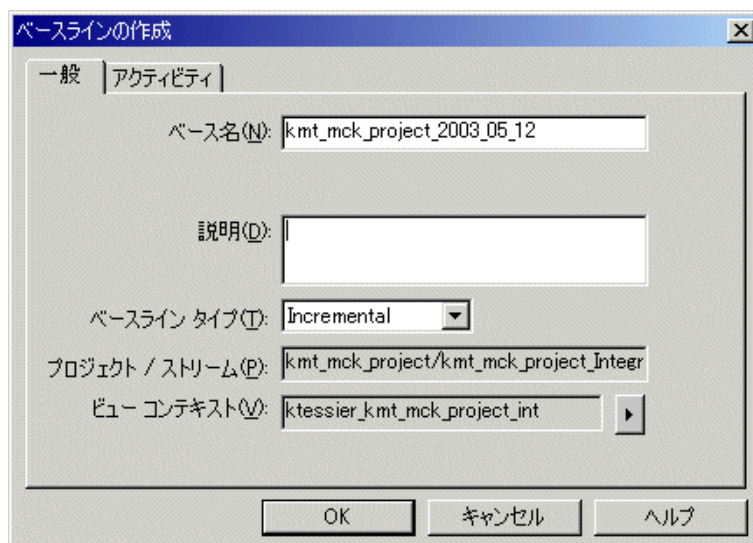
開発者はインテグレーション ストリームまたは機能固有の開発ストリームに作業結果をデリバリーするので、変更を記録するために新規ベースラインを頻繁に作成することが重要です。それによって、開発者は新規にリベースし、ほかの開発者の変更を反映した最新の状態で作業することができます。ベースラインを作成するときには、ストリームがロックされ、開発者が作業結果をデリバリーできないことを確認します。

新規ベースラインの作成

ストリーム内のすべてのコンポーネントの新規ベースラインを作成するには

- 1 プロジェクト エクスプローラで、ベースラインを作成するインテグレーション ストリームまたは機能固有の開発ストリームを選択します。
- 2 [ツール]、[ベースラインの作成] をクリックします。[ベースラインの作成] ダイアログ ボックスが開きます (図 32)。

図 32 [ベースラインの作成] ダイアログ ボックス



- 3 プロジェクトにベースライン命名テンプレートが設定されている場合は、新しいベースラインに使用される名前が [テンプレート名] フィールドに表示されます。図 32 には、プロジェクト名、コンポーネント名、日付で構成されるベースライン命名テンプレートが表示されています。[ベース名] フィールド (Windows) や [ベースライン タイトル] フィールド (UNIX) に名前を入力するのは、プロジェクトにベースライン命名テンプレートが設定されていない場合や、テンプレートにベース名トークンが含まれている場合だけです。それ以外の場合は、[ベースラインの作成] ダイアログ ボックスには [ベース名] フィールドまたは [ベースライン タイトル] フィールドは表示されません。
- 4 作成するベースラインの種類を選択します。

インクリメンタル ベースラインは、最新のフル ベースラインと、その最新フル ベースラインが作成された以降に変更されたバージョンを記録して作成されるベースラインです。

フル ベースラインは、コンポーネントのルート ディレクトリの下位にあるすべてのバージョンを記録するベースラインです。

一般的に、インクリメンタル ベースラインはフル ベースラインと比較して短時間で作成できますが、ClearCase による内容の検索は、フル ベースラインの方がインクリメンタル ベースラインよりも短時間で行うことができます。
- 5 操作を実行するビューを指定します。ベースラインを作成するストリームにアタッチされたビューを選択します。

複数のアクティビティを取り込んだベースラインの作成

デフォルトでは、最新のベースラインが作成されてから変更されたすべてのアクティビティが新規ベースラインに取り込まれます。しかし、特定のアクティビティのみを取り込んだベースラインを作成したい場合もあります。その場合は、[ベースラインの作成] ダイアログ ボックスの [アクティビティ] タブをクリックし、ベースラインに取り込むアクティビティを選択します。

1 つのコンポーネントのベースラインの作成

ストリーム内のすべてのコンポーネントではなく、ある特定のコンポーネントのベースラインを作成するには、次の手順を実行します。

- 1 プロジェクト エクスプローラで、新しいベースラインを作成するストリームを選択します。
[ファイル]、[プロパティ] をクリックして、ストリームのプロパティ シートを表示します。
- 2 [ベースライン] タブをクリックします。コンポーネントを選択し、[ベースラインの作成] をクリックします。
- 3 [ベースラインの作成] ダイアログ ボックスのフィールドに情報を入力し、[OK] をクリックします。

ストリームのアンロック

新規ベースラインを作成した後は、インテグレーション ストリームまたは機能固有の開発ストリームをアンロックして、開発者がストリームに作業結果をデリバーできるようにします。ストリームをアンロックするには、次の手順を実行します。

- 1 プロジェクト エクスプローラで、ストリームを選択します。
- 2 [ファイル]、[プロパティ] をクリックして、ストリームのプロパティ シートを表示します。
- 3 [ロック] タブをクリックします。
- 4 [アンロック済み] をクリックして、[OK] をクリックします。

ベースラインのテスト

インテグレーション ストリームまたは機能固有の開発ストリームを長時間ロックしないためには、新規ベースラインに対する詳細なテスト (システム テスト、リグレーション テスト、受け入れテストなど) 別の開発ストリームで実行することをお勧めします。開発ストリームの作成に関する詳細については、105 ページの「ベースライン テスト用開発ストリームの作成」を参照してください。

新規ベースラインを作成し、インテグレーション ストリーム内でビルドと初期検証テストの合格を確認した後は、次の手順で開発ストリームをリベースします。

- 1 プロジェクト エクスプローラ で開発ストリームを選択して、[ツール]、[ストリームのリベース] をクリックします。

[リベースのプレビュー] ダイアログ ボックスが表示されます。

- 2 デフォルトでは、**ClearCase** は開発ストリームを推奨ベースラインにリベースします。新規ベースラインは広範にテストされていないので、推奨ベースラインに関連するレベルにプロモートされていない可能性があります。テストするベースラインをリベースするには、[詳細設定] をクリックします。

[リベース構成の変更] ダイアログ ボックスが表示されます。

- 3 テストするベースラインのあるコンポーネントを選択します。[詳細設定] をクリックします。

[リベース構成の変更] ダイアログ ボックスが開き、そこに選択したコンポーネントのすべてのベースラインが表示されます。

- 4 テストするベースラインを選択して、[OK] をクリックします。

- 5 [リベース構成の変更] ダイアログ ボックス内で別のコンポーネントを選択して、このプロセスを繰り返します。ベースラインの選択が完了したら、[OK] をクリックして [リベース構成の変更] ダイアログ ボックスを閉じます。

- 6 [リベースのプレビュー] ダイアログ ボックス内で [OK] をクリックして、リベース操作を続けます。開発ストリームのリベースに関する詳細については、オンライン ヘルプまたは『**Rational ClearCase** ソフトウェア開発ガイド』を参照してください。開発ストリームのリベースが完了したら、次に新規ベースラインでのテストを開始します。

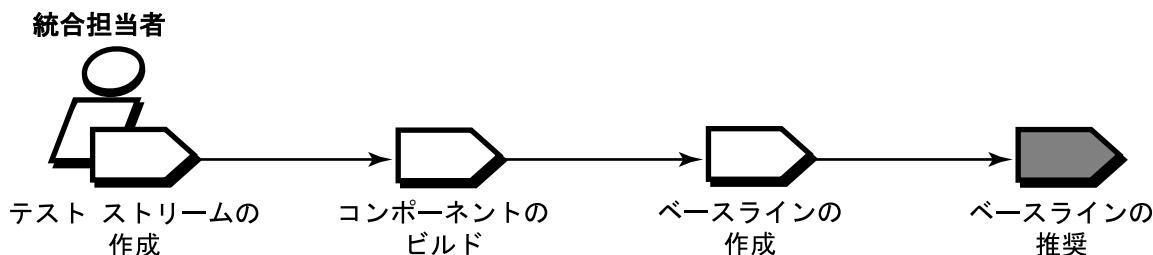
問題の解決

テスト中にベースラインに問題を発見した場合は、影響のあるファイルの問題を解決して、変更内容を次のようにインテグレーション ストリームにデリバーします。

- 1 開発ストリームにアタッチされている開発ビューから、問題のあるファイルをチェックアウトします。ファイルをチェックアウトしたら、アクティビティを指定する必要があります。
- 2 ファイルに必要な変更を加え、チェックインします。
- 3 開発ビュー内で変更をビルドし、テストします。
- 4 変更が反映されたことが確認できたら、開発ストリーム内の変更を取り込んだ新しいベースラインを作成します。

- 5 新しいベースラインをインテグレーション ストリームまたは機能固有の開発ストリームにデリバーします。新しいベースラインをインテグレーション ストリームまたは機能固有の開発ストリームにデリバーするときには、前回のベースラインの作成以降に開発者がデリバーした作業結果に変更内容をマージします。ベースラインのデリバーに関する詳細については、「第9章 複数プロジェクトの並行リリースの管理」を参照してください。
- 6 インテグレーション ストリームまたは機能固有の開発ストリームの推奨ベースラインにテスト ストリームで作成した新しいベースラインを追加します。ベースラインを別のストリームで推奨する方法の詳細については、119 ページの「ベースラインの推奨」を参照してください。

ベースラインの推奨



プロジェクトの作業が進み、コンポーネントの品質と安定性が向上したら、ベースラインが合格したテストのレベルに合わせてベースラインのプロモーション レベル属性を変更し、詳細なテストに合格したベースラインを推奨します。

ベースラインのプロモーション レベルを変更するには

- 1 プロジェクト エクスプローラで、ストリームを右クリックし、ショートカット メニューを表示します。[プロパティ] をクリックして、ストリームの [プロパティ] ダイアログ ボックスを開きます。

UNIX: プロジェクト エクスプローラで、ストリームを選択します。[ファイル]、[プロパティ] をクリックして、ストリームの [プロパティ] ダイアログ ボックスを開きます。
- 2 [ベースライン] タブをクリックします。
- 3 [コンポーネント] リスト内から、プロモートするベースラインのあるコンポーネントを選択します。[ベースライン] リスト内から、ベースラインを選択します。[プロパティ] をクリックします。ベースラインの [プロパティ] ダイアログ ボックスが開きます。
- 4 [プロモーション レベル] リスト内の矢印をクリックして、使用可能なすべてのプロモーション レベルを表示します。新規プロモーション レベルを選択します。

ベースラインを推奨するには

- 1 プロジェクト エクスプローラで、ストリームを選択します。[ツール]、[推奨ベースライン] をクリックします。
- 2 [推奨ベースライン] ダイアログ ボックスでは、プロモーション レベルを選択し、[リストを作成] をクリックすることによって、ベースラインのリストをフィルタできます。これによって、選択したプロモーション レベル以上のベースラインだけが表示されます。
- 3 リストからベースラインを削除するには、削除するベースラインを選択し、[削除] をクリックします。ベースラインを追加するには、[追加] をクリックし、[ベースラインの追加] ダイアログ ボックスでベースラインを選択します。
- 4 コンポーネントの別のベースラインを推奨するには、ベースラインを選択し、[変更] をクリックします。[ベースラインの変更] ダイアログ ボックスで、推奨するベースラインを選択します。テスト ストリームなど、別のストリームのベースラインを選択するには、[変更] をクリックし、[ストリームの選択] ダイアログ ボックス (Windows) または [ベースラインの変更] ダイアログ ボックス (UNIX) で、そのストリームまで移動します。

以下の条件が当てはまる場合は、現在のストリームまたはその基本にはないベースラインを推奨できます。

- ベースラインがストリームと同じプロジェクト内にある。
 - ベースラインがストリームにデリバリーされているか、ストリームがベースラインまたはその子孫にリベースしている。
 - ベースラインに現在の推奨ベースラインが含まれている。つまり、ベースラインが現在の推奨ベースラインの子孫である。
- 5 推奨ベースラインのリストが完成したら、[推奨ベースライン] ダイアログ ボックスの [OK] をクリックします。

ベースラインの競合の解決

複合ベースラインを使用しているプロジェクトでは、同じコンポーネントの2つのベースライン間で発生するストリームの構成の競合を解決しなければならないことがあります。競合が発生する可能性があるのは、次のようなベースラインに関連する操作の実行中です。

- ベースラインの作成
- ストリームの構成へのベースラインの追加
- ベースラインの推奨
- ストリームのリベース

複合ベースラインと非複合ベースラインの競合

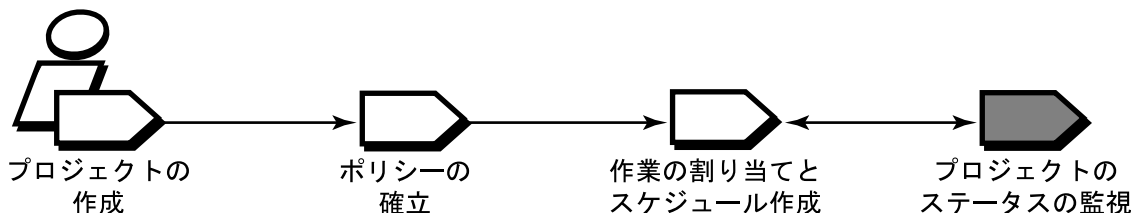
複合ベースライン間の競合よりも複合ベースラインと非複合ベースラインの競合の方が単純です。たとえば、あるストリームの構成にコンポーネント A のベースライン BL4 を選択する複合ベースラインがあり、その複合ベースラインが推奨ベースラインであるとしてします。コンポーネント A の新しいベースライン BL5 をテストし、それを推奨することになりました。これによって、複合ベースラインで選択されたメンバベースライン BL4 が無効になります。[推奨ベースライン] ダイアログ ボックスには、BL5 が「上書き」、BL4 が「上書き済み」と示されます。UCM で使用される上書きの識別子と上書き済みの識別子は、ほかの GUI のものと同じです。

複合ベースライン間の競合

複合ベースラインどうしの競合の方が複雑です。この競合が発生する可能性があるのは、ストリームの構成に複数の複合ベースラインがあり、それぞれが同じコンポーネントのベースラインを選択する場合です。1 つのストリームが同じコンポーネントの 2 つのベースラインを選択することはできません。このような状況を引き起こす可能性のある操作を実行すると、UCM が競合を認識し、操作の完了前に競合を解決するよう要求します。

プロジェクト状態の監視

プロジェクト
マネージャー



ClearCase では、プロジェクトの進捗を追跡するための便利なツールが提供されます。ここでは、これらのツールの使用方法について説明します。

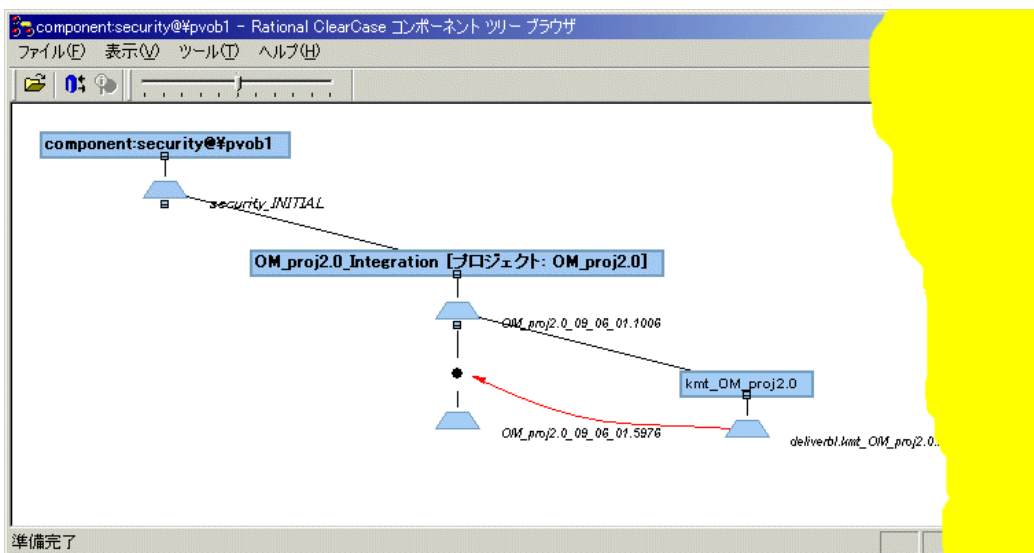
ベースライン履歴の表示 (Windows のみ)

ClearCase の コンポーネント ツリー ブラウザは、コンポーネントのベースライン履歴を表示する GUI です。コンポーネント ツリー ブラウザを開始するには、次の手順を実行します。

- 1 プロジェクト エクスプローラを起動して、参照するベースライン履歴のあるコンポーネントに移動します。
- 2 コンポーネントを右クリックして、ショートカット メニューを表示します。[ベースラインの参照] をクリックします。

図 33 に示す コンポーネント ツリー ブラウザが表示されます。

図 33 ClearCase コンポーネント ツリー ブラウザ



コンポーネント ツリー ブラウザはコンポーネントの開発ラインと、そのコンポーネントを使用している各ストリームを表示します。図 33 の **security_INITIAL** は、プロジェクト マネージャが **security** コンポーネントを作成したときに作成された初期ベースラインです。

OM_proj2.0_09_06_01.1006 は、統合担当者がコンポーネントの作成後、最初に作成したベースラインです。これは、インテグレーション ストリームと開発ストリーム **kmit_OM_proj2.0** の基本ベースラインです。

プレフィックス **deliverbl** が付いたベースラインは、デリバー操作の実行中に **ClearCase** が開発ストリーム内に作成するベースラインです。開発ストリームからインテグレーション ストリームへのインテグレーション 矢印は、デリバー操作を表します。ベースライン

OM_proj2.0_09_06_01.5976 には、このデリバー操作によってデリバーされた作業結果が含まれています。

2 つのベースラインを比較するには、アイコンをクリックしてベースラインを選択します。次に [ツール]、[比較]、[別のベースラインと比較] をクリックします。比較対象のベースラインのアイコンをクリックします。[ベースラインの比較] GUI が開きます。または、[ツール]、[比較]、[以前のベースラインと比較] をクリックして、ベースラインを直接の祖先と比較することもできます。

ベースラインの比較

ClearCase を使用すると、2 つのベースラインの相違をグラフィカルに表示することができます。2 つのベースラインを比較するには、`cleardiffbl` コマンドを使用します。たとえば、次のように指定します。

```
% cleardiffbl OM_proj2.0_Integration_08_12_01 OM_proj2.0_09_06_01
```

このコマンドによって、図 34 に示す [ベースラインの比較] ダイアログ ボックスが開きます。[ベースラインの比較] ダイアログ ボックスは、次の手順でベースラインのプロパティ シート内から開くこともできます。

- 1 ClearCase のプロジェクト エクスプローラで、インテグレーション ストリームを選択して、[ファイル]、[プロパティ] をクリックし、インテグレーション ストリームのプロパティ シートを表示します。
- 2 [ベースライン] タブをクリックして、次に比較するベースラインのあるコンポーネントを選択します。
- 3 ベースラインを選択して右クリックし、[前のベースラインと比較] または [別のベースラインと比較] をクリックします。

図 34 ベースラインの比較

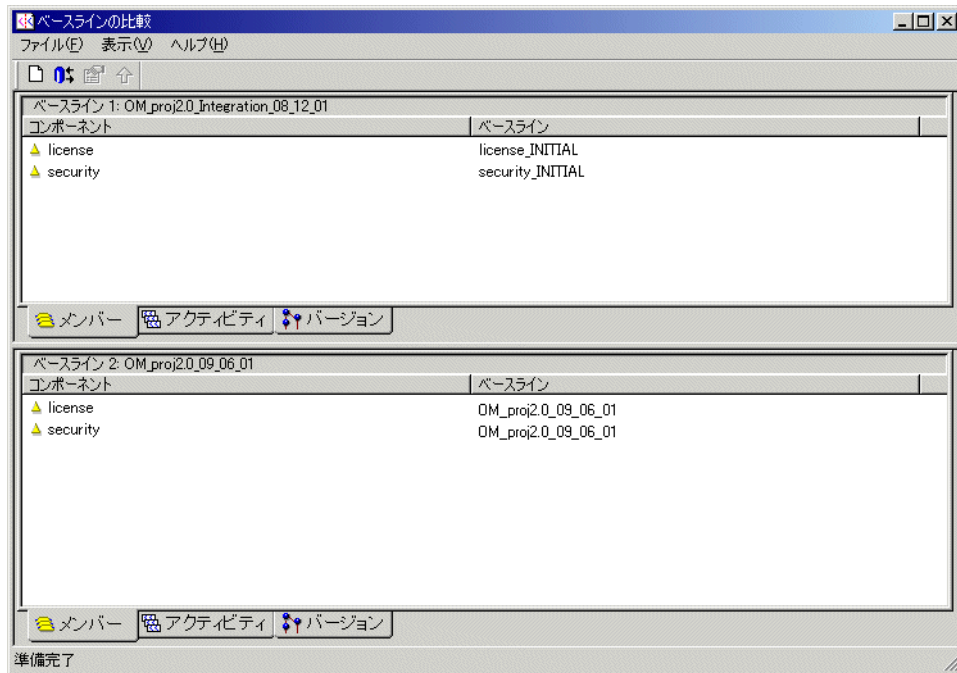
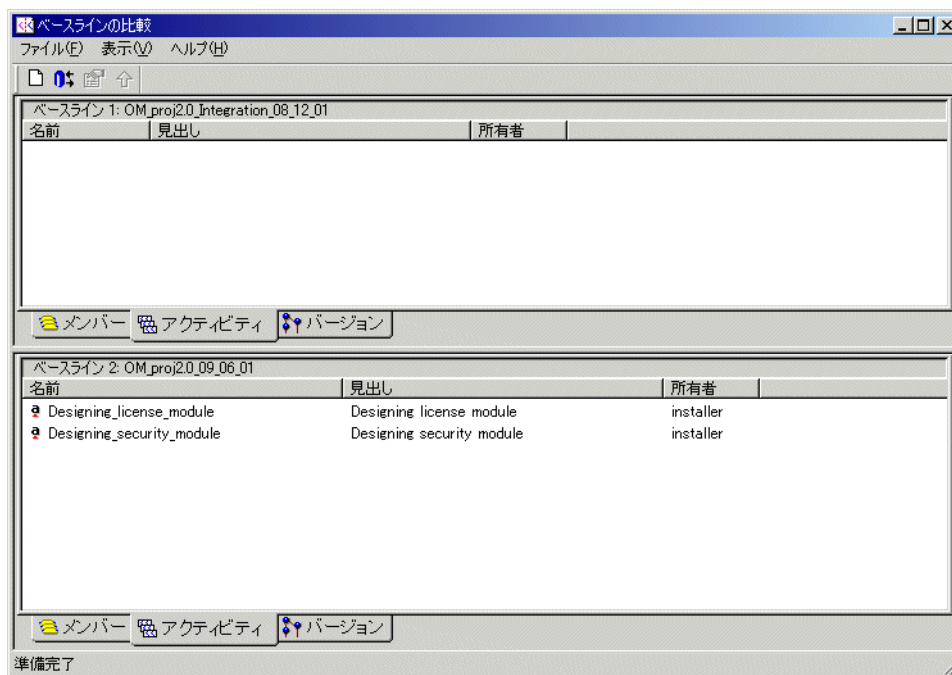


図 34 の [ベースラインの比較] ウィンドウには、2 つの複合ベースライン OM_proj2.0_Integration_08_12_01 と OM_proj2.0_09_06_01 の比較結果が表示されています。[メンバー] タブには、これらの複合ベースラインを構成するベースラインが表示されます。

各ベースラインに含まれるアクティビティを表示するには、[アクティビティ] をクリックします。図 35 に示すとおり、最初のベースラインにはアクティビティがありません。これは、このベースラインのメンバ ベースラインがコンポーネント **license** と **security** の初期ベースラインだからです。

アクティビティに関連する変更セットを表示するには、[バージョン] をクリックします。

図 35 アクティビティごとのベースラインの比較



ClearQuest ユーザー データベースのクエリー

UCM と ClearQuest を統合して使用している場合、ClearQuest のクエリーを使用して、プロジェクトの状態を取得することができます。ClearQuest のユーザー データベースを作成または更新して UCM に適用可能なスキーマを使用する場合、統合によって、ユーザー データベースのワークスペース内にある [共用クエリー] フォルダの 2 つのサブフォルダに 6 つのクエリーがインストールされます。これらのクエリーを使用すると、開発者は自分に割り当てられているアクティビティを容易に参照し、プロジェクト マネージャーはどのプロジェクトのどのアクティビティがアクティブかを簡単に参照できます。表 4 に、それら 6 つのクエリーを挙げ、機能を説明します。

表 4 UCM に適用可能なスキーマ内でのクエリー

クエリー	説明
ActiveForProject	1 つ以上の指定されたプロジェクトに対して、 active 状態タイプのすべてのアクティビティを選択します。
ActiveForStream	1 つ以上の指定されたストリームに対して、 active 状態タイプのすべてのアクティビティを選択します。
ActiveForUser	1 人以上の開発者に対して、 active 状態タイプの割り当てられたすべてのアクティビティを選択します。
AllActivitiesInStream	1 つ以上の指定されたストリームに対して、すべてのアクティビティを選択します。
MyCompletedWork	このクエリーを実行している開発者のアクティビティのうち、 completed 状態にあるすべてのアクティビティを選択します。
MyToDoList	クエリーを実行している開発者に割り当てられている、 active または ready 状態タイプのすべてのアクティビティを選択します。
UCMProjects	ClearQuest ユーザー データベースにリンクされているすべてのプロジェクトを選択します。
UCMCustomQuery1	<p>このクエリーは、ユーザーではなく、UCM と ClearQuest の統合によって使用されます。開発者がファイルをチェックアウトまたはチェックインしたり、ファイルをソース管理に追加する際にアクティビティを選択するときに、このクエリーが呼び出され、開発者のビューに関連付けられているストリーム内で使用可能なアクティビティのリストを表示します。</p> <p>このクエリーは開発者ごとにカスタマイズできます。このクエリーを [共用クエリー] フォルダから開発者の [個人用クエリー] フォルダにコピーし、クエリー エディタを使用してカスタマイズします。</p>

また、ClearQuest クライアント内で、[クエリー]、[新規クエリー]をクリックして、独自のクエリーを作成することもできます。UCM が使用可能なすべてのレコードタイプを検索するクエリーの場合には、表示される [レコードタイプ] ダイアログ ボックス内で、All_UCM_Activities を選択します。

ClearCase レポートの使用 (Windows のみ)

ClearCase レポート アプリケーション (Report Builder と Report Viewer) を使用すると、プロジェクト環境固有のレポートの作成と参照が可能です。レポートのパラメータを選択して定義するには、Report Builder を使用します。レポートの出力結果を参照するには、Report Viewer を使用します。

製品メモ: ClearCase Report Builder を起動するには

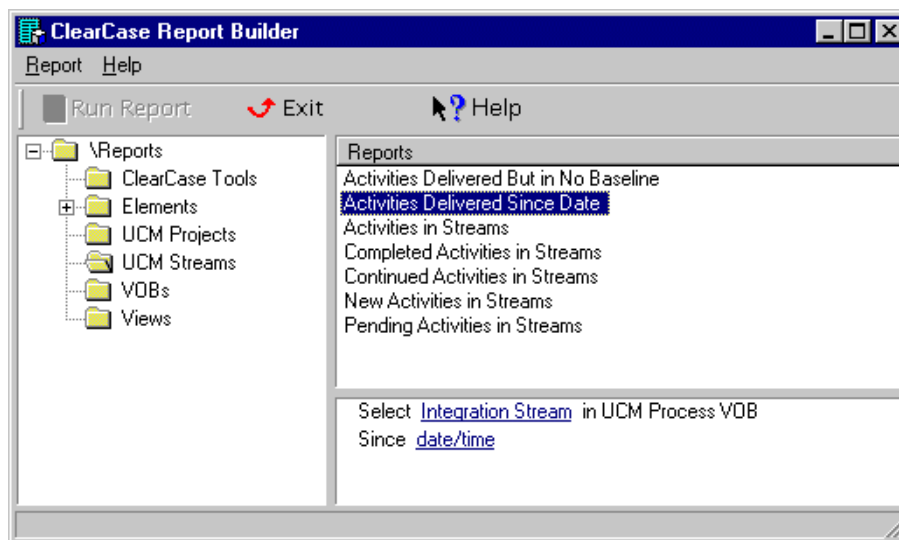
- ClearCase で、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[管理]、[Report Builder] をクリックします。
- ClearCase LT で、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase LT] の順にポイントし、[Report Builder] をクリックします。

ClearCase Report Builder は、UCM プロジェクトやストリームなどのオブジェクト型を基にレポート进行分类します。左側のペインでカテゴリを選択すると、そのカテゴリで利用できるレポートが右上のペインに表示されます。レポートを選択すると、入力すべきパラメータが右下のペインに表示されます。たとえば、図 36 では、Activities Delivered Since Date レポートが選択されており、Report Builder はインテグレーション ストリームの名前と日付の入力を要求されています。

Report Builder と Report Viewer の使用方法の詳細については、それぞれのオンライン ヘルプを参照してください。

ClearCase レポートには Report Builder と Report Viewer アプリケーションへの一連のフックがあります。これらのフックはレポート プロシージャと呼ばれ、特定のレポートを作成して参照するために必要なすべての操作を実装しています。ClearCase レポートのプログラミング インターフェイスを使用すると、レポート プロシージャをカスタマイズすることができます。カスタマイズ方法の詳細については、「付録 C ClearCase レポートのカスタマイズ」を参照してください。

図 36 ClearCase Report Builder



プロジェクトのクリーンアップ

開発チームがプロジェクトの作業を完了し、新しいソフトウェアをリリースまたは配置した場合、プロジェクトの次のバージョンを作成する前に現在のプロジェクト環境をクリーンアップする必要があります。クリーンアップでは、未使用のオブジェクトの削除や、プロジェクトとストリームのロックや非表示を行います。このプロセスを行うことで、不要になった情報を整理して、プロジェクト エクスプローラ内を参照しやすくなります。

未使用のオブジェクトの削除

プロジェクトの進行中、プロジェクト マネージャーや開発者がオブジェクトを作成し、それを使用しない場合があります。別の命名規則を使用することにして、既存のオブジェクトの名前を変更して使う代わりに新規に作成する場合などです。混乱を避け、無用な情報を整理するために、これらの未使用のオブジェクトは削除します。

プロジェクト、ストリーム、コンポーネント、アクティビティを削除するには、プロジェクト エクスプローラでオブジェクトを選択して、[ファイル]、[削除] をクリックします。ベースラインを削除するには、`cleartool rmbl` コマンドを使用します。

プロジェクト

プロジェクトは、プロジェクト内にストリームがない場合のみ削除することができます。プロジェクト作成ウィザードを使用してプロジェクトを作成した場合、ウィザードはインテグレーションストリームも作成します。そのため、プロジェクトは **cleartool mkproject** を使用して作成した場合や、はじめて削除する場合はインテグレーション ストリームを削除することができます。プロジェクトの削除の詳細については、『**Rational ClearCase リファレンス ガイド**』にある **rmproject** のリファレンス ページを参照してください。

ストリーム

開発ストリームまたはインテグレーション ストリームは、次のすべての条件を満たす場合のみ削除することができます。

- ストリームにアクティビティがない。
- ストリーム内にベースラインが作成されていない。
- ストリームにビューがアタッチされていない。

さらに、プロジェクトに開発ストリームがある場合には、インテグレーション ストリームを削除することはできません。ストリームの削除に関する詳細については、『**Rational ClearCase リファレンス ガイド**』にある **rmstream** のリファレンス ページを参照してください。

コンポーネント

コンポーネントは、次のすべての条件を満たす場合のみ削除することができます。

- 初期ベースライン以外、コンポーネントのベースラインが存在しない。
- コンポーネントの初期ベースラインが、ほかのストリームの基本ベースラインとして機能していない。

コンポーネントの削除の詳細については、『**Rational ClearCase リファレンス ガイド**』にある **rmcomp** リファレンス ページを参照してください。

ベースライン

ベースラインは、次のすべての条件を満たす場合のみ削除することができます。

- ベースラインが基本ベースラインとして機能していない。
- ベースラインがコンポーネントの初期ベースラインではない。
- ストリームがベースラインに変更を加えていない。
- ベースラインがインクリメンタル ベースラインの基礎として使用されていない。

ベースラインの削除の詳細については、『**Rational ClearCase リファレンス ガイド**』にある **rmbl** リファレンス ページを参照してください。

アクティビティ

アクティビティは、次の 2 つの条件を満たす場合のみ削除できます。

- アクティビティの変更セット内にバージョンがない。
- 現在、アクティビティにビューが設定されていない。

イベント レコードの詳細については、『Rational ClearCase リファレンス ガイド』の **rmactivity** リファレンス ページを参照してください。

プロジェクトとストリームのロックと不要化

プロジェクトやストリームをプロジェクト エクスプローラに表示されないようにするには、オブジェクトをロックして、[不要にする] オプションを使用します。[不要にする] オプションを使用すると、オブジェクトは非表示になります。

1 プロジェクト エクスプローラで、非表示にするストリームまたはプロジェクトを選択して、[ファイル]、[プロパティ] をクリックし、プロパティ シートを表示します。

2 [ロック] タブをクリックして、[不要にする] をオンにします。[OK] をクリックします。

不要化したオブジェクトを表示するには、プロジェクト エクスプローラ で [表示]、[不要にされた項目を表示] をクリックします。

トリガを使用した開発ポリシーの実施

8

UCM には、GUI や CLI を使用してプロジェクトで容易に設定できる開発ポリシーが用意されています。また、特定の UCM 操作でトリガを使用して、プロジェクト チームに合わせてカスタマイズされた開発ポリシーを実施することもできます。この章では、トリガの作成方法と、トリガを使用して UCM プロジェクトの各種開発ポリシーを実装する方法について説明します。詳細については、`cleartool mktrigger` と `mktrtype` のリファレンス ページを参照してください。

トリガの概要

トリガとは、特定の ClearCase 操作が実行されるたびに 1 つ以上のプロシージャまたはアクションを起動するモニタです。一般にトリガは Perl スクリプト、バッチ スクリプト、シェルスクリプトを実行します。操作を特定のユーザーに限定したり、ユーザーが操作を実行できる条件を指定するときにトリガを使用できます。トリガを使用できる UCM 操作を次に示します。

- `chbl`
- `chfolder`
- `chproject`
- `chstream`
- `deliver`
- `mkactivity`
- `mkbl`
- `mkcomp`
- `mkfolder`
- `mkproject`
- `mkstream`
- `rebase`
- `rmbl`
- `rmcomp`
- `rmfolder`
- `rmproject`
- `rmstream`
- `setactivity`
- `setplevel`

操作前トリガと操作後トリガ

トリガは操作前トリガと操作後トリガの2種類に分類されます。操作前トリガは操作実行前に起動、つまり該当するプロシーダを実行します。操作後トリガは操作実行後に起動します。ユーザーが **ClearCase** コマンドを入力すると、**ClearCase** によりそのコマンドの操作前トリガがチェックされます。コマンドにトリガが関連付けられている場合には、**ClearCase** によりトリガ プロシーダが起動されます。トリガ プロシーダが失敗すると、**ClearCase** ではユーザーが要求した操作が拒否されます。トリガ プロシーダが成功すると、**ClearCase** により操作が実行されます。

特定の条件に一致しない場合にユーザーが操作を実行できないようにするには、操作前トリガを使用します。操作完了後にアクションを実行するには、操作後トリガを使用します。たとえば開発者がプロジェクトのインテグレーション ストリームに作業をデリバリーするたびにチーム メンバーに通知するには、デリバリー操作に操作後トリガを設定します。

トリガの範囲

トリガ タイプにより、**VOB** または **PVOB** で使用するトリガが定義されます。**cleartool mktrtype** コマンドを使用してトリガ タイプを作成するときに、トリガの範囲として次のいずれかを指定します。

- エレメント トリガ タイプは、1 つ以上のエレメントに適用されます。1 つ以上のエレメントにトリガ タイプのインスタンスを関連付けるには、**cleartool mktrigger** コマンドを使用します。
- すべてのエレメント トリガ タイプは、**VOB** のすべてのエレメントに適用されます。
- タイプ トリガ タイプは、**VOB** のタイプ オブジェクト (属性タイプなど) に適用されます。
- **UCM** トリガ タイプは、**PVOB** の **UCM** オブジェクト (ストリームまたはオブジェクトなど) に適用されます。
- すべての **UCM** オブジェクト トリガ タイプは、**PVOB** のすべての **UCM** オブジェクトに適用されます。

トリガでの属性の使用法

開発ポリシーを実施するトリガを設計するときに、属性を使用すると便利です。属性は、属性名/属性値のペアで表します。属性は属性タイプにより定義されます。属性をオブジェクト (ストリームやアクティビティなど) または特定のバージョンのエレメントに適用できます。トリガ スクリプトで属性値をテストし、トリガを起動するかどうかを決定できます。たとえば、エレメントがテスト済みであるかどうかを示すには、**TESTED** という属性タイプを定義し、**TESTED** 属性をエレメントに追加します。指定できる値は **Yes** と **No** です。

UCM トリガの代わりに ClearQuest スクリプトを使用する場合

この章では、UCM トリガのさまざまな使用例について説明します。UCM プロジェクトを Rational ClearQuest で使用できる場合には、次のポリシーを設定できます。これらのポリシーについては、61 ページの「UCM と ClearQuest の統合」で説明します。

- 作業前に ClearQuest のアクションを実行
- デリバー前に ClearQuest のアクションを実行
- アクティビティ変更前に ClearQuest のアクションを実行
- デリバー後に ClearQuest のアクションを実行
- アクティビティ変更後に ClearQuest のアクションを実行
- デリバー後に完了に遷移
- アクティビティ変更後に完了に遷移
- デリバー前に ClearQuest のマスターシップを転送
- デリバー後に ClearQuest のマスターシップを転送

上記の一部のポリシーには、ClearQuest グローバル フック スクリプトが関連付けられています。ClearQuest Designer でグローバル フック スクリプトを編集または置き換え、ご使用の環境に合わせてポリシーをカスタマイズすることができます。また、開発ポリシーを実施する ClearQuest フックを独自に作成することもできます。一般に、実施するポリシーで ClearQuest アクションが使用される場合には、前述の ClearQuest ポリシーまたは ClearQuest フックを使用します。実施するポリシーで ClearCase アクションが使用される場合には、UCM トリガを使用します。

操作によっては、ClearCase トリガと ClearQuest フックが関連付けられていることがあります。たとえば、開発者がデリバー操作を完了した時点で電子メールをチーム メンバーに送信するトリガを定義し、「デリバー後に ClearQuest のアクションを実行」ポリシーを有効にします。ClearCase と ClearQuest では、トリガ、フック、UCM 操作が次の順序で実行されます。

- 1 ClearCase 操作前トリガ
- 2 ClearQuest 操作前フック
- 3 UCM アクション
- 4 ClearQuest 操作後フック
- 5 ClearQuest アクティビティ遷移フック
- 6 ClearCase 操作後トリガ

UNIX と Windows 間でのトリガの共有

UNIX と Windows コンピュータの両方で正しく起動するトリガを定義することができます。ここでは 2 つの方法について説明します。1 つはプラットフォームごとに異なるパス名または異なるスクリプトを使用し、もう 1 つは両方のプラットフォームで同じスクリプトを使用します。

異なるパス名または異なるスクリプトの使用

UNIX または Windows、あるいはその両方で起動し、トリガ スクリプトを示すパス名が異なるトリガを定義するには、`mktrtype` コマンドに `-execunix` オプションと `-execwin` オプションを指定します。これらのオプションは、該当するプラットフォーム (UNIX または Windows) 上で起動されたときに `-exec` と同じ振る舞いをします。該当しないプラットフォーム上では何もしません。この方法を使用すると、1 つのトリガ タイプが、同一スクリプトに対しての別のパスを使用したり、UNIX と Windows コンピュータ上でまったく異なるスクリプトを使用することができます。たとえば、次のように指定します。

```
cleartool mktrtype -element -all -nc -preop checkin
-execunix /public/scripts/precheckin.sh
-execwin ¥¥neon¥scripts¥precheckin.bat
pre_ci_trig
```

メモ: このコマンド行の例は、読みやすいように複数の行に分割されています。入力時にはすべて 1 行に入力してください。

UNIX 上では、スクリプト `precheckin.sh` だけが実行されます。Windows 上では `precheckin.bat` だけが実行されます。

新しいプラットフォームでユーザーがこのトリガ プロセスを回避できないようにするため、`-execunix` だけを指定したトリガは Windows 上では常に失敗します。同様に、`-execwin` だけを指定したトリガは UNIX 上では失敗します。

同一スクリプトの使用

Windows と UNIX プラットフォームで同一のトリガ スクリプトを使用する場合、両方のオペレーティング システムで実行できるパッチ コマンド インタープリタを使用する必要があります。このため、Rational ClearCase には `ccperl` プログラムが組み込まれています。このプログラムは、Windows と UNIX の両方で使用できる Perl プログラムです。

次の `mktrtype` コマンドは、`pre_ci_trig` というサンプル トリガ タイプを作成し、実行可能トリガ スクリプトとして `precheckin.pl` を指定します。

```
cleartool mktrtype -element -all -nc -preop checkin ¥
-execunix 'Perl /public/scripts/precheckin.pl' ¥
-execwin 'ccperl ¥¥neon¥scripts¥precheckin.pl' ¥
pre_ci_trig
```

ヒント

- オペレーティング システムに基づいてスクリプトの実行を調整するには、Perl スクリプト内で環境変数を使用します。
- 対話式に情報を収集して表示するために、`clearprompt` コマンドを使用することができます。

順次デリバー操作の実施

UCM では複数の開発者が同一のインテグレーション ストリームに作業を同時にデリバーできるので、複数の開発者が同一エレメントに対する変更をデリバーしようとする競合が発生します。ある開発者がエレメントをチェックアウトすると、最初のデリバー操作が完了またはキャンセルするまで、2 番目の開発者がこのエレメントに対する変更をデリバーすることはできません。2 番目のデリバー操作はチェックアウト済みエレメントを除くすべてのエレメントのチェックアウトを試みますが、操作のマージ フェーズへは進みません。2 番目の開発者は、最初のデリバー操作が完了するまで待つか、2 番目のデリバー操作を取り消します。

デリバーが同時に実行される結果として発生する混乱を回避する開発ポリシーを実装できます。この項では、複数の開発者が作業を同一インテグレーション ストリームに同時にデリバーすることを防止する 3 種類の Perl スクリプトを示します。

- スクリプト 1 は、トリガ タイプと属性タイプを作成します。
- スクリプト 2 は、デリバー操作の開始時に起動する操作前トリガアクションです。
- スクリプト 3 は、デリバー操作の終了時に起動する操作後トリガアクションです。

セットアップスクリプト

このセットアップ スクリプトでは、操作前トリガ タイプ、操作後トリガ タイプ、属性タイプが作成されます。操作前トリガアクションはデリバー操作開始時に起動します。この起動条件は `deliver_start` という操作の種類 (`opkind`) により示されます。操作後トリガアクションは、デリバー操作がキャンセルまたは完了したときに起動します。この起動条件はそれぞれ `deliver_cancel` と `deliver_complete` という `opkind` により示されます。

このセットアップ スクリプトは Windows プラットフォームと UNIX プラットフォームの両方で実行できます。Windows または UNIX で PVOB があるかどうかによって、Windows での操作前スクリプトと操作後スクリプトを実行するためのコマンド行構文が多少異なります。このため、セットアップ スクリプトでは IF ELSE ブール式を使用して適切な PVOB タグを設定します。

`mkttrtype` コマンドで `-ucmobject` オプションと `-all` オプションを使用して、このトリガ タイプが PVOB のすべての UCM オブジェクトに適用されることを指定しますが、`-stream` オプションにより、範囲が 1 つのインテグレーション ストリームに限定されます。

操作前トリガ スクリプト

この操作前トリガ アクションは、開発者が指定のインテグレーション ストリームへの作業のデリバーを開始するときに起動されます。このスクリプトは、タイプ `deliver_in_progress` の属性をインテグレーション ストリームに関連付けます。同一ストリームに別の開発者が作業をデリバー中である場合、`mkattr` コマンドは失敗し、開発者にデリバー操作を後で再試行するよう通知するメッセージが表示されます。同一ストリームに作業をデリバー中である開発者がほかにはいない場合には、`mkattr` コマンドが成功し、現在のデリバー操作が完了するまではほかの開発者による同一インテグレーション ストリームへの作業のデリバーが禁止されます。

```
use Config;

my $PVOBTAG;
my $tempfile;
my $exit_value;

if ($Config{'osname'} eq 'MSWin32') {
    $PVOBTAG = '¥cyclone-pvob';
    $tempfile =
$ENV{TMP}."¥¥expreop.".$ENV{"CLEARCASE_PPID"}.".txt";
}
else {
    $PVOBTAG = '';
}

my $STREAM = "stream: ".$ENV{"CLEARCASE_STREAM"};
my $ATYPE = "atype:deliver_in_progress¥@$PVOBTAG";

print $STREAM."¥n";
print $ATYPE."¥n";

my $cmdline;
my $cmdoutput;

# deliver in progress 属性がターゲット ストリームに
# 適用されているかどうかを確認するためのテスト。

$msg = 'cleartool describe -fmt "%a" $STREAM ' ;
print $msg."¥n";

if ($ENV{"CLEARCASE_CMDLINE"} eq "") {
    open(TEMPFH, "> $tempfile");
    print TEMPFH $msg."¥n";
    close(TEMPFH);
    $cmdline = "clearprompt text -outfile $tempfile -multi_line
-dfile ¥"$tempfile¥" -prompt ¥"Describe Results¥"";
    $cmdoutput = '$cmdline ' ;
}
```

```

if (index($msg, "deliver_in_progress") >=0) {
    print "***¥n";
    print "*** すでに実行中のデリバリー操作があります。後で再試行してください。¥n";
    print "***¥n";
    exit 1;
}

$cmdline = "cleartool mkattr -default $ATTTYPE $STREAM";
print $cmdline."¥n";

$msg = '$cmdline ';
$exit_value = $? >> 8;
if (!(($exit_value eq 0)) {
    print "***¥n";
    print "*** このデリバリー操作を実行する直前に別のデリバリー操作が開始されました。¥n";
    print "*** そのデリバリー操作がすでに実行中であるので、後で再試行してください。¥n";
    print "***¥n";
    exit 1;
}
exit 0;

```

操作後トリガ スクリプト

この操作後トリガアクションは、開発者が指定のインテグレーション ストリームに対するデリバリー操作をキャンセルまたは完了した時点で起動されます。このスクリプトは、デリバリー操作開始時に操作前スクリプトがインテグレーション ストリームに関連付けた **deliver_in_progress** 属性を削除します。属性が削除されたら、別の開発者がこのインテグレーション ストリームに作業をデリバリーできます。

```

# deliver_complete または deliver_cancelperl 操作後トリガで起動する perl スクリプト。
use Config;

# プラットフォーム依存引数の定義。
my $PVOBTAG;
if ($Config{'osname'} eq 'MSWin32') {
    $PVOBTAG = '¥cyclone-pvob';
}
else{
    $PVOBTAG = '';
}

```

```
my $STREAM = "stream:". $ENV{"CLEARCASE_STREAM"};
my $ATYPE = "atype:deliver_in_progress¥@$PVOBTAG";

# デリバーを許可する属性の削除。
print `cleartool rmattr -nc $ATYPE $STREAM`;
```

開発者へのデリバー操作に関するメールの送信

プロジェクトチームの開発者間の意思伝達を促進するために、開発者がデリバー操作を完了するたびにチームメンバーに電子メールメッセージを送信するトリガタイプを作成できます。この項では次の2つのスクリプトを示します。

- スクリプト1は、デリバー操作が正常終了すると起動するトリガタイプを作成します。
- スクリプト2は、開発者に電子メールメッセージを送信する操作後トリガアクションです。

セットアップスクリプト

このスクリプトは、開発者がデリバー操作を完了した時点で起動する操作後トリガタイプを作成します。このトリガ起動条件は **deliver_complete opkind** により示されます。 **mkttrtype** コマンドの **-stream** オプションにより、指定のインテグレーションストリームをターゲットとするデリバー操作だけにこのトリガタイプが適用されることが指定されます。

```
# デリバー操作に関する電子メール通知を送信する
# トリガタイプをセットアップする Perl スクリプト。
use Config;

# プラットフォーム依存引き数を定義する。
my $PVOBTAG;
if ($Config{'osname'} eq 'MSWin32') {
    $PVOBTAG = '¥cyclone_pvob';
    $WCMD = '-execwin "ccperl
¥¥¥¥pluto¥disk1¥ucmtrig_examples¥ex2¥ex2_postop.pl"';
}
else {
    $PVOBTAG = '/pvobs/cyclone_pvob';
    $WCMD = '-execwin "ccperl
¥¥¥¥¥¥pluto¥disk1¥ucmtrig_examples¥ex2¥ex2_postop.pl"';
}
my $STREAM = "stream:P1_int¥@$PVOBTAG";
my $TRTYPE = "trtype:ex2_postop¥@$PVOBTAG";
my $UCMD = '-execunix "Perl
/net/pluto/disk1/ucmtrig_examples/ex2/ex2_postop.pl"';

print 'cleartool mkttrtype -ucmobject -all -postop deliver_complete
$WCMD $UCMD -stream $STREAM -nc $TRTYPE`';
```

操作後トリガ スクリプト

この操作後トリガ アクションは、管理者がインテグレーション ストリームへの作業のデリバーを完了した時点で起動されます。このスクリプトは、デリバー操作が完了したことを通知する電子メール メッセージを作成し、プロジェクト チームのほかの開発者にこのメッセージを送信します。このスクリプトは、**ClearCase** 環境変数を使用して、メッセージの本文にデリバー操作に関する次の詳細情報を挿入します。

- プロジェクト名
- 作業をデリバーした開発ストリーム
- デリバーされた作業を受け取ったインテグレーション ストリーム
- デリバー操作により作成されたインテグレーション アクティビティ
- デリバーされたアクティビティ
- デリバー操作で使用されたインテグレーション ビュー

デリバー操作完了時にメールを送信する Perl スクリプト。

```
#####  
# Mail::Send の "open" メソッドを上書きする単純なパッケージであるので、  
# メール送信メカニズムを制御できる。  
  
package SendMail;  
  
use Config;  
use Mail::Send;  
  
@ISA = qw(Mail::Send);  
  
sub open {  
    my $me = shift;  
    my $how; # How to send mail  
    my $notused;  
    my $mailhost;  
  
    # Windows では SMTP を使用する。  
  
    if ($Config{'osname'} eq 'MSWin32') {  
        $how = 'smtp';  
        $mailhost = "localmail0.company.com";  
    }  
  
    # それ以外の場合は Mail::Mailer により提供されるデフォルト値を使用する。  
    Mail::Mailer->new($how, $notused, $mailhost)->open($me);  
}  
  
#  
#####  
# メイン プログラム  
  
    my @to = "developers¥@company.com";
```

```

my $subject = " デリバー完了 ";
my $body = join '', ("¥n",
    "UCM プロジェクト： ", $ENV{CLEARCASE_PROJECT}, "¥n",
    "UCM ソース ストリーム： ", $ENV{CLEARCASE_SRC_STREAM}, "¥n",
    "UCM 宛先ストリーム： ", $ENV{CLEARCASE_STREAM}, "¥n",
    "UCM インテグレーション アクティビティ： ",
$ENV{CLEARCASE_ACTIVITY}, "¥n",
    " デリバーされた UCM アクティビティ： ", $ENV{CLEARCASE_DLVR_ACTS},
"¥n",
    "UCM ビュー： ", $ENV{CLEARCASE_VIEW_TAG}, "¥n"
);

my $msg = new SendMail(Subject=>$subject);
$msg->to(@to);
my $fh = $msg->open($me);
$fh->print($body);
$fh->close();
1; # return success
#
#####

```

インテグレーション ストリームでのアクティビティの作成禁止

インテグレーション ストリームに関連付けられているインテグレーション ビューを所有するユーザーなら誰でも、そのストリームにアクティビティを作成できます。ただし、UCM プロセスでは開発者が各自の開発ストリームでアクティビティを作成する必要があります。開発者が誤ってインテグレーション ストリームにアクティビティを作成することを防止するポリシーを実装できます。この項では、このポリシーを実施する Perl スクリプトを示します。

操作前トリガ タイプ **block_integration_mkact** を作成する **mktrtype** コマンドを次に示します。このトリガ タイプは、開発者がアクティビティを作成しようとする时起動されます。

```

cleartool mktrtype -ucmobject -all -preop mkactivity -execwin "ccperl ^
¥¥pluto¥disk1¥triggers¥block_integ_mkact.pl" -execunix "Perl ^
/net/jupiter/triggers/block_integ_mkact.pl" block_integration_mkact@¥my_pvob

```

block_integration_mkact トリガの起動時に実行される操作前トリガ スクリプトを次に示します。このスクリプトは、**cleartool lsproject** コマンドと **CLEARCASE_PROJECT** 環境変数を使用してプロジェクトのインテグレーション ストリーム名を判別します。**ClearCase** により、デリバー操作中に行われる変更内容を追跡するインテグレーション アクティビティが作成されます。このスクリプトは、**CLEARCASE_POP_KIND** 環境変数を使用して、作成されるアクティビティがインテグレーション アクティビティであるかどうかを判別します。デリバー操作の結果 **mkactivity** 操作が実行されると、親操作を識別する **CLEARCASE_POP_KIND** の値は **deliver_start** になります。

CLEARCASE_POP_KIND の値が **deliver_start** でない場合には、アクティビティがインテグレーション アクティビティではないため、**mkactivity** 操作が禁止されます。

```
# このプロジェクトのインテグレーション ストリーム名を取得する。
my $istream = `cleartool lsproject -fmt "[%i]stream)p"
$ENV{"CLEARCASE_PROJECT"} `;

# 現在のストリームを取得し、VOB タグを削除する。
$_ = $ENV{"CLEARCASE_STREAM"};
s/¥@.*///;
my $curstream = $_;

# ユーザーのストリームと同一である場合、これはインテグレーション ストリームである
if ($istream eq $curstream) {
    # Only allow this mkact if it is a result of a deliver
    # Determine this by checking the parent op kind
    if ($ENV{"CLEARCASE_POP_KIND"} ne "deliver_start") {
        print " アクティビティを作成できるストリームはデリバー用インテグレーション
        ストリームだけです。¥n";
        exit 1
    }
}

exit 0
```

役割に基づいたアクセス制御システムの実装

ClearCase 環境ではユーザーが異なる役割を担うので、特定の ClearCase 操作へのアクセスを役割に基づいて制限できます。この項では、役割に基づいたアクセス制御システムを実装するトリガ定義とスクリプトを示します。

操作前トリガ タイプ **role_restrictions** を作成する **mktrtype** コマンドを次に示します。このトリガタイプは、ユーザーがベースライン、ストリーム、アクティビティのいずれかを作成しようとするすると起動されます。

```
cleartool mktrtype -nc -ucmobject -all -preop mkstream,mkbl,mkactivity ¥
-execunix "perl /net/jupiter/triggers/role_restrictions.pl" ¥
-execwin "ccperl ¥¥pluto¥disk1¥triggers¥role_restrictions.pl" ¥
role_restrictions@¥my_pvob
```

操作前トリガ スクリプト

ユーザーを次の役割へマップする操作前トリガ スクリプトを示します。

- プロジェクト マネージャー
- 統合担当者
- 開発者

このスクリプトは、mkactivity、mkbl、mkstream 操作を、それぞれの操作を実行できる役割にマップします。たとえば、プロジェクト マネージャーまたは統合担当者に指定されているユーザーだけがベースラインを作成できます。

このスクリプトは、CLEARCASE_USER 環境変数を使用してユーザー名を取得し、CLEARCASE_OP_KIND 環境変数を使用してユーザーが実行する操作を確認し、CLEARCASE_POP_KIND 環境変数を使用して親操作を確認します。このスクリプトでは、親操作が deliver または rebase の場合には権限をチェックしません。

```
use strict;

sub has_permission
{
    my ($user,$op,$pop,$proj) = @_;

    # 複合操作 ('deliver' や 'rebase' など) を実行する場合、
    # この複合操作を構成する
    # 個々のサブ操作に関する権限チェックは必要ない。

    return 1 if($pop eq 'deliver_start' || $pop eq 'rebase_start' ||
                ($pop eq 'deliver_complete' || $pop eq
'rebase_complete' ||
                ($pop eq 'deliver_cancel' || $pop eq
'rebase_cancel'));

    # どの役割がどの操作を実行するか？
    # プロジェクトごとに制御できるように、これらのマップは、
    # トリガ スクリプトのこの部分にハード コーディングされるのではなく、
    # 各プロジェクトの ClearCase 属性に格納される点に注意。

    my %map_op_to_roles = (
        mkactivity => [ "projectmgr", "integrator", "developer" ],
        mkbl       => [ "projectmgr", "integrator" ],
        mkstream   => [ "projectmgr", "integrator", "developer" ],
    );

    # どのユーザーがどの役割に所属しているか？
```

```

my %map_role_to_users = (
    projectmgr => [ "kate" ],
    integrator => [ "kate", "mike" ],
    developer  => [ "kate", "mike", "jones" ],
);

# この操作を実行できる役割にユーザーが属しているか？

my ($role,$tmp_user);

for $role (@{ $map_op_to_roles{$op} }) {
    for $tmp_user (@{ $map_role_to_users{$role} }) {
        if ($tmp_user eq $user) {
            return 1;
        }
    }
}

return 0;
}
sub Main
{
    my $user = $ENV{CLEARCASE_USER};
    my $proj = $ENV{CLEARCASE_PROJECT};
    my $op   = $ENV{CLEARCASE_OP_KIND};
    my $pop  = $ENV{CLEARCASE_POP_KIND};

    my $perm = has_permission($user, $op, $proj);

    printf("$user %s permission to perform '$op' in project
    $proj\n",
        $perm ? "has" : "does NOT have");

    exit($perm ? 0 : 1);
}

Main();

```

UCM トリガのその他の使用法

これまでの項の例では、UCM トリガを使用して開発ポリシーを実施するいくつかの方法を示しました。ほかにも次のような UCM トリガの使用法があります。

- UCM と変更依頼管理 (CRM) システムとの間でのインテグレーションの作成。ほとんどの場合、**ClearQuest** のすぐに利用できるインテグレーションを使用しますが、別の CRM システムと統合することもできます。次のいずれかの方法で別の CRM システムと統合できます。
 - **mkactivity** を使用して、開発者が新しいアクティビティを作成するときに CRM データベースに対応するレコードを作成するトリガ タイプを作成する。
 - **setactivity** を使用して、開発者がアクティビティの処理を開始するときに、CRM データベースのレコードをスケジュール済み状態に遷移するトリガ タイプを作成する。
 - **deliver** を使用して、開発者がインテグレーション ストリームへのアクティビティのデリバリーを完了した時点で、CRM データベースのレコードを **completed** 状態へ遷移するトリガ タイプを作成する。
- **rebase** を使用して、開発者による特定の開発ストリームのリベースを防止するトリガ タイプを作成する。特定の 1 つのバグを修正するために使用される開発ストリームに対してこのポリシーを実施できます。
- **setactivity** を使用して、特定の開発者が特定のアクティビティを処理できるようにするトリガ タイプを作成する。

複数プロジェクトの 並行リリースの管理

9

前章では、単一のプロジェクトの管理方法について説明しました。しかし、プロジェクトの複数リリースを同時に管理する場合があります。そのためには、1つのプロジェクトから別のプロジェクトへ変更をマージする必要があります。この章では、次の一般的な2つの場合でのマージ方法について説明します。

- 現在のプロジェクトと次回プロジェクトの同時管理
- パッチ リリースのプロジェクト新規リリースへの組み込み

この章では、ベース ClearCase ツールを使用して UCM プロジェクトからベース ClearCase プロジェクトに作業をマージする方法についても説明します。

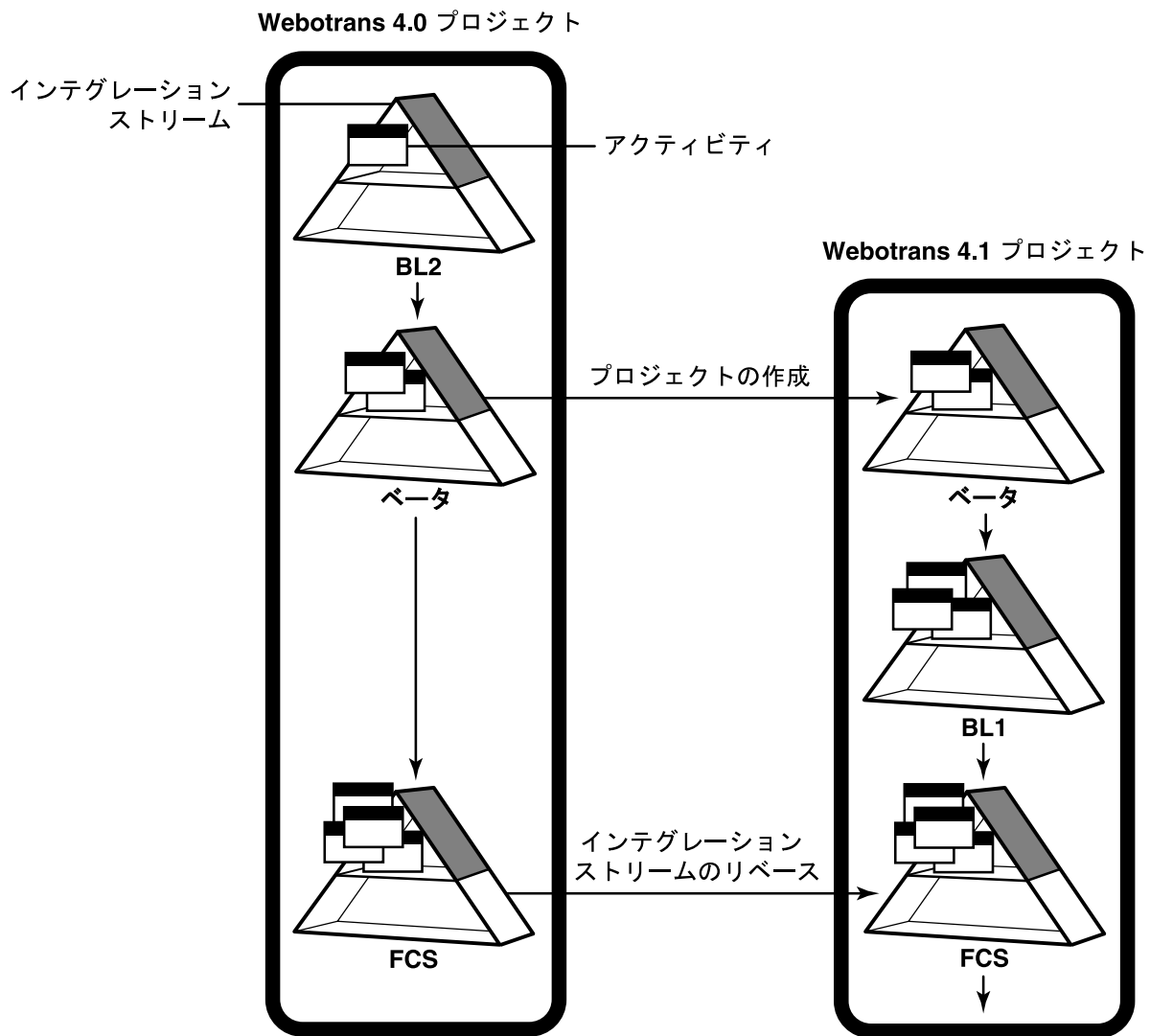
現在のプロジェクトと次回プロジェクトの同時管理

多くの開発チームで行われていることですが、ソフトウェア開発の厳しいスケジュールの中では、プロジェクトの新規リリースの開発を現在のリリースが完了する前に開始することは一般的に行われています。次回リリースで新規の機能を追加する場合や、現在のリリースを別のプラットフォームに移植する場合があります。

例

図 37 は、現在のプロジェクトである Webotrans 4.0 と、次回のプロジェクトである Webotrans 4.1 を示します。

図 37 次回リリースの管理



この例では、以下ようになります。

- 次回プロジェクトのプロジェクト マネージャーは、Webotrans 4.0 プロジェクトに使用されているコンポーネントの Beta ベースラインを基に、Webotrans 4.1 プロジェクトを作成します。両方のプロジェクト チームの開発者がそれぞれに変更を加え、両方の統合担当者がそれらの変更を組み込んだ新規ベースラインを作成します。
- 4.0 のチームが作業を完了すると、統合担当者は FCS と呼ばれる最終ベースラインを作成します。次に、4.1 のプロジェクト マネージャーは、4.1 インテグレーション ストリームをその FCS ベースラインにリベースします。

プロジェクト間のリベース操作の実行

インテグレーション ストリームを、別のプロジェクトのインテグレーション ストリームのベースライン セットにリベースするには、次の手順を実行します。

- 1 ClearCase プロジェクト エクスプローラで、リベースするインテグレーション ストリームを選択します。
- 2 [ツール] メニューの [ストリームのリベース] をクリックします。
- 3 [リベースのプレビュー] ダイアログ ボックスで、[詳細設定] をクリックします。
- 4 [リベース構成の変更] ダイアログ ボックスで、ストリームをリベースするベースラインが含まれているコンポーネントを選択します。[変更] をクリックします。
- 5 [ベースラインの変更] ダイアログ ボックスで次の手順を実行します。

Windows では [変更] をクリックします。

UNIX では [元のストリーム] フィールドの端にある矢印をクリックします。

- 6 Windows では、[ストリームの選択] ダイアログ ボックスでほかのプロジェクトのインテグレーション ストリームへ移動します。インテグレーション ストリームを選択し、[OK] をクリックします。

UNIX では、ほかのプロジェクトのインテグレーション ストリームを選択します。

これにより、[ベースラインの変更] ダイアログ ボックスが更新され、ほかのプロジェクトのインテグレーション ストリームで使用可能なベースラインのセットが表示されます。

- 7 [ベースラインの変更] ダイアログ ボックスで、コンポーネントを選択します。[ベースライン] リストに、ほかのプロジェクトのインテグレーション ストリームで選択されているコンポーネントに使用可能なベースラインがすべて表示されます。インテグレーション ストリームのリベース先ベースラインを選択します。[OK] をクリックします。選択したベースラインが、[リベース構成の変更] ダイアログ ボックスに表示されます。
- 8 インテグレーション ストリームのリベース先ベースラインをすべて選択するまで、手順 4 から手順 7 までを繰り返します。

- 9 [OK] をクリックして、[リベース構成の変更] ダイアログ ボックスを閉じます。
[リベースのプレビュー] ダイアログ ボックスで [OK] をクリックします。
- 10 ClearCase では、競合しない変更は自動的にマージされます。ClearCase は競合する変更を検出すると、差分マージを開始するよう要求します。差分マージは、競合する変更を解決するツールです。差分マージの使用法に関する詳細については、差分マージのヘルプと『Rational ClearCase ソフトウェア開発ガイド』を参照してください。

プロジェクトのインテグレーション ストリームをリベースできるのは、リベースするベースラインがインテグレーション ストリームの現在の基本ベースラインの後続バージョンである場合のみです。この例では、FCS ベースラインは Beta ベースラインの後続バージョンであり、Beta ベースラインは Webotrans 4.1 のインテグレーション ストリームの現在の基本ベースラインです。

プロジェクト新規バージョンへのパッチ リリースの組み込み

並行開発で一般的なもう 1 つのシナリオは、プロジェクトのパッチ リリースと新規リリースの作業を同時に行うことです。ここでは、このシナリオについて説明します。

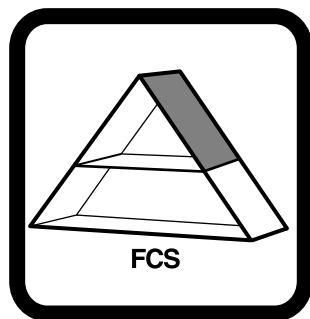
例

図 38 は、パッチ リリースと新規リリースの流れを示します。この例では、以下のようになります。

- Webotrans 3.0 Patch と Webotrans 4.0 プロジェクトは、共に Webotrans 3.0 プロジェクトのコンポーネントの FCS ベースラインを基本ベースラインとして使用しています。パッチリリースの目的は、Webotrans 3.0 がリリースされた後に発見された問題を解決することです。Webotrans 4.0 は、Webotrans 製品の次回メジャー リリースを表しています。
- 3.0 Patch と 4.0 の両方のプロジェクトで開発が行われ、それぞれの統合担当者が定期的にベースラインを作成します。
- 3.0 Patch プロジェクトの開発者の作業が完了すると、統合担当者は最終変更を BL2 ベースラインに組み込みます。次に統合担当者はこれらの変更を、3.0 Patch のインテグレーション ストリームから 4.0 のインテグレーション ストリームにデリバリーして、4.0 のプロジェクトに解決策が反映されるようにします。

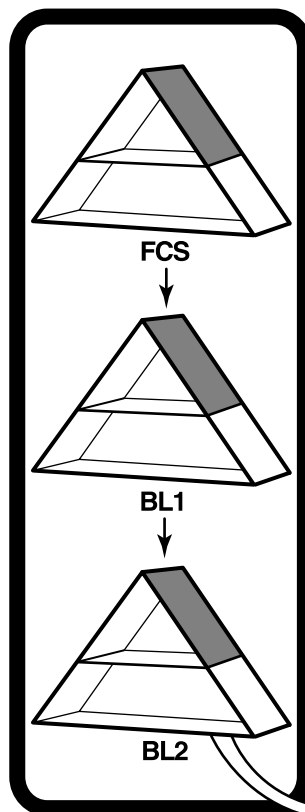
図 38 パッチ リリースの組み込み

Webotrans 3.0 プロジェクト

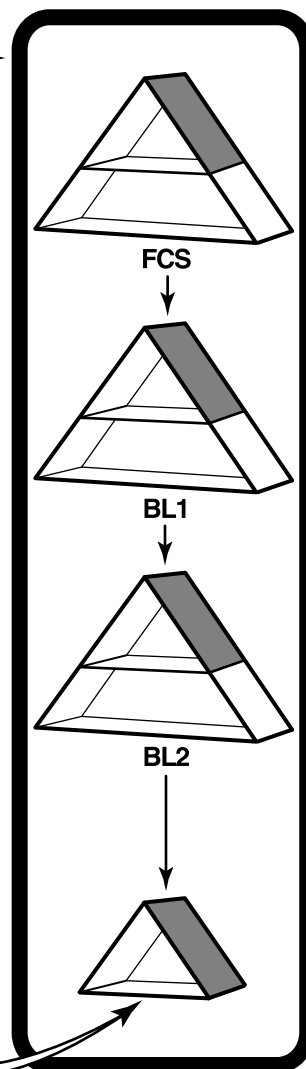


プロジェクトの作成

Webotrans 3.0 プロジェクト
Patch



Webotrans 4.0 プロジェクト



マージ

ほかのプロジェクトへの作業のデリバー

あるプロジェクトのインテグレーション ストリームの作業を別のプロジェクトのインテグレーション ストリームにデリバーするには、次の手順を実行します。

- 1 ソース ストリームで、デリバーする変更を組み込む 1 つまたは複数のベースラインを選択します。インテグレーション ストリームから作業をデリバーするときには、ベースラインをデリバーする必要があります。
- 2 ターゲット インテグレーション ストリームのデリバー ポリシー設定で、ほかのプロジェクトからのデリバーが許可されていることを確認してください。プロジェクト エクスプローラで、ターゲット インテグレーション ストリームを選択し、[ファイル] をポイントして [ポリシー] をクリックします。[別プロジェクトのプロジェクトまたはストリームへのデリバーを許可する] ポリシーが有効になっていない場合には、プロジェクト マネージャーにこの設定を有効にするように依頼します。
- 3 プロジェクト エクスプローラでソース インテグレーション ストリームを選択し、[ツール]、[ベースラインのデリバー] の順にポイントし、[デフォルト ターゲットへ] または [別のターゲットへ] をクリックします。インテグレーション ストリームのデフォルト デリバー ターゲットを確認するには、[ファイル] をポイントして [プロパティ] をクリックします。[一般] タブの [デリバー先] ボックスに、デフォルト デリバー ターゲットが示されています。デフォルト デリバー ターゲットを変更するには、[変更] をクリックします。[別のターゲットへ] オプションを選択すると、[ストリームからデリバー (別のターゲット)] ダイアログ ボックスが表示されます。このダイアログ ボックスではターゲット ストリームを選択できます。
- 4 [ストリームからのデリバー プレビュー] ダイアログ ボックスで [追加]、[変更]、[削除] を使用して、デリバーするベースラインを選択します。[ビュー] ボックスに、ターゲット インテグレーション ストリームに関連付けられているビューが表示されていることを確認します。必要に応じて [変更] をクリックし、別のビューを選択します。[OK] をクリックし、デリバー操作のマージ部分を開始します。
- 5 Rational ClearCase では、競合しない変更は自動的にマージされます。競合する変更が検出されると、差分マージを開始するよう要求されます。差分マージは、競合する変更を解決するツールです。差分マージの使用法に関する詳細については、差分マージのヘルプと『Rational ClearCase ソフトウェア開発ガイド』を参照してください。
- 6 ファイルのマージが完了したら、[完了] をクリックします。変更内容がチェックインされます。

メインライン プロジェクトの使用法

チームがシステムの複数のバージョンを開発、リリースする場合には、メインライン プロジェクトを作成できます。メインライン プロジェクトは、一定期間内に存続する関連プロジェクトを統合した 1 つのプロジェクトとして使用できます。メインライン プロジェクトは 1 つのリリースに固有のものではありません。

たとえば Webotrans チームが 6 ヶ月ごとに製品の新バージョンを開発してリリースするとします。プロジェクト マネージャーは、新しいバージョンごとに、前回のプロジェクトのインテグレーション ストリームの最終推奨ベースラインを基本ベースラインとして使用するプロジェクトを作成できます。たとえば、Webotrans 2.0 の基本ベースラインは Webotrans 1.0 のインテグレーション ストリームの最終推奨ベースラインであり、Webotrans 3.0 の基本ベースラインは Webotrans 2.0 のインテグレーション ストリームの最終推奨ベースラインです。この手法は、カスケード型プロジェクト設計と呼ばれます。この手法の欠点は、Webotrans プロジェクトの履歴全体を確認するときにすべてのインテグレーション ストリームを参照しなければならない点です。

メインライン プロジェクトを使用する手法では、Webotrans プロジェクト マネージャーが初期ベースライン セットを使用してメインライン プロジェクトを作成し、この初期ベースラインに基づいて Webotrans 1.0 を作成します。開発者による Webotrans 1.0 の作業が完了すると、プロジェクト マネージャーは最終推奨ベースラインをメインライン プロジェクトのインテグレーション ストリームにデリバーします。メインライン プロジェクトのインテグレーション ストリームにデリバーされたこれらのベースラインは、Webotrans 2.0 プロジェクトの基本ベースラインとして使用されます。Webotrans 2.0 チームによる作業が完了すると、プロジェクト マネージャーは最終推奨ベースラインをメインライン プロジェクトのインテグレーション ストリームにデリバーします。この手法の長所は、各プロジェクトの最終推奨ベースラインが、メインライン プロジェクトのインテグレーション ストリームで使用可能である点です。

プロジェクトから非 UCM ブランチへのマージ

開発チームの一部のメンバーはプロジェクトで作業し、残りのメンバーはベース ClearCase で作業している場合があります。ClearCase を長い間使用しているユーザーの場合、まずシステムのごく一部に UCM を使用するようになります。この方法により、ベース ClearCase から UCM に一度に移行するのではなく段階的に移行することができます。

この場合、プロジェクトのインテグレーション ストリームからシステムのインテグレーション ブランチとして機能するブランチへ、作業を定期的にマージする必要があります。作業を定期的にマージするには、ベース ClearCase 機能を使用して変更をマージする次のようなスクリプトを使用します。

```

# UCM プロジェクトの内容を非 UCM プロジェクトにデリバーする
# サンプル Perl スクリプト。宛先ブランチを参照するビューへの設定時に
# このスクリプトを実行する。
#
# 使用法: Perl <this-script> <project-name> <project-vob>

use strict;

my $mergeopts = '-print';
my $project = shift @ARGV;
my $pvob = shift @ARGV;
my $bl;

chdir ($pvob) or die("can't cd to project VOB '$pvob'");

print("##### プロジェクト '$project' の推奨ベースラインの取得 ¥n");

my @recbls = split(' ', `cleartool lsproject -fmt "%[rec_bls]p"
$project`);
foreach $bl (@recbls) {
    my $comp = `cleartool lsbl -fmt "%[component]p" $bl`;
    my $vob = `cleartool lscomp -fmt "%[root_dir]p" $comp`;
    print("##### $vob のベースライン '$bl' からの変更のマージ ¥n");
    my $st = system("cleartool findmerge $vob -fver $bl
$mergeopts");
    $st == 0 or die("findmerge error");
}

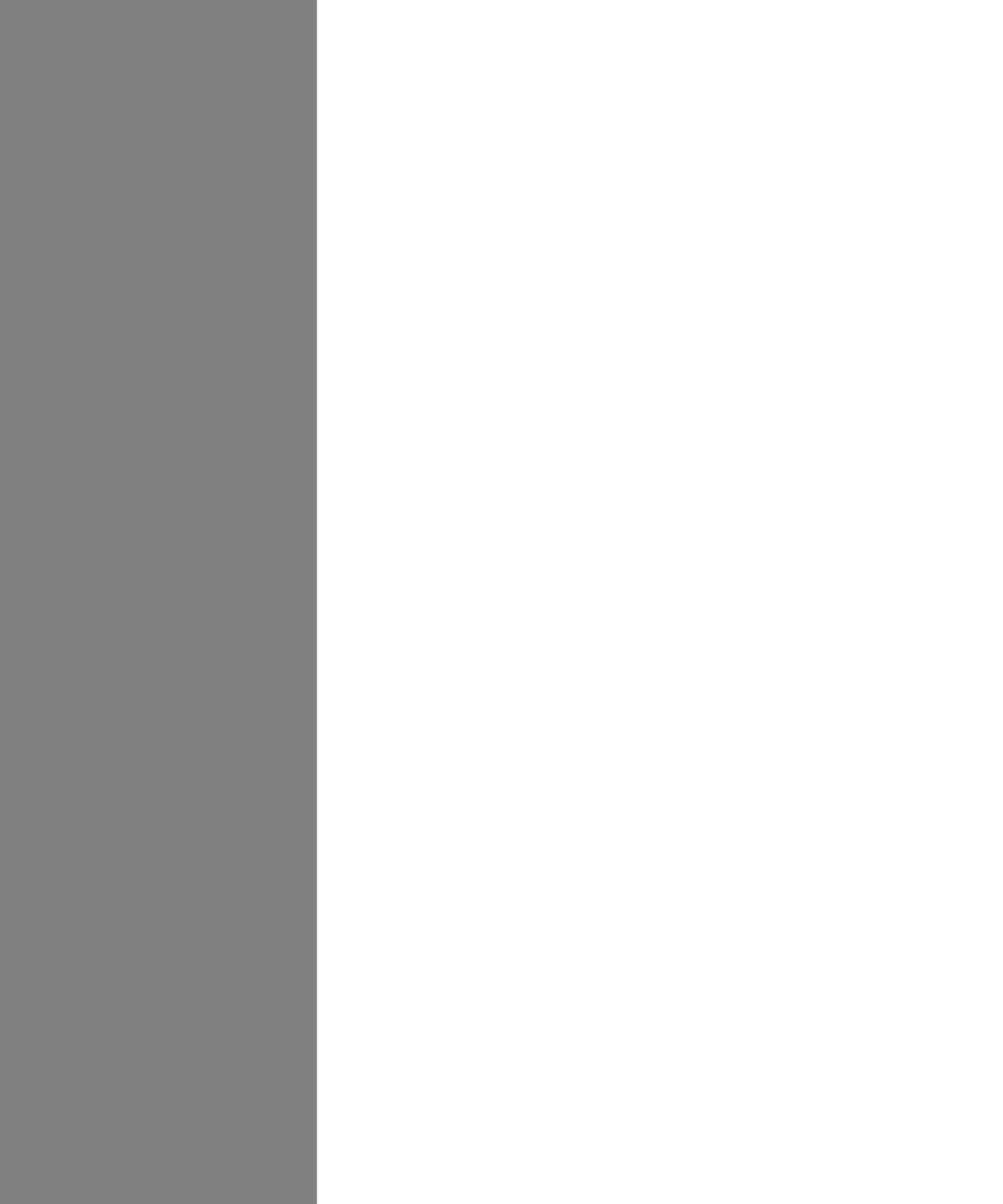
exit 0;

```

このスクリプトはマージ元のインテグレーション ストリームの推奨ベースラインを検出します。次に、**cleartool findmerge** コマンドを使用して、推奨ベースラインのバージョンとターゲット ブランチの最新バージョンの相違点を検出します。詳しくは、**findmerge** のリファレンス ページを参照してください。

このスクリプトを使用する際には、事前にサイトに応じたエラー処理などのロジックをスクリプトに追加することをお勧めします。

ベース ClearCase での作業



ベース ClearCase での プロジェクト管理

10

プロジェクト マネージャーには、ソフトウェア開発プロジェクトの技術的側面を計画、人員配置、管理する責任があります。プロジェクト マネージャーは、作業内容の決定、プロジェクトのチーム メンバーへの作業の割り当て、作業スケジュールの作成、作業を行う上でのポリシーや手順の決定を行います。

開発中には、プロジェクト マネージャーは進捗状況を監視してプロジェクト状態のレポートを作成します。また、ビルドやその後のベースラインにある特定の作業項目を承認することもあります。

プロジェクト マネージャーはプロジェクト統合担当者でもあります。つまり、それぞれの開発者が完了した作業を、デリバリー可能でビルド可能なシステムに組み込む責任があります。プロジェクト マネージャーはプロジェクトのベースラインを作成し、それらのベースラインの品質レベルを設定します。

ベース ClearCase には多くの機能があり、この作業を簡単に行うことができます。開発を開始する前には、次に示すいくつかの計画と設定タスクを完了する必要があります。

- プロジェクト環境の設定
- 開発ポリシーの実装
- 統合ポリシーの定義と実装

この章では、これらの概要について説明します。残りの章では実装上の詳細について説明します。「第 16 章 開発サイクル全体を通じた ClearCase の使用法」では、開発サイクル全体を通して、プロジェクトで ClearCase を使用方法について示します。

この章以降を読む前に、『Rational ClearCase ソフトウェア開発ガイド』で VOB、ビュー、構成仕様について十分理解してください。

プロジェクトの設定

ここでは、開発を始める前に行う必要のある計画と設定作業について説明します。

VOB の作成と入力

プロジェクトをほかのバージョン管理製品から ClearCase に移行する場合や、構成と変更管理の計画を初めて採用する場合は、プロジェクトの VOB に初期データ群 (ファイルとディレクトリ エlement群) を入力する必要があります。専任の ClearCase 管理者がいる場合、VOB の作成と保守は管理者の責任ですが、VOB へのデータのインポートは管理者の責任ではありません。

これらの内容に関しては、『Rational ClearCase 管理ガイド』で詳細に説明しています。

ブランチ作成ポリシーの計画

ClearCase は、並行開発を可能にするためにブランチを使用します。ブランチとは、Element のバージョンのシーケンスを指定する直列形のオブジェクトです。各 Element には main ブランチが 1 つあり、開発の主要な系列を表します。main ブランチには複数のサブブランチを作成できます。各サブブランチは個別の開発系列を表します。たとえば、新規開発作業のための main ブランチと、バグを解決するためのサブブランチのように、プロジェクトでは 2 つのブランチを並行して使用することができます。すべての Element の集約された main ブランチは、コードベースの main ブランチを構成します。

サブブランチは、複数のサブブランチを持つことができます。たとえば、あるサブブランチを製品の別のプラットフォームへの移植用に指定し、その移植用のブランチの下にバグ解決用のサブブランチを作成することができます。ClearCase では、複雑なブランチ階層を作成することができます。2 ページの図 1 は、複数レベルのブランチ階層を示します。このような環境の場合、プロジェクト マネージャーは、開発者が正しいブランチで作業を行うように注意する必要があります。そのためには、構成仕様内の規則を開発者に知らせ、開発者のビューが適切な一連のバージョンにアクセスするようにします。

「第 11 章 プロジェクト ビューの定義」では、構成仕様とブランチの詳細を説明しています。以下の説明に進む前に、ブランチ作成ポリシーの概要を理解すると役立ちます。

ブランチ作成ポリシーは、プロジェクトの開発目的に依存し、コードベースの進展を管理するメカニズムを提供します。ブランチ作成ポリシーには、ClearCase を使用する開発チームの数だけ多くのバリエーションがあります。しかしまた、ベストプラクティスを実現するための共通点もあります。ここでは、一般的なブランチタイプとその使用法について説明します。

タスク ブランチは、短期間で通常少数のファイルを使用し、タスクが完了すると親ブランチにマージされます。タスク ブランチは、一連の変更を特定のタスクに関連付ける監視トレイルを永続的に残すことで、責任の所在を明確にします。また、ビューや派生オブジェクトなど、必要なくなったときに削除できるタスクの成果物を簡単に識別できます。個々のタスクが同じファイルに変更を加える必要がない場合は、タスク ブランチをその親にマージすることは簡単です。

プライベート開発ブランチは、開発者グループがより包括的な一連の変更内容を共通のコードベースに作成する場合に役立ちます。必要に応じて **main** ブランチにブランチを作成することにより、開発者が単独で作業することができます。各開発者は、マージ前に **main** ブランチをプライベート ブランチにマージして、変更したファイルをチェックインする前に相違点を解決することで、**main** ブランチへのマージを簡単に行うことができます。

インテグレーション ブランチは、プライベート開発ブランチと **main** ブランチの間のバッファとして機能します。このブランチは、開発者が各自の作業を統合する代わりに、ある 1 人が代表でインテグレーションを行う場合に役立ちます。

ブランチ名

ブランチの作業内容を示すような命名規則を採用することをお勧めします。たとえば、**rel2.1_main** はリリース 2.1 のコードが最終的に存在するブランチ、**rel2.1_feature_abc** は ABC 機能に固有の変更を含むブランチ、**rel2.1_b12** はリリース 2.1 のコードの安定した第 2 のベースラインになります。必要に応じて、ブランチ名をより長くわかりやすくすることもできますが、長いブランチ名を付けるとバージョン ツリー表示が見づらくなります。

メモ: ラベル タイプと同じ名前前のブランチ タイプは作成しないでください。構成仕様がバージョン セレクタでラベルを使用する際に問題が発生するためです。たとえば、すべてのブランチ名は小文字に、すべてのラベルは大文字にします。

ブランチと ClearCase MultiSite

製品メモ: 現在、Rational ClearCase LT は ClearCase MultiSite をサポートしていません。

ブランチは、チームが作業に使用している VOB のレプリカが、ClearCase MultiSite 製品により別のサイトに作成されている場合には特に重要です。別のサイトの開発者は、エレメントの別のブランチで作業しています。このスキームによって衝突を回避します。たとえば、2 つのサイトの開発者が同じエレメントのバージョン **/main/17** を作成しているとします。この場合、ファイルのバージョンはマージできないか、マージしてはならないため、異なるサイトの開発者どうしてブランチを共有する必要があります。詳細については、197 ページの「特定ブランチの MultiSite のサイト間での共有」を参照してください。

共有ビューと標準構成仕様の作成

プロジェクト マネージャーとしては、開発者がファイルをチェックアウトするときのブランチの作成方法を決定する構成仕様を管理したい場合があります。このタスクを扱うには以下の方法があります。

- それぞれの開発者が使用する必要のある構成仕様テンプレートを作成します。開発者はテンプレートを各自の構成仕様に貼り付けるか、**ClearCase** のインクルード ファイル機能を使用して共通ソースから構成仕様を取得できます。
- 開発者が共有するビューを作成します。この方法は、通常、開発者にインテグレーションビューを提供し、プライベート ブランチで個別に行った作業をチェックインする際に便利です。

メモ: 単一の共有ビュー内で作業することは、システムのパフォーマンスが低下する場合がありますのでお勧めしません。

- Windows では、**ClearCase** のビュー プロファイルのメカニズムを使用して、プロジェクトチームが使用するビューを構成します。ビュー プロファイル ツールは、**ClearCase** を効果的に使用するための固有のモデルを利用します。このモデルに準拠するプロジェクト チームは、いくつかの領域で自動サポートを利用し、**ClearCase** のより高度な機能を簡単に利用できるようになります。ビュー プロファイルの詳細については、ヘルプを参照してください。

製品メモ: Rational ClearCase LT はビュー プロファイルをサポートしていません。

- すべてのチーム メンバーがビューを確実に同じ方法で構成するために、プロジェクト マネージャーは次のような標準の構成仕様ファイルを作成します。
 - /public/config_specs/ABC には、ABC チームの構成仕様が含まれます。
 - /public/config_specs/XYZ には、XYZ チームの構成仕様が含まれます。

これらの構成仕様ファイルを VOB の外部の標準ディレクトリに格納し、すべての開発者が同じバージョンを確実に取得できるようにします。

推奨されるビュー名

ブランチと同様の理由で、ビューの命名規則を採用します。ビューとそれが使用されるタスクを関連付けるのは、ブランチの場合より簡単です。**ClearCase** のビュー作成ツールは適当なビュー名を提示しますが、別の名前を使用することもできます。たとえば、すべてのビュー名 (ビュー タグ) に、所有者の名前とタスク (bill_V4.0_bugfix) や、ビューのホスト コンピュータの名前 (platinum_V4.0_int) を付けることができます。

開発ポリシーの実装

開発ポリシーを実施するために、ClearCase メタデータを作成して、バージョンの状態情報を保持することができます。プロジェクトの進捗状況を監視するために、このデータと、イベントレコードから取得した情報から、さまざまなレポートを生成することができます。

ラベルの使用法

ラベルは、バージョンに関連付けるユーザー定義の名前です。ラベルは、プロジェクト マネージャーとシステム統合担当者にとって強力なツールです。ラベルをエレメントのグループに適用することにより、開発ライフ サイクルのある時点での、一連のファイルとディレクトリのバージョンに関する相互関係を定義し、維持することができます。たとえば、ラベルを以下のバージョンに適用することができます。

- 統合とテスト後に安定していると考えられるすべてのバージョン。このベースライン ラベルを新規作業の基本として使用します。
- 部分的に安定しているか、有用な機能のサブセットを含むすべてのバージョン。このチェックポイント ラベルを途中のテストに使用したり、以降の変更によって退行したり不安定になった場合に開発をロールバックする地点として使用することができます。
- 特定の機能を実装するための変更を含むすべてのバージョン、またはパッチ リリースの一部であるすべてのバージョン。

属性、ハイパーリンク、トリガ、ロックの使用法

属性は、名前と値のペアであり、さまざまな視点からバージョンの状態情報を取り込むことができます。たとえば、CommentDensity という名前の属性をソース ファイルのそれぞれのバージョンに関連付けて、コード内のコメント状況を示すことができます。このような属性はそれぞれ、unacceptable、low、medium、high のいずれかの値を指定できます。

ハイパーリンクを使用すると、1 つ以上の VOB 内のエレメント間の関係を識別し、維持できます。この機能では、ソース ファイルを要求文書にリンクすることで、要求追跡などのプロセス管理要求を処理することができます。

トリガを使用すると、特定のプログラムや実行可能なスクリプトがコマンド実行の前または後に起動し、cleartool コマンドと ClearCase の操作の振る舞いを管理することができます。エレメントを変更する操作では、実質的にすべてトリガを起動することができます。特殊な環境変数によって、プロシージャを実装するスクリプトまたはプログラムが適切な情報を使用できるようにします。

操作前トリガは、指定した **ClearCase** コマンドが実行される前に起動します。**checkin** コマンドの操作前トリガにより、開発者に適切なコメントを追加するように促すことができます。操作後トリガは、コマンドが終了してコマンドの終了状態が利用できるようになった後に起動します。たとえば、**checkin** コマンドの操作後トリガにより、QA 部署に電子メールを送信し、特定の開発者が特定のエレメントに変更を加えたことを通知することができます。

また、トリガを使用して次のようなさまざまなプロセス管理機能を自動化することができます。

- オブジェクトが変更された場合の、オブジェクトへの属性の適用やラベルの関連付け
- **ClearCase** のイベント レコードには含まれない情報のログ取得
- 特定のオブジェクトが変更された場合の、ビルドとソース コード解析の開始

これらのメカニズムの詳細については、「第 12 章 プロジェクト開発ポリシーの実装」を参照してください。

エレメントまたはディレクトリをロックすると、(例外リストに指定されている以外の)すべての開発者はそれらを変更できなくなります。ロックは、一時的にアクセスを制限する際に役立ちます。たとえば、統合期間中に単一のオブジェクトである **main** ブランチ タイプをロックすると、インテグレーション チーム以外のユーザーはそれを変更できなくなります。

ロックの範囲は自由に設定できます。特定のエレメントの特定のブランチを新規開発ができないようにロックしたり、**VOB** 全体をロックして、**compressed_file** タイプの新規エレメントの作成や、**RLS_1.3** のバージョン ラベルの使用をできなくすることができます。

使用しなくなった名前、ビュー、**VOB** をロックすることもできます。この場合、ロックされたオブジェクトは不要としてタグ付けされ、ほとんどのコマンドで非表示になります。

グローバル タイプ

ClearCase のグローバル タイプ機能を使用すると、必要なブランチ、ラベル、属性、ハイパーリンク、エレメントを、プロジェクトで使用するすべての **VOB** で使用することができます。グローバル タイプの作成と使用方法に関する詳細については、『**Rational ClearCase** 管理ガイド』を参照してください。

レポートの作成

ClearCase は、エレメントが変更またはマージされるごとに、イベント レコードを作成して格納します。**ClearCase** のコマンドの大部分には、選択とフィルタのオプションがあり、このオプションを使用することでこれらのレコードに基づくレポートを作成することができます。レポートは、一連のオブジェクトまたは **VOB** 全体について 1 つのエレメントを対象とします。

「第 12 章 プロジェクト開発ポリシーの実装」には、イベント レコードとメタデータを使用して、プロジェクト ポリシーを実装するための詳細な説明があります。また、イベント レコードやその他のメタデータは、ClearCase が管理するアクティビティに関するレポート (たとえば、エレメント変更の完全な履歴) を作成する場合にも役立ちます。ClearCase には、さまざまなレポート生成ツールがあります。詳細については、『Rational ClearCase リファレンス ガイド』の `fmt_ccase` のリファレンス ページを参照してください。

変更の統合

プロジェクトの進行中、個々のエレメントの内容はブランチ作成操作によって分岐し、通常マージ操作で収束します。通常、プロジェクト マネージャーは、定期的に大部分のブランチを **main** ブランチに戻し、コード ベースが高度な整合性を確実に維持している状態で新規バージョンを安全にブランチ化できるように、各エレメントの最新バージョンを 1 つだけ保持するようにします。定期的にマージ作業を行わないと、現在のコード ベースにはすぐに多くの下位ブランチが作成され、それぞれが微妙に異なる内容を含むようになります。このような状況では、1 つのバージョンに加えた変更内容を手動でほかのバージョンに配信する必要があり、エラーが発生しやすい煩雑なプロセスになってしまいます。

プロジェクト マネージャーは、プロジェクトのマージ ポリシーを設定する必要があります。一般的に使用されるポリシーは、以下のとおりです。

- 開発者が変更内容を **main** ブランチにマージします。これは、開発者数か変更されたファイル数、またはその両方が少なく、開発者がマージのメカニズムを十分理解している場合にうまく作用します。開発者は、マージ ターゲットがマージされるバージョンの直前の祖先データでない場合に発生する可能性のある変更についても理解しておく必要があります。これは、数人の開発者が同じファイルに対して並行して作業している場合に発生します。
- 開発者が変更内容をインテグレーション ブランチにマージします。これは、個々の開発者のマージ作業と **main** ブランチ間のバッファになります。プロジェクト マネージャーまたはシステム統合担当者が、そのインテグレーション ブランチを **main** ブランチにマージします。
- 開発者が **main** ブランチまたはインテグレーション ブランチにマージする前に、**main** ブランチを各自の開発ブランチにマージする必要があります。この種のマージにより、マージに関連する不安定性の問題が開発者のプライベート ブランチでほかのチーム メンバーに影響を与える前に解決されるため、安定性が高まります。
- プロジェクト マネージャーは、開発者が **main** ブランチへのマージを行うためのスロットを指定します。これは、既に説明したいくつかのメカニズムのバリエーションになります。これにより、並行開発が行われている状況下で追加の管理レベルを提供します。

このほかにもさまざまなポリシーがあります。マージの詳細については、「第 14 章 変更の統合」を参照してください。

プロジェクト ビューの 定義

11

この章では、構成仕様の働きと構成仕様のサンプルについて説明します。サンプルは、プロジェクト開発作業、進捗状況の監視や調査の実行などの開発以外のタスク、プロジェクトのビルドの実行に役立ちます。また、Windows と UNIX のシステム間での構成仕様の共有方法についても説明します。

構成仕様の働き

プロジェクトにビューを作成する際には、1 つ以上の構成仕様を用意する必要があります。構成仕様を使用することで、開発者が参照できるバージョンや特定のビュー内で実行できる操作を管理して、プロジェクト作業全体を適切に制御することができます。ビューの範囲は、特定のブランチに限定したり、VOB 全体に広げることができます。また、選択したすべてのバージョンのチェックアウトを不許可にすることや、特定のブランチのみにチェックアウトを制限することもできます。

構成仕様には、ビューに表示されるバージョンを選択するために ClearCase が使用する一連の規則があります。チーム メンバーがビューを使用する場合、メンバーが参照するバージョンは、構成仕様内の規則の少なくとも 1 つと一致します。ClearCase は、各エレメントのバージョン ツリーから構成仕様の最初の規則に一致する最初のバージョンを検索します。最初の規則に一致するバージョンが存在しない場合、2 番目の規則に一致するバージョンを検索します。エレメントのバージョンが構成仕様のどの規則にも一致しない場合、ビューにはそのエレメントのバージョンは表示されません。

構成仕様内に規則を指定する順序によって、選択されるエレメントのバージョンが決定します。この章のいくつかの例では、さまざまな状況でのこの処理方法について検討します。構成仕様の準備の詳細については、`config_spec` のリファレンス ページを参照してください。

デフォルト構成仕様

この構成仕様は、動的な構成を定義します。この構成により、ソース ツリー全体のすべてのエレメントの **main** ブランチ上に開発者が加えた変更が選択されます。

- (1) element * CHECKEDOUT
- (2) element * /main/LATEST

これがデフォルト構成仕様で、新規に作成される各ビューはこの仕様に従って初期化されます。**mkview** コマンドまたはビュー作成ウィザード (Windows のみ) を使用してビューを作成すると、**default_config_spec** (ccase-home-dir にあります) というファイルの内容が、新規ビューの構成仕様になります。

この構成仕様によるビューは、チェックアウトされたバージョンを選択する個人用作業空間を提供します (規則 1)。デフォルトでは、ファイルをチェックアウトすると、**main** ブランチ上の最新バージョンからチェックアウトします (規則 2)。エレメントがチェックアウトされている間は、ほかのユーザーの作業に影響を与えることなくそれを変更することができます。新しいバージョンをチェックインすると、ビューで **/main/LATEST** バージョンを選択している開発者に対し、その変更が即座に使用可能になります。

ビューでは、その他のすべてのエレメント (つまり、チェックアウトしていないすべてのエレメント) も読み取り専用として選択されます。別のユーザーがこのようなエレメントの **main** ブランチ上で新しいバージョンをチェックインすると、新しい **LATEST** バージョンがすぐにこの動的ビューに表示されます。

デフォルトでは、スナップショット ビューにも以上のような 2 つのバージョン選択規則があります。さらに、スナップショット ビューの構成仕様にはロード規則があります。この規則は、スナップショット ビューにロードするエレメントまたはサブツリーを指定するものです。スナップショット ビューの作成の詳細については、『**Rational ClearCase ソフトウェア開発ガイド**』またはヘルプを参照してください。

製品メモ: Rational ClearCase LT はスナップショット ビューのみをサポートします。

標準構成規則

デフォルト構成仕様の 2 つの構成規則は、この章の例で頻繁に使用されます。**CHECKEDOUT** 規則を使用すると、既存のエレメントを変更することができます。この規則を省略したビュー内でエレメントをチェックアウトしようとする、チェックアウトはできませんが、**cleartool** が次の警告を出力します。

```
% cleartool checkout -nc cmd.c
```

```
cleartool: 警告: 名前を "cmd.c" から "cmd.c.keep" に変更できません :  
Read-only filesystem.
```

```
cleartool: エラー: バージョンがチェックアウトされましたが、データをビューの "cmd.c" に  
コピーできません: File exists.
```

この状態を修正してから、エレメントのチェックアウトを取り消し、再チェックアウトします。

```
cleartool: 警告 : チェックアウト済みバージョンのデータを "cmd.c.checkedout" に  
コピーしました。  
cleartool: 警告 : 'チェックアウト済みバージョンはビューで選択されていません。  
バージョン "/main/7" から "cmd.c" をチェックアウトしました。
```

この例では、構成仕様はエレメント **cmd.c** のバージョン 7 を読み取り専用で選択し続けます。このバージョンの読み書き可能なコピーである **cmd.c.checkedout** がビュープライベート記憶域内に作成されます (この作業方法は推奨されません)。

/main/LATEST 規則は、main ブランチの最新のバージョンを選択して、ビューに表示します。

/main/LATEST 規則は、ビュー内に新規エレメントを作成するためにも必要です。この規則を省略して新規エレメントを作成した場合、各自のビューではそのエレメントを「参照する」ことができません (エレメントを作成すると、main ブランチと空のバージョンの /main/0 が作成されます)。

標準構成規則の省略

ビューを使用してデータを変更しない場合にのみ、標準構成規則の 1 つまたは両方を省略します。たとえば、古いデータを参照するためだけに使用する 履歴ビューを構成することができます。同様に、コンパイルとテスト専用のビューや、ソースが正しくラベル付けされているかを検証するビューを構成することができます。

構成仕様のインクルード ファイル

ClearCase はインクルード ファイル機能をサポートしています。この機能を使用すると、すべてのチーム メンバーが簡単に同一の構成仕様を使用することができます。たとえば、この構成仕様の構成を **/public/c_specs/major.csp** というファイルに配置できます。この場合、各開発者は構成仕様に次の 1 行を追加する必要があります。

```
(1)          include /public/c_specs/major.csp
```

メモ: UNIX と Windows NT コンピュータの間で構成仕様を共有中に、両者の VOB タグが異なる場合、2 つのソースを用意するか、両方のプラットフォームからアクセスできる UNIX ディレクトリに構成仕様を格納する必要があります。

この構成仕様を変更する場合 (たとえば、ディレクトリなしのブランチ作成ポリシーを採用するなど)、変更する必要があるのは **major.csp** の内容だけです。次のコマンドを使用すると、変更した構成仕様でビューを再構成できます。

```
% cleartool setcs -current
```

構成仕様のサンプルでのプロジェクト環境

異なる種類の開発と管理タスクには、異なる構成仕様を使用できます。以下の3項では、プロジェクト開発、プロジェクト管理と調査、プロジェクトビルドのさまざまな面で役立つ構成仕様のサンプルを提示します。最初に、これらの構成仕様を使用する開発環境について説明します。

製品メモ: 次の例で、`/vobs/monet` などの複数コンポーネントの VOB タグを示す引数は、`/monet` などの単一コンポーネントの VOB タグのみを認識する UNIX 上の ClearCase LT には適用されません。

開発者は、`/vobs/monet` という VOB タグを持つ、次のような構造の VOB を使用しています。

<code>/vobs/monet</code>	(VOB タグ、VOB のマウントポイント)
<code>src/</code>	(C 言語ソース ファイル)
<code>include/</code>	(C 言語ヘッダー ファイル)
<code>lib/</code>	(プロジェクトのライブラリ)

この章の説明では、`lib` ディレクトリに次のサブ構造があるものとします。

<code>lib/</code>	
<code>libcalc.a</code>	(ライブラリのチェックイン状態のバージョン)
<code>libcmd.a</code>	(ライブラリのチェックイン状態のバージョン)
<code>libparse.a</code>	(ライブラリのチェックイン状態のバージョン)
<code>libpub.a</code>	(ライブラリのチェックイン状態のバージョン)
<code>libaux1.a</code>	(ライブラリのチェックイン状態のバージョン)
<code>libaux2.a</code>	(ライブラリのチェックイン状態のバージョン)
<code>libcalc/</code>	(calc ライブラリのソース)
<code>libcmd/</code>	(cmd ライブラリのソース)
<code>libparse/</code>	(parse ライブラリのソース)
<code>libpub/</code>	(pub ライブラリのソース)
<code>libaux1/</code>	(aux1 ライブラリのソース)
<code>libaux2/</code>	(aux2 ライブラリのソース)

ライブラリのソースは、**lib** のサブディレクトリにあります。ソース ディレクトリ内でビルドした後のライブラリは、**/vobs/monet/lib** に置くことができます。

UNIX: プロジェクトの実行可能プログラムのビルド スクリプトでは、リンク エディタ **ld(1)** が、標準的な場所 (たとえば、**/usr/local/lib**) の代わりにこのディレクトリ (ライブラリのステージング領域) 内のライブラリを使用するように指示することができます。

Windows: **LIB** 環境変数を設定するか、**makefile** を変更することで、標準的な場所の代わりにこのディレクトリ (ライブラリのステージング領域) 内のライブラリを使用することができます。

monet エレメントのバージョンには、次のラベルが割り当てられます。

バージョン ラベル

説明

R1.0

顧客への最初のリリース

R2_BL1

顧客への 2 回目のリリース前のベースライン 1

R2_BL2

顧客への 2 回目のリリース前のベースライン 2

R2.0

顧客への 2 回目のリリース

これらのバージョン ラベルは、各エレメントの **main** ブランチ上のバージョンに割り当てられています。ほとんどのプロジェクト開発作業は、**main** ブランチ上で行われます。特殊なタスクについては、サブブランチで開発が行われます。

サブブランチ

説明

major

アプリケーションのグラフィカル ユーザー インターフェイスや特定の計算アルゴリズムなどの主要な拡張作業用に使用されます。

r1_fix

リリース 1.0 のバグ修正用に使用されます。

Windows メモ: 構成仕様には、絶対 **VOB** パス名 (**VOB** タグで始まりドライブ文字やビュータグプレフィックスを含まない絶対パス名) を指定できます。この形式のパス名には、現在のドライブの割り当てやアクティブなビューとは関係なく、**VOB** エレメントを指定する必要があります。例を次に示します。

¥vob_gopher¥lib¥*

(絶対 **VOB** パス名。¥vob_gopher が **VOB** タグです。)

¥monet¥src¥*

(絶対 **VOB** パス名。¥monet が **VOB** タグです。)

Z:¥monet¥src¥*

(ドライブを特定したパス名。お勧めしません。)

M:¥myview¥vob_gopher¥lib¥*

(ビュー拡張パス名。お勧めしません。)

プロジェクト開発用のビュー

ここで説明する構成仕様は、さまざまなブランチ作成ポリシーを含むため、プロジェクト開発に役立ちます。

ブランチ上の新規開発用のビュー

次の構成仕様を使用すると、**major** という名前のブランチに作業を分離することができます。

- | | |
|-----|---|
| (1) | <code>element * CHECKEDOUT</code> |
| (2) | <code>element * .../major/LATEST</code> |
| (3) | <code>element * BASELINE_X -mkbranch major</code> |
| (4) | <code>element * /main/LATEST -mkbranch major</code> |

このスキームでは、すべてのチェックアウトが **major** という名前のブランチ上で発生します (規則 2)。

major ブランチは、整合性のあるベースラインを構成するバージョンに作成する必要があります。ベースラインは、メジャー リリース、マイナー リリース、アプリケーションの作業バージョンを生成する一連のバージョンのいずれかになります。この構成仕様では、ベースラインは **BASELINE_X** というバージョン ラベルによって定義されています。

時間規則を使用するバリエーション

ほかの開発者がチェックインしたバージョンが、自分のビューに表示されたとしても、自分の作業と互換性がない場合があります。このような場合、ソースの変更前の状態で作業を継続することができます。たとえば、次の構成仕様の規則 2 は、11 月 12 日午後 4:00 時点での **main** ブランチの最新バージョンを選択します。

- | | |
|-----|---|
| (1) | <code>element * CHECKEDOUT</code> |
| (2) | <code>element * /major/LATEST -time 12-Nov.16:00</code> |
| (3) | <code>element * BASELINE_X -mkbranch major</code> |
| (4) | <code>element * /main/LATEST -mkbranch major</code> |

この規則は、独自のチェックアウトには影響しません。

古い構成を変更するビュー

次の構成仕様を使用すると、バージョン ラベルで定義された構成を変更できます。

- | | |
|-----|--|
| (1) | <code>element * CHECKEDOUT</code> |
| (2) | <code>element * .../r1_fix/LATEST</code> |
| (3) | <code>element * R1.0 -mkbranch r1_fix</code> |

次の点に注意してください。

- エレメントはチェックアウト可能です (規則 1)。
- **checkout** コマンドは、最初に選択されたバージョンに **r1_fix** という名前のブランチを作成します (規則 3 の **-mkbranch**)。

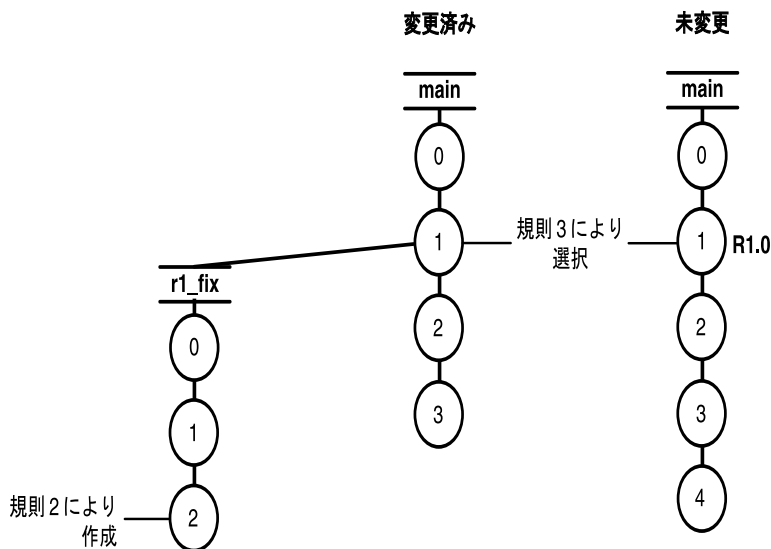
このスキームの重要な点は、**r1_fix** というブランチ名がすべての変更されるエレメントに使用されることです。管理の手間は、**mkbtype** コマンドを使用して単一のブランチ タイプ **r1_fix** を作成することだけです。

この構成仕様は効果的です。2 つの規則 (規則 2 と 3) が、すべてのエレメントの適切なバージョンを構成します。

- 変更されたエレメントの場合、このバージョンは **r1_fix** サブブランチ上で最新です (規則 2)。
- 変更されていないエレメントの場合、このバージョンは **R1.0** というラベルが付けられたバージョンです (規則 3)。

図 39 は、これらのエレメントを示しています。**r1_fix** ブランチは、**main** ブランチのサブブランチです。ただし、規則 2 はより一般的な場合も扱います。... ワイルドカードを使用すると、**r1_fix** ブランチを、任意のエレメントのバージョン ツリー内の任意の場所や、別のエレメントのバージョン ツリー内の別の場所で使用することができます。

図 39 古いバージョンの変更



/main/LATEST 規則の省略

170 ページの「古い構成を変更するビュー」にある構成仕様では、標準の /main/LATEST 規則を省略しています。この規則は、バージョン ラベル R1.0 が存在しない VOB を扱う作業には適していません。また、新規エLEMENTを作成する場合にも適していません。開発ポリシーが、以前の構成のメンテナンス作業で新規エLEMENTの作成を行わないものである場合には、/main/LATEST 規則を省略します。

変更プロセス中に新規エLEMENTの作成を許可するには、次の 4 つの構成規則を追加します。

- (1) element * CHECKEDOUT
- (2) element * /main/r1_fix/LATEST
- (3) element * R1.0 -mkbranch r1_fix
- (4) element * /main/LATEST -mkbranch r1_fix

新規エLEMENTが mkelem で作成されると、規則 4 の -mkbranch 節により、新規エLEMENTが r1_fix ブランチ上でチェックアウトされます (自動的に作成されます)。この規則は、r1_fix ブランチにすべての変更を集めるというスキームに準拠します。

時間規則を使用するバリエーション

次のベースライン構成は、時間規則により定義されます。

- (1) element * CHECKEDOUT
- (2) element * /main/r1_fix/LATEST
- (3) element * /main/LATEST -time 4-Sep:02:00 -mkbranch r1_fix

複数レベルのブランチ作成を実装するビュー

次の構成仕様により、整合性のある複数レベルのブランチ作成を実装し、実施します。

- (1) element * CHECKEDOUT
- (2) element * .../major/autumn/LATEST
- (3) element * .../major/LATEST -mkbranch autumn
- (4) element * BASELINE_X -mkbranch major
- (5) element * /main/LATEST -mkbranch major

この構成仕様で構成されるビューは、次の状況に適しています。

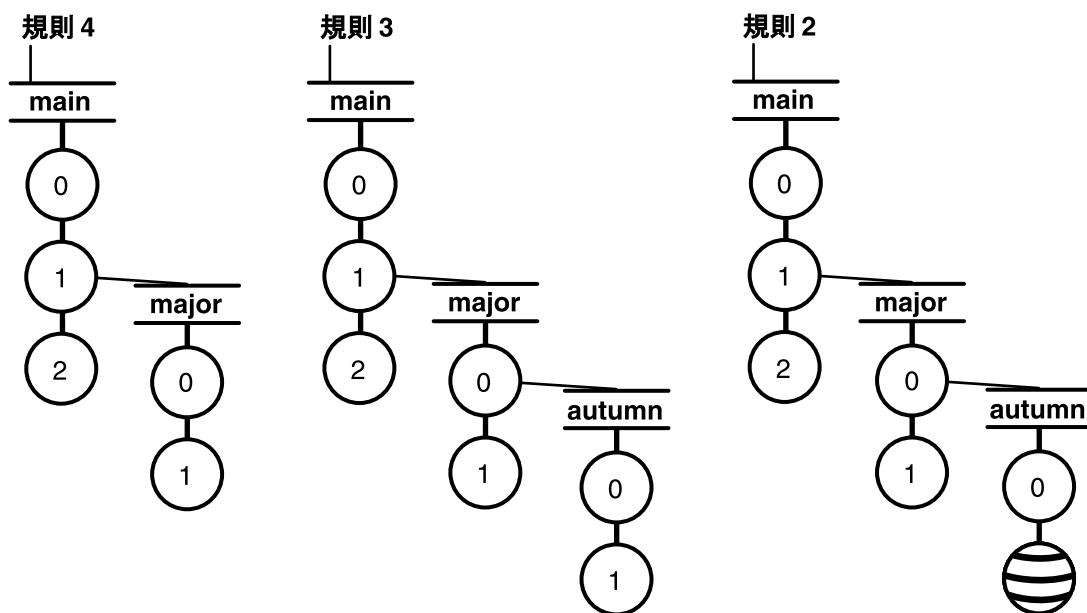
- BASELINE_X バージョン ラベルで指定されるベースラインからのすべての変更を major という名前のブランチ上で行う場合
- さらに、特殊なプロジェクトで作業しており、変更を major のサブブランチである autumn という名前のブランチ上で行う場合

図 40 は、ベースラインから変更していないエレメントをチェックアウトした場合に、このようなビューで何が発生するかを示しています。

- 1 エレメントをチェックアウトすると、規則 4 の mkbranch 節により、BASELINE_X バージョンに major ブランチが作成されます。
- 2 規則 3 の mkbranch 節により、¥main¥major¥0 に autumn ブランチが作成されます。
- 3 checkout 操作の終了時に、規則 2 が適用されます。最新バージョンの ¥main¥major¥autumn¥0 がチェックアウトされます。

複数レベルのブランチ作成の詳細については、config_spec と checkout のリファレンス ページを参照してください。

図 40 複数レベルの自動ブランチ作成



単一ディレクトリに変更を制限するビュー

次の構成仕様は、単一のディレクトリ /vobs/monet/src 内だけで変更を行う開発者に適しています。

- | | |
|-----|------------------------------------|
| (1) | element * CHECKEDOUT |
| (2) | element src/* /main/LATEST |
| (3) | element * /main/LATEST -nocheckout |

各エレメントの最新バージョンが選択されます (規則 2 と 3)。ただし、規則 3 により、指定したディレクトリにあるエレメントを除くすべてのエレメントのチェックアウトが禁止されます。

規則 2 は、任意の VOB 内にある **src** という名前のすべてのディレクトリのエレメントに一致します。`/vobs/monet/src/*` と指定すると、1 つの VOB のみを検索するように制限できます。

この構成仕様は、変更可能なソース ツリー領域を含むような規則を追加することで簡単に拡張することができます。

プロジェクト状態を監視するためのビュー

ここで示す構成仕様は、ビューを使用してプロジェクト状態を調査し、監視するために役立ちます。

バージョンを選択するための属性を使用するビュー

QA チームも **major** ブランチ上で作業を行っているとします。個々の開発者は、自分のモジュールを QA チェックに確実に合格させる責任があります。QA チームは、チェックに合格した最新のバージョンを使用して、アプリケーションのビルドとテストを行います。

QA チームは、次の構成仕様を使用したビュー内で作業することができます。

```
(1)          element -file src/* /main/major/{QAOK=="Yes"}
(2)          element * /main/LATEST
```

このスキームを機能させるには、**QAOK** という属性タイプを作成する必要があります。QA チェックに合格した新規バージョンが **major** ブランチ上にチェックインされる場合には、常に **QAOK** のインスタンスが **Yes** という値でそのバージョンに関連付けられます (これは手動または **ClearCase** のトリガにより行います)。

`/src` ディレクトリ内のエレメントが **major** ブランチ上で編集された場合、このビューでは QA チェックに合格したとマークされているブランチ上の最新バージョンが選択されます (規則 1)。合格としてマークされたバージョンがない場合、または **major** ブランチが作成されていない場合、**main** ブランチ上の最新のバージョンが使用されます (規則 2)。

メモ: この構成仕様の規則 1 では、エレメントに **major** ブランチがあっても、そのブランチ上に **QAOK** 属性を含むバージョンがない場合、検索は行われません。次のコマンドは、この属性を含まないブランチを特定します。

UNIX:

```
% cleartool find . -branch '{brtype(major) && ¥! attype_sub(QAOK)}' -print
```

¥ マークは、感嘆符 (!) が履歴の置換を示さないようにするため、C シェル内でのみ必要です。

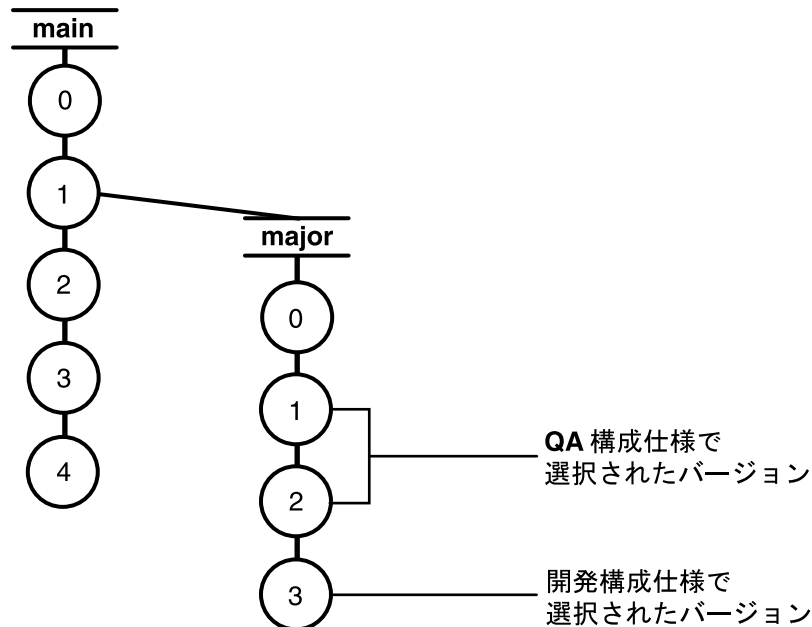
Windows:

```
cleartool find . -branch "{brtype(major) && ! attype_sub(QAOK)}" -print
```

attype_sub プリミティブは、エレメントのバージョンとブランチの属性に加えて、エレメント自体の属性を検索します。

このスキームを使用すると、QA チームは残りのグループの進捗状況を監視することができます。開発の構成仕様は常に **major** ブランチの最新バージョンを選択しますが、QA の構成仕様は中間バージョンを選択する場合があります (図 41)。

図 41 開発の構成仕様と QA の構成仕様



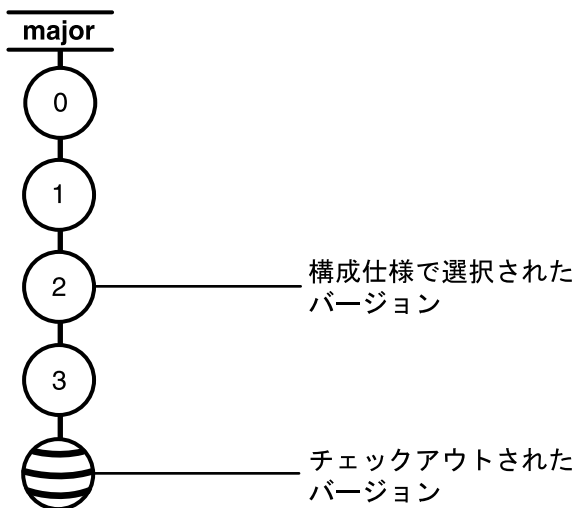
この構成を開発で使用する場合の注意点

次のように、この構成仕様に **CHECKEDOUT** 規則を追加して、QA 構成から開発構成に変更したい場合があります。

- (0) element * CHECKEDOUT
- (1) element -file src/* /main/major/{QAOK=="Yes"}
- (2) element * /main/LATEST

開発ビュー内でバージョン選択を管理するために、属性やその他のメタデータ、さらに (または代わりに) ブランチを使用することは有用に見えます。しかし、このようなスキームは複雑になります。この構成仕様で `.../src/cmd.c` というエレメントの `/main/major/2` というバージョンを選択するとします (図 42)。

図 42 エレメントのブランチのチェックアウト



このビュー内でチェックアウトを実行すると、バージョン `/main/major/3` はチェックアウトされますが、バージョン `/main/major/2` はチェックアウトされません。

cleartool: 警告: Version checked out is different from version previously selected by view.

バージョン `"/main/major/3"` から `"cmd.c"` をチェックアウトしました。

この振る舞いは、新規バージョンはブランチの最後にのみ作成できるという ClearCase の制約を反映しています。このような操作は可能ですが、ほかのチームメンバーを混乱させる可能性があります。また、エレメントをチェックアウトする開発者にとっても望ましい状況ではありません。

この問題は、構成仕様を変更して、属性が選択するバージョンに別のブランチ レベルを作成することで回避できます。新しい構成仕様は、次のとおりです。

- (0) element * CHECKEDOUT
- (0a) element * /main/major/temp/LATEST
- (1) element -file src/* /main/major/{QAOK=="Yes"} -mkbranch temp
- (2) element * /main/LATEST

1 人の開発者による変更を表示するビュー

次の構成仕様を使用すると、1 人の開発者により特定のマイルストーン以降に加られたすべての変更を簡単に調べることができます。

- (1) `element * '/main/{created_by(jackson) && created_since(25-Apr)}'`
- (2) `element * /main/LATEST -time 25-Apr`

メモ: 規則 1 は物理的に 1 行で記述する必要があります。

4 月 25 日という特定の日付は、マイルストーンとして使用されています。この構成は、その日付時点での開発のメイン ラインのスナップショットであり (規則 2)、その日以降に **jackson** というユーザーにより **main** ブランチ上に加えられたすべての変更によってオーバーレイされます (規則 1)。

`cleartool ls` コマンドの出力結果では、**jackson** のファイルがほかのユーザーのファイルと区別されます。それぞれのエントリには、選択したバージョンにどの構成規則を適用するかを示す注釈が含まれます。

これは、調査ビューであり、開発ビューではありません。選択された一連のファイルは整合性に欠ける場合があります。**jackson** による変更は、ほかのユーザーの変更に基づいている場合がありますが、ほかのユーザーの変更はこのビューからは除外されています。このため、この構成仕様では、標準の **CHECKEDOUT** 規則と **/main/LATEST** 規則を省略しています。

バージョン ラベルによって定義される履歴ビュー

次の構成仕様は、履歴の構成を定義します。

- (1) `element * R1.0 -nocheckout`

このビューは、常に **R1.0** というラベルが付けられた一連のバージョンを選択します。この場合、これらのすべてのバージョンは、エレメントの **main** ブランチ上にあります。**R1.0** のラベルタイプがエレメントあたり 1 つであり、ブランチあたり 1 つではない場合、この構成仕様はサブブランチ上の **R1.0** バージョンを選択します。詳細については、**mk1btype** のリファレンス ページを参照してください。

-nocheckout 修飾子は、任意のエレメントがこのビュー内でチェックアウトされないようにします (また、新規エレメントも作成できないようにします。親ディレクトリのエレメントをチェックアウトする必要があるからです)。このため、**CHECKEDOUT** 構成規則は必要ありません。

メモ: このビューで選択される一連のバージョンは、バージョン ラベルの移動や変更に伴って変化することがあります。たとえば、`mklabel -replace` コマンドを使用すると、**R1.0** がエレメントのバージョン 5 からバージョン 7 に移動するため、ビューに表示されるバージョンが変更されます。同様に、`rmlabel` を使用すると、指定したエレメントがビューに表示されなくなります (`cleartool` の `ls` コマンドは、[バージョンが選択されていません] という注釈を使用してこれらを一覧表示します)。ラベル タイプが `lock` コマンドによりロックされていると、構成は変更できません。

この構成を使用してリリース 1.0 を再ビルドし、すべてのソース エlement に正しくラベルが付けられているかを確認することができます。また、古いリリースを参照する場合に使用することもできます。

時間規則によって定義される履歴ビュー

次の構成仕様は、前の仕様とは少し異なる方法で、フリーズされた構成を定義します。

```
(1)          element * /main/LATEST -time 4-Sep.02:00 -nocheckout
```

この構成は、`main` ブランチ上の 9 月 4 日午前 2:00 時点での最新のバージョンを選択します。その後のチェックアウトとチェックインではこの基準を満たすバージョンの種類を変更することができません。`rmver` または `rmelem` などの削除コマンドによってのみ構成を変更することができます。`-nocheckout` 修飾子は、エレメントのチェックアウトも作成もできないようにします。

この構成を使用して、特定の時間に存在した一連のバージョンを参照することができます。このソース ベースに変更を加える必要がある場合、構成仕様を変更して、構成を「フリーズ解除」する必要があります。

プロジェクト ビルド用のビュー

ここでの構成仕様は、プロジェクトに必要なさまざまな種類のビルドを実行する際に役立ちます。

夜間ビルドの結果を使用するビュー

多くのプロジェクトでは、スクリプトを使用して、無人でソフトウェアのビルドを毎晩行っています。これらのビルドの成否は、アプリケーション上でチェックインされた任意の変更に影響します。階層化されたビルド環境では、下位レベルのソフトウェア (ライブラリ、ユーティリティ プログラムなど) の最新バージョンが提供される場合もあります。

毎晩、スクリプトで次のことを行おうとします。

- /vobs/monet/lib のさまざまなサブディレクトリ内でのライブラリのビルド
- ライブラリのステージング領域である /vobs/monet/lib 内での、派生オブジェクトのバージョンとしてのライブラリのチェックイン
- バージョンへの LAST_NIGHT というラベルの添付

夜間ビルドにより作成されるライブラリを使用する場合、次の構成仕様を使用することができます。

```
(1)          element * CHECKEDOUT
(2)          element lib/*.a LAST_NIGHT
(3)          element lib/*.a R2_BL2
(4)          element * /main/LATEST
```

ライブラリの LAST_NIGHT バージョンは、このようなバージョンが存在する場合には常に選択されます (規則 2)。夜間ビルドに失敗した場合は、前回の夜間ビルドのラベルが LAST_NIGHT のままなので、これが選択されます。LAST_NIGHT バージョンが存在しない場合 (ライブラリが現在開発中ではない場合)、代わりに R2_BL2 ラベルの付けられた安定バージョンが使用されます (規則 3)。

それぞれのライブラリに対して、ステージング領域内の最新バージョンではなく、LAST_NIGHT ラベルが付いたバージョンを選択することにより、開発者は、この構成仕様を使用する開発者に影響を与えることなく、新規バージョンを次の日にステージングすることができます。

プロジェクト ライブラリのバージョンを選択するバリエーション

これまでに説明したスキームでは、バージョン ラベルを使用してライブラリの特定のバージョンを選択します。次の構成仕様を使用すると、一部のライブラリでは LAST_NIGHT バージョンを選択して、別のライブラリでは R2_BL2 バージョンや最新のバージョンを選択するなど、より柔軟な処理を行うことができます。

```
(1)          element * CHECKEDOUT
(2a)         element lib/libcmd.a LAST_NIGHT
(2b)         element lib/libparse.a LAST_NIGHT
(3a)         element lib/libcalc.a R2_BL2
(3b)         element lib/*.a /main/LATEST
(4)          element * /main/LATEST
```

(規則 3b はここでは必要ありません。規則 4 がその他のすべてのライブラリを処理するからです。内容を明確にするために示しています。)

その他のメタデータもライブラリのバージョンを選択するために使用できます。たとえば、**lib_selector** 属性は、**experimental**、**stable**、**released** などの値を取ることができます。次の構成仕様では、ライブラリのバージョンの組み合わせを検索することができます。

```
(1)      element * CHECKEDOUT
(2)      element lib/libcmd.a {lib_selector=="experimental"}
(3)      element lib/libcalc.a {lib_selector=="experimental"}
(4)      element lib/libparse.a {lib_selector=="stable"}
(5)      element lib/*.a {lib_selector=="released"}
(6)      element * /main/LATEST
```

アプリケーション サブシステムのバージョンを選択するビュー

次の構成仕様は、アプリケーション サブシステムの特定バージョンを選択します。

```
(1)      element * CHECKEDOUT
(2)      element /vobs/monet/lib/... R2_BL1
(3)      element /vobs/monet/include/... R2_BL2
(4)      element /vobs/monet/src/... /main/LATEST
(5)      element * /main/LATEST
```

この場合、開発者は **main** ブランチ上のアプリケーション ソース ファイルに変更を加えます (規則 4)。アプリケーションのビルドには、ベースライン 1 のビルドに使用した **/lib** ディレクトリ内のライブラリと、ベースライン 2 のビルドに使用した **/include** ディレクトリ内のヘッダー ファイルを使用します。

特定のプログラムをビルドしたバージョンを選択するビュー

次の構成仕様は、特定のプログラムの再ビルド、またはそのソースの検証に必要なファイルだけを選択するビューを定義します。

```
(1)      element * -config /vobs/monet/src/monet
```

ClearCase の **ls** コマンドを実行すると、**monet** のビルドに含まれていなかったエレメントは、すべて [バージョンが選択されていません] という注釈付きで表示されます。

この構成仕様では、特定の派生オブジェクトの構成レコード内 (およびビルドが依存するオブジェクトの構成レコード内) に一覧表示されるバージョンが選択されます。このオブジェクトは、現在のビューでビルドされた派生オブジェクト、別のビューでビルドされた派生オブジェクト、派生オブジェクトのバージョンのいずれかです。

この構成仕様では、**monet** は現在のビューでの派生オブジェクトです。派生オブジェクトは、派生オブジェクトの ID を含む拡張パス名を使用して別のビューで参照することができます。

```
(1) element * -config /vobs/monet/src/monet@@09-Feb.13:56.812
```

ただし、一般的にこの種の構成仕様は、派生オブジェクトのバージョンとしてチェックインされた派生オブジェクトからのビューを構成するために使用します。

makefile の構成

デフォルトでは、派生オブジェクトの構成レコードには、そのビルドで使用した **makefile** のバージョンは表示されません。その代わり、構成レコードにはビルドスクリプト自体のコピーが含まれています (1 つのターゲットのビルドスクリプトを更新して **makefile** の新規バージョンを作成すると、その **makefile** でビルドされたその他すべての派生オブジェクトの構成レコードが古くならず済むからです)。

ただし、**monet** のプログラムがこのビューで **clearmake** (または、UNIX の **make** か Windows の **omake**) を使用して再ビルドされる場合、**makefile** のバージョンを何らかの方法で選択する必要があります。ターゲットの依存関係リスト内に特殊な **clearmake** マクロ呼び出し **\$(MAKEFILE)** を含めることにより、**clearmake** は構成レコード内の **makefile** のバージョンを記録することができます。

Windows:

```
monet.exe: $(MAKEFILE) monet.obj ...
    link -out:monet.exe monet.obj ...
```

UNIX:

```
monet: $(MAKEFILE) monet.o ...
    cc -o monet ...
```

clearmake は、常に構成レコード内の明示的依存関係のバージョンを記録します。

または、**makefile** をソース レベルで構成することができます。ビルド時にバージョン ラベルを **makefile** に関連付けて、177 ページの「バージョン ラベルによって定義される履歴ビュー」または 170 ページの「古い構成を変更するビュー」のような構成仕様を使用して、ビルド用のビューを構成することができます。また、**.DEPENDENCY_IGNORED_FOR_REUSE** という特殊なターゲットを使用することもできます。詳細については、『**Rational ClearCase ソフトウェア ビルド ガイド**』を参照してください。

プログラム内のバグの修正

monet プログラムでバグが発見されても、特定のプログラムを再ビルドするために必要なファイルのみを選択しているビュー内で再ビルドが行われるので、ビューをビルド構成から開発構成に変換するのは簡単です。通常、古いソースを変更する場合は、次のポリシーに従います。

- 変更する各バージョンに対してブランチを作成します。
- すべてのエレメントで同一のブランチ名を使用 (つまり、同一のブランチ タイプのインスタンスを作成) します。

修正用のブランチ タイプが `r1_fix` の場合、この変更された構成仕様は修正のためのビューを再構成します。

```
(1)          element * CHECKEDOUT
(2)          element * .../r1_fix/LATEST
(3)          element * -config /vobs/monet/src/monet -mkbranch r1_fix
(4)          element * /main/LATEST -mkbranch r1_fix
```

一連のプログラムをビルドしたバージョンの選択

特定のプログラムを再ビルドするために必要なファイルのみを選択する構成仕様を拡張して、1 つのプログラムではなく、複数のプログラムをビルドするためのソースを含むビューを簡単に構成できます。

```
(1)          element * -config /proj/monet/src/monet
(2)          element * -config /proj/monet/src/xmonet
(3)          element * -config /proj/monet/src/monet_conf
```

ただし、このような構成ではバージョンが競合する可能性があります。たとえば、`monet` と `xmonet` のビルドで、バージョンの異なる `params.h` ファイルが使用されていることもあります。この場合、その構成規則が最初に記述されているため、`monet` で使用されるバージョンが構成されます。同様に、1 つの `-config` 規則を使用している場合にも競合が発生することがあります。あるターゲットを実際にビルドし、ほかのターゲットの派生オブジェクトのバージョンを使用して、指定した派生オブジェクトを作成した場合、いくつかのソース ファイルの複数のバージョンが必要になる場合があります。

この構成仕様は、181 ページの「プログラム内のバグの修正」で説明したように変更して、ビルド構成から開発構成に変更することができます。

UNIX と Windows 間での構成仕様の共有

基本的に、UNIX と Windows システム間で構成仕様を共有することができます。つまり、両方のシステム上のユーザーは、それぞれのプラットフォーム上に格納ディレクトリを持つビューを使用して、同じ一連の構成仕様の設定と編集を行うことができます。

異なるプラットフォーム間での構成仕様の共有は避けることをお勧めします。可能であれば、各プラットフォームで個別の構成仕様を作成します。構成仕様を共有する必要がある場合は、次の条件に準拠してください。

- パス名にはスラッシュ (/) を使用し、円記号 (¥) を使用しないようにします。
- 可能な限り相対パス名を使用して、フルパス名は使用しないようにします。また、パス名に VOB タグを使用しないようにします。UNIX と Windows の VOB タグが共に同じパス名の単一コンポーネントを使用しており、最初のスラッシュ文字のみが異なる (¥src と /src など) 場合、この制限は無視できます。
- 構成仕様の編集と設定は、常に UNIX 上で行います。

以下の項では、これらの条件を詳細に説明します。

パス名区切り文字

Windows と UNIX コンピュータで共有する構成仕様を作成する場合、パス名区切り文字にはスラッシュ (/) を使用し、円記号 (¥) は使用しません。UNIX 上の ClearCase が認識できるのはスラッシュのみです (cleartool はパス名のスラッシュと円記号の両方を認識しますが、clearmake は両者を識別しません。詳細については、『Rational ClearCase ソフトウェア ビルドガイド』を参照してください)。

構成仕様のエレメント規則内のパス名

Windows と UNIX のネットワーク リージョンは、同じ VOB を登録する際に異なる VOB タグを使用することがあります。Windows コンピュータでは ¥proj1 などの単一コンポーネント VOB タグ名だけが許可されますが、UNIX では /vobs/src などの複数コンポーネント VOB タグが一般的です。

リージョン間で VOB タグが異なる場合、(VOB タグのある) フルパス名を使用する任意の構成仕様エレメント規則は、構成仕様が (cleartool edcs と setcs コマンドで) コンパイルされる際に解決されますが、これは適切なネットワーク リージョン内のコンピュータを使用した場合のみです。このことは、共有構成仕様に関する一般的な制約を示しています。特定の構成仕様は、エレメント規則内のフルパス名を認識できるオペレーティング システム上でのみコンパイルする必要があります。つまり、フルパス名を含む構成仕様は、VOB タグが一致していなくてもネットワーク リージョン間で共有可能ですが、コンパイルは適切な場所で行う必要があります。

次のいずれかに該当する場合、この制約は当てはまりません (184 ページの「例」を参照)。

- 構成仕様のエレメント規則が、VOB タグを含まない相対パス名のみを使用している場合
- 共有されている VOB が、同一の単一コンポーネントの VOB タグで Windows と UNIX の両方のネットワーク リージョンに登録されている場合 (VOB タグの ¥r3vob と /r3vob は、最初のスラッシュと円記号が異なるだけなので、同一のものとして扱われます)

構成仕様のコンパイル

使用中の構成仕様は、テキストファイルとコンパイル済みのフォーマットで存在します。構成仕様のコンパイルされた形式は移植可能です。ただし、エレメント規則内で使用する VOB のフルパス名は、コンパイル時に解決できなければならないという制限があります。構成仕様は、編集または設定時に (`cleartool edcs` または `cleartool setcs` コマンドや、ClearCase GUI を使用して) コンパイルします。ユーザーが別のオペレーティングシステム上で (`edcs` または `setcs` コマンドを実行するか、GUI でこれらのコマンドを実行することにより) 構成仕様を再コンパイルした場合、その構成仕様はそのビューを使用している任意のコンピュータで使用できなくなります。これを解決するには、元のオペレーティングシステム上でその構成仕様を再コンパイルします。

例

次の構成仕様のエレメント規則には問題があります。

```
element %vob_p2%abc_proj_src%*          %main%rel2%LATEST
```

Windows ネットワーク リージョンでは `%vob_p2`、UNIX ネットワーク リージョンでは `/vobs/vob_p2` という VOB タグを使用して VOB が登録されている場合、Windows コンピュータのみがこの構成仕様をコンパイルできます。

この問題を解決するには、次のいずれかを行います。

- 次のように、VOB タグを含まない相対パス名を使用します。

```
element ...%abc_proj_src%*          %main%rel2%LATEST
```

- UNIX 上で VOB タグを変更し、単一コンポーネント `/vob_p2` にします。

この章では、ClearCase を使用して一般的な開発ポリシーの実装と実施を行うための簡単なシナリオをいくつか紹介します。これらのシナリオでは、次の機能とメタデータをさまざまに組み合わせて使用します。

- 属性
- ラベル
- ブランチ
- トリガ
- 構成仕様
- ロック
- ハイパーリンク

198 ページの「UNIX と Windows 間でのトリガの共有」では、UNIX と Windows コンピュータ上で使用するトリガの定義方法について説明しています。

変更についての十分な説明

VOB を変更するそれぞれの ClearCase コマンドは、1 つ以上のイベント レコードを作成します。このようなコマンド (たとえば、**checkin**) の多くは、コメントの入力を促します。イベント レコードには、変更内容に対するユーザー名、日付、コメント、ホスト、説明が含まれます。

開発者が空のコメントを入力してシステムの品質を低下させることがないように、**checkin** コマンドを監視する操作前トリガを作成できます。トリガ アクション スクリプトは、環境変数内に渡されたユーザーのコメントを解析して、容認できないコメントを許可しないようにします。

Windows メモ: ClearCase では、トリガを起動する際に、トリガ操作の成功または失敗を基にして処理を続けますが、それはトリガ スクリプトの終了コードによって判定されます。**.bat** ファイルは、その直前のコマンドの終了コードを返します。操作前トリガは、ClearCase の実行を失敗させる唯一の種類のトリガです。

UNIX のトリガ定義:

```
% cleartool mkttrtype -element -all -preop checkin -c "must enter descriptive comment" ¥  
-exec /public/scripts/comment_policy.sh CommentPolicy
```

Windows のトリガ定義:

```
cleartool mkttrtype -element -all -preop checkin ^  
-c "must enter descriptive comment" -exec ¥¥neon¥scripts¥comm_pol.bat CommentPolicy
```

UNIX のトリガアクションスクリプト:

```
#!/bin/sh  
#  
#      comment_policy  
#  
ACCEPT=0  
REJECT=1  
WORDCOUNT= `echo $CLEARCASE_COMMENT | wc -w`  
  
if [ $WORDCOUNT -ge 10 ] ; then  
    exit $ACCEPT  
else  
    exit $REJECT  
fi
```

Windows のトリガアクションスクリプト:

```
@echo off  
rem comm_pol.bat  
rem  
rem NULL のチェック  
rem  
if "%CLEARCASE_COMMENT%"==" " copy > NUL:
```

すべてのソース ファイルへのプロGRESS インジケータの添付

個々のファイルの進捗状況の監視や、特定の状態にあるファイルとそのファイル数を調べる必要がある場合があります。属性を使用してこの情報を保持したり、トリガを使用して情報を収集することができます。

この場合、一連の指定した値を受け付ける **Status** という文字列値の属性タイプを作成します。

UNIX の属性の定義:

```
% cleartool mkatttype -c "standard file levels" ¥  
-enum ' "inactive","under_devt","QA_approved" ' Status  
属性タイプ "Status" を作成しました。
```

Windows の属性の定義:

```
cleartool mkatttype -c "standard file levels" ^  
-enum "¥"inactive¥","¥"under_devt¥","¥"QA_approved¥"" Status  
属性タイプ "Status" を作成しました。
```

開発者は、**Status** 属性をエレメントの多くの異なるバージョンに適用します。ブランチ上の初期バージョンでは、この値は **inactive** と **under_devt** になり、以後のバージョンでは、この値は **QA_approved** になります。同じ値をいくつかのバージョンで使用したり、初期バージョンから後のバージョンに移動することができます。

この属性をすべてのソース ファイルのバージョンに確実に適用するために、**Status** 属性を持たないバージョンを開発者がチェックインできないようにするアクション スクリプトを含む **CheckStatus** トリガを作成することができます。

UNIX のトリガ定義:

```
% cleartool mkttrtype -element -all -preop checkin ¥  
-c "all versions must have Status attribute" ¥  
-exec 'Perl /public/scripts/check_status.pl' CheckStatus
```

Windows のトリガ定義:

```
cleartool mkttrtype -element -all -preop checkin ^  
-c "all versions must have Status attribute" ^  
-exec "ccperl ¥¥neon¥scripts¥check_status.pl" CheckStatus
```

トリガアクション スクリプト:

```
$pname = $ENV{'CLEARCASE_PN'};  
$val = "";  
$val = 'cleartool describe -short -aattr Status $pname '  
  
if ($val eq "") {  
    exit (1);  
} else {  
    exit (0);  
}
```

主要構成に使用されているすべてのバージョンへのラベルの関連付け

特定のベースラインまたはリリースにあるエレメントを特定し、そのバージョンを識別するために、それらのバージョンにラベルを関連付けることができます。たとえば、リリース 2 のビルドとテストを行った後に、**mklibtype** コマンドを使用して、**REL2** というラベル タイプを作成します。次に **mklabel** コマンドを使用して、**REL2** を適切なソース バージョンのバージョンラベルとして関連付けます。

適切なバージョンを検討します。たとえば、リリース 2 が特定のビュー内で下位からビルドされた場合、このビューで選択したバージョンにラベルを関連付けることができます。

```
% cleartool mklibtype -c "Release 2.0 sources" REL2
```

```
% cleartool mklabel -recurse REL2 top-level-directory
```

または、リリースの派生オブジェクトの構成レコードを使用して、ラベル関連付けのプロセスを管理することもできます。

```
% clearmake vega
```

... QA がビルドを承認した後:

```
% cleartool mklabel -config vega@@@17-Jun.18:05 REL2
```

構成レコードを使用してバージョン ラベルを関連付けると、開発者がリリース ビルド以降に新規バージョンを作成した場合でも、正確かつ完全にラベルを関連付けることができます。これにより、品質保証中やリリース準備中にも開発作業を継続することができます。

バージョン ラベル **REL2** が再度使用されないようにするには、次のようにラベル タイプをロックします。

```
% cleartool lock -nusers vobadm ltype:REL2
```

このオブジェクトは、**-nusers** オプションで指定したユーザー (この場合は **vobadm**) 以外のすべてのユーザーに対してロックされます。

リリース バグの作業のブランチ上への隔離

リリースしたシステムで発見したバグをバグ修正用ブランチで修正する際に、このリリースの正確なバージョン構成を使用して作業を開始したい場合があります。

このポリシーは、**ClearCase** のベースラインと変更のモデルを反映しています。まず、ラベル (たとえば、**REL2**) をリリース構成に関連付ける必要があります。次に、任意のチーム メンバーは、次の構成仕様を使用したビューを作成してポリシーを実装します。

```
element * CHECKEDOUT
element * .../rel2_bugfix/LATEST
element * REL2 -mbranch rel2_bugfix
```

この構成により 1 つ以上のビュー内ですべてのバグが修正される場合、変更は `rel2_bugfix` タイプのブランチ上に隔離されます。`-mkbranch` オプションは、エレメントがチェックアウトされる際に、必要に応じてこのようなブランチを作成します。

この構成仕様は、このタイプのブランチが存在する `rel2_bugfix` ブランチからバージョンを選択します。このようなブランチは、`REL2` バージョンがチェックアウトされる場合には常に作成されます。

ほかの開発者による作業の混乱の回避

生産性を高めるために、開発者は、変更を参照するタイミングと変更の内容を管理する必要があります。この目的のための適切なメカニズムは、ビューです。開発者は、既存の構成仕様を変更するか、構成仕様を新規に作成することで、参照する変更と参照しない変更を厳密に指定することができます。

このポリシーを実装するには、チェックインした変更をフィルタ処理する構成仕様規則の記述とデリバリーを、開発者に行わせる必要があります。構成仕様の例は、以下のとおりです。

- 自分の作業と、リリース 2 のビルドに含まれるすべてのバージョンを選択する場合

```
element * CHECKEDOUT
element * REL2
```

- 自分の作業と、日曜日の夜の時点での最新バージョンを選択する場合

```
element * CHECKEDOUT
element * /main/LATEST -time Sunday.18:00
```

- 自分の作業、`graphics` ディレクトリ内に作成された新規バージョン、昨夜ビルドしたバージョンのすべてを選択する場合

```
element * CHECKEDOUT
element graphics/* /main/LATEST
element * -config myprog@@12-Jul.00:30
```

- 自分の作業、自分 (`jones`) または `Mary` が今日チェックインしたバージョン、最新の品質保証バージョンのすべてを選択する場合

```
element * CHECKEDOUT
element * '/main/{ created_since(06:00) && ( created_by(jones) ||
created_by(mary) ) }'
element * /main/{QAed=="TRUE"}
```

- 構成仕様インクルード機能を使用して、次のように開発者用の一連の標準構成規則を設定し、開発者独自の構成仕様を追加することもできます。

```
element * CHECKEDOUT
element msg.c /main/18
include /usr/cspecs/rules_for_rel2_maintenance
```

必要に応じたプロジェクト データへのアクセスの拒否

プロジェクト チーム メンバーの全員または大部分からのアクセスを拒否する場合があります。たとえば、通知するまではパブリック ヘッダー ファイルを変更させたくない場合です。lock コマンドは、次のような一時的なポリシーを実施するために設計されています。

- あるディレクトリ内のすべてのヘッダー ファイルをロックします。
% cleartool lock src/pub/*.h
- Mary と Fred 以外のすべてのユーザーに対して、ヘッダー ファイルをロックします。
% cleartool lock -nusers mary,fred src/pub/*.h
- VOB 内のすべてのヘッダー ファイルをロックします。
% cleartool lock eltype:c_header
- VOB 全体をロックします。
% cleartool lock vob:/vobs/myproj

関連する変更のチーム メンバーへの通知

チーム メンバーが、自分の作業に影響する変更を追跡しやすくするために、操作後トリガを使用して、さまざまなイベントの通知を送信することができます。たとえば、開発者が GUI を変更した場合、電子メールのメッセージをマニュアル制作グループに送り、これらの変更を確実にマニュアルに反映することができます。

このポリシーを実施するには、メールを送信するトリガ タイプを作成して、それを関連するエレメントに関連付けます。

UNIX のトリガ定義:

```
% cleartool mktrtype -nc -element -postop checkin ¥
    -exec /public/scripts/informwriters.sh InformWriters
```

トリガ タイプ "InformWriters" を作成しました。

UNIX のトリガ アクション スクリプト:

```
#!/bin/sh
#
# 初期化
tmp=/tmp/checkin_mail

# チェックインを記述するメール メッセージの作成

cat > $tmp <<EOF
Subject: Checkin $CLEARCASE_PNAME by $CLEARCASE_USER
$CLEARCASE_XPNAME
Checked in by $CLEARCASE_USER.

Comments:
$CLEARCASE_COMMENT
EOF

# メッセージの送信

mail docgrp <$tmp

# クリーンアップ

#rm -f $tmp
```

Windows のトリガ定義:

```
cleartool mkttrtype -nc -element -postop checkin ^
-exec "ccperl ¥¥neon¥scripts¥informwriters.pl" InformWriters
トリガ タイプ "InformWriters" を作成しました。
```

Windows のトリガ アクション スクリプト:

```
use Net::SMTP;

my $smtp = new Net::SMTP 'neon.purpledock.com';

$smtp->mail('ClearCase Admin');
$smtp->to('ClearCase Admin');
$smtp->to('docgrp');

$smtp->data();
$smtp->datasend("From: ClearCase Admin¥n");
$smtp->datasend("To: docgrp¥n");
$smtp->datasend("Subject: checkin¥n");
$smtp->datasend("¥n");
```

```
# pathname/user/comment の変数の作成

$ver = $ENV{'CLEARCASE_XPN'};
$user = $ENV{'CLEARCASE_USER'};
$comment = $ENV{'CLEARCASE_COMMENT'};

$var = "Version: $ver¥nUser: $user¥nComment: $comment¥n";

$smtp->datasend($var);
$smtp->dataend();
$smtp->quit;
```

トリガを既存のエレメントに関連付けるには

- 1 トリガを、GUI ソース ツリー内のすべての既存ディレクトリ エレメントの継承リストに配置します。

```
% cleartool find /vobs/gui_src -type d ¥
-exec 'cleartool mktrigger -nattach InformWriters $CLEARCASE_PN'
```
- 2 トリガを、GUI ソース ツリー内のすべての既存ファイル エレメントの関連付けリストに配置します。

```
% cleartool find /vobs/gui_src -type f ¥
-exec 'cleartool mktrigger InformWriters $CLEARCASE_PN'
```

すべてのソース ファイルのプロジェクト標準への準拠

開発者がコーディング ガイドラインなどの標準を確実に遵守しているかどうかを確認するために、ソース ファイルを評価できます。操作前トリガを作成して、ユーザー定義プログラムの実行や、トリガを起動したコマンドのキャンセルができます。

たとえば、品質基準を満たしていない C 言語ファイルのチェックインを拒否したい場合があります。**c_source** というエレメント タイプを C 言語ファイル (*.c) に定義するとします。

UNIX のトリガ定義:

```
% cleartool mktrtype -element -all -eltype c_source ¥
-preop checkin -exec '/public/scripts/apply_metrics.sh $CLEARCASE_PN' ApplyMetrics
```

Windows のトリガ定義:

```
cleartool mktrtype -element -all -eltype c_source ^
-preop checkin -exec "¥¥neon¥scripts¥appl_met.bat %CLEARCASE_PN%" ApplyMetrics
```

このトリガ タイプ **ApplyMetrics** は、すべてのエレメントに対して適用されます。これは、**c_source** というタイプの任意のエレメントがチェックインされると起動します (**c_source** エレメントが新規作成された場合は監視されます)。開発者が、**apply_metrics.sh** または **appl_met.bat** テストに失敗した **c_source** ファイルをチェックインしようとする、チェックインは失敗します。

メモ: **apply_metrics.sh** スクリプトと **appl_met.bat** ファイルは、その環境から **CLEARCASE_PN** の値を読み取ることができます。ファイル名の引数を受け取るようにすると、スクリプトまたはバッチ ファイルをトリガ アクションとして起動し、開発者が手動でそれを使用できるため、柔軟性が高まります。

変更と変更順序の関連付け

設計変更指示 (Engineering Change Order、以下 ECO) に応じて行った作業を追跡するために、属性とトリガを使用することができます。たとえば、バージョンを ECO と関連付けるには、次のように ECO を整数値の属性タイプとして定義します。

```
cleartool mkatttype -c "bug number associated with change" -vtype integer ECO
```

属性タイプ "ECO" を作成しました。

次に、すべてのエレメントのトリガ タイプである **EcoTrigger** を定義します。このトリガ タイプは、新規バージョンが作成されると必ず起動して ECO 属性を関連付けるスクリプトを実行します。

トリガ定義:

```
cleartool mktrtype -element -all -postop checkin -c "associate change with bug number" ¥  
-execunix 'Perl /public/scripts/eco.pl' -execwin 'ccperl ¥¥neon¥scripts¥eco.pl' EcoTrigger  
トリガ タイプ "EcoTrigger" を作成しました。
```

トリガ アクション スクリプト:

```
$pname = $ENV{' CLEARCASE_XPN' };  
  
print "Enter the bug number associated with this checkin: ";  
$bugnum = <STDIN>;  
chomp ($bugnum);  
$command = "cleartool mkattr ECO $bugnum $pname";  
  
@returnvalue = '$command';  
$rval = join " ",@returnvalue;  
print "$rval";  
  
exit(0);
```

新規バージョンが作成されると、たとえば次のように属性がバージョンに関連付けられます。

```
cleartool checkin -c "fixes for 4.0" src.c
Enter the bug number associated with this checkin: 2347
属性 "ECO" を "/vobs/dev/src.c@@/main/2" 上に作成しました。
"src.c" のバージョン "/main/2" をチェックインしました。

cleartool describe src.c@@/main/2
バージョン "src.c@@/main/2"
...
属性 :
    ECO = 2347
```

プロジェクト要求とソース ファイルの関連付け

ハイパーリンクを使用すると、要求追跡を実装できます。ハイパーリンクは VOB オブジェクトどうしを関連付けます。この関連付けは、(ブランチやエレメント レベルではなく)バージョンレベルで行う必要があります。ソース コード モジュールの各バージョンを、関連する設計文書の特定期間バージョンと関連付ける必要があります。

たとえば、プロジェクト マネージャーが **DesignDoc** という名前のハイパーリンク タイプを作成し、ソース コードと設計文書を関連付けるには次のようにします。

```
cleartool mkhlttype -c "associate code with design docs" ¥
DesignDoc@/vobs/dev DesignDoc@/vobs/design
ハイパーリンク タイプ "DesignDoc" を作成しました。
ハイパーリンク タイプ "DesignDoc" を作成しました。
```

ハイパーリンク継承機能を使用すると、次のように簡単に要求追跡を実装できます。

- ソース モジュール **hello.c** と設計文書 **hello_dsn.doc** が更新されると、プロジェクト マネージャーはこの 2 つの更新されたバージョンを接続する新規のハイパーリンクを作成します。

```
cleartool mkhlink -c "source doc" DesignDoc hello.c /vobs/design/hello_dsn.doc
ハイパーリンク "DesignDoc@90@/vobs/dev" を作成しました。
```

- ソース モジュールまたは設計文書に小さな更新が取り込まれても、ハイパーリンク レベルの変更は必要ありません。新規バージョンはその祖先の接続を継承します。

```
cleartool checkin -c "fix bug" hello.c
"hello.c" のバージョン "/main/2" をチェックインしました。
```

継承されたハイパーリンクを表示するには、**describe** コマンドに **-ihlink** オプションを使用します。

ハイパーリンクが継承されたバージョン -> % cleartool describe -ihlink DesignDoc hello.c@@/main/2
hello.c@@/main/2

ハイパーリンクが明示的に関連付けられているバージョン -> 継承されたハイパーリンク : DesignDoc@90@/vobs/dev
/vobs/dev/hello.c@@/main/1 ->
/vobs/doc/hello_dsn.doc@@/main/1

ハイパーリンクが継承されたバージョン -> cleartool describe -ihlink DesignDoc hello.c@@¥main¥2
hello.c@@¥main¥2

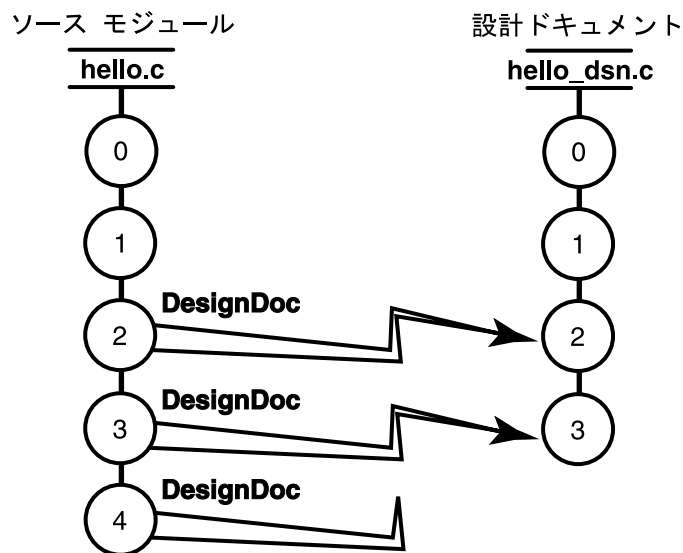
ハイパーリンクが明示的に関連付けられているバージョン -> 継承されたハイパーリンク : DesignDoc@90@¥dev
¥dev¥hello.c@@¥main¥1 ->
¥doc¥hello_dsn.doc@@¥main¥1

- ソース モジュールまたは設計文書のいずれかに重要な更新が取り込まれると、接続が無効になり、プロジェクト マネージャーは接続を切断する NULL で終わるハイパーリンクを作成します。

cleartool mkhlink -c "sever connection to design doc" DesignDoc hello.c
ハイパーリンク "DesignDoc@94@/vobs/dev" を作成しました。

図 43 は、ソース ファイルと設計文書を接続するハイパーリンクを示しています。

図 43 要求追跡



特定コマンドの使用の禁止

ClearCase オブジェクトに対して特定のコマンドを実行できるユーザーを管理するために、トリガ タイプのペアを作成することができます。ペアの一方でエレメント関連のオブジェクトに対するコマンドの使用を管理し、もう一方でタイプ オブジェクトに対するコマンドの使用を管理します。両方のトリガ タイプでは、**-nuser** フラグを使用して、そのコマンドを使用できるユーザーを指定します。

メモ: エレメント関連またはタイプ オブジェクト以外のオブジェクトに使用するコマンドを禁止するトリガは使用できません。たとえば、VOB オブジェクトまたはレプリカ オブジェクトに対する操作を制限するトリガ タイプは作成できません。

トリガ可能なコマンドの一覧については、**events_ccase** と **mktrtype** のリファレンス ページを参照してください。

たとえば、以下のコマンドは、2 つのトリガ タイプを作成します。これらのトリガ タイプは、**stephen**、**hugh**、**emma** 以外のすべてのユーザーが、現在の VOB 内にあるエレメント関連のオブジェクトとタイプ オブジェクトに対して **chmaster** コマンドを実行できないようにします。

```
cleartool mktrtype -element -all -preop chmaster -nusers stephen,hugh,emma ¥
-execunix 'Perl -e "exit -1;"" -execwin 'ccperl -e "exit (-1);"" ¥
-c "ACL for chmaster" elem_chmaster_ACL
```

```
cleartool mktrtype -type -preop chmaster -nusers stephen,hugh,emma ¥
-execunix 'Perl -e "exit -1;"" -execwin 'ccperl -e "exit (-1);"" ¥
-attype -all -brtype -all -eltype -all -lbtype -all -hltype -all ¥
-c "ACL for chmaster" type_chmaster_ACL
```

tony というユーザーが、制限されたオブジェクトに対して **chmaster** コマンドを実行しようとすると、コマンドは失敗します。例を次に示します。

```
cleartool chmaster -c "give mastership to london" london@/vobs/dev ¥
/vobs/dev/acc.c@@/main/lex_dev
```

```
cleartool: 警告 : トリガ "elem_chmaster_ACL" で chmaster の続行が拒否されました。
cleartool: エラー : 操作 "change master" を実行できません (レプリカ "lex"、VOB
"/vobs/dev")。
```

特定ブランチの MultiSite のサイト間での共有

製品メモ: 現在、Rational ClearCase LT は ClearCase MultiSite をサポートしていません。

ClearCase MultiSite を使用して、異なるサイトで開発を行っている場合、ブランチ作成ポリシーをこれらのサイトの要求に応じて調整する必要があります。標準 MultiSite 開発モデルでは、各サイトに VOB のレプリカが存在します。各レプリカは、サイト固有のブランチタイプを管理 (マスター登録) します。あるサイトの開発者は、別のサイトで管理 (マスター登録) されているブランチ上では作業できません (MultiSite でのマスタースhipの詳細については、『Rational ClearCase MultiSite 管理ガイド』を参照してください)。

エレメントにブランチを作成してマージすることができない、またはマージしたくない場合もあります。たとえば、ファイルタイプにはマージできないものがあるので、開発は 1 つのブランチで行う必要があります。この場合、すべての開発者は 1 つのブランチ (通常は main ブランチ) で作業する必要があります。MultiSite では、1 つのブランチを管理できるのは一度に 1 つのレプリカだけです。そのため、別のサイトの開発者が、そのエレメントに対して作業する場合、ブランチのマスタースhipをそのサイトに転送する必要があります。

MultiSite は、ブランチのマスタースhipを転送するためのモデルを 2 つ提供しています。

- プッシュ モデル: ブランチを管理するレプリカの管理者は、**chmaster** コマンドを使用して別のレプリカにマスタースhipを与えます。

このモデルは、ブランチを共有している状況下では効率的ではありません。リモートサイトの管理者との通信が必要になるからです。このモデルの詳細については、『Rational ClearCase MultiSite 管理ガイド』を参照してください。

- プル モデル: ブランチ上で作業する開発者が、**reqmaster** コマンドを使用してブランチのマスタースhipを要求します。

メモ: 開発者は、ブランチタイプのマスタースhipを要求することもできます。詳細については、『Rational ClearCase MultiSite 管理ガイド』を参照してください。

このモデルでは、MultiSite の管理者がそれぞれのレプリカのマスタースhip要求を行うことができるようにして、個々の開発者によるマスタースhip要求を許可する必要があります。このモデルを実装する場合、次の情報を MultiSite 管理者に与える必要があります。

- マスタースhip要求を扱うために、有効にする必要のあるレプリカ VOB
- マスタースhip要求を行うために、許可する必要のある開発者の識別情報 (ドメイン名とユーザー名)
- マスタースhip要求を拒否すべきブランチタイプとブランチ (たとえば、サイト固有のブランチタイプや、1 つのサイトで管理し続ける必要のあるブランチ)

『Rational ClearCase MultiSite 管理ガイド』では、プル モデルを有効にするプロセスと、プル モデルを使用するシナリオについて説明しています。『Rational ClearCase ソフトウェア開発ガイド』では、開発者がマスターシップ要求に使用するプロシージャについて説明しています。

UNIX と Windows 間でのトリガの共有

UNIX と Windows コンピュータの両方で正しく起動するトリガを定義することができます。ここでは、2 つの方法について説明します。1 つはプラットフォームごとに異なるパス名または異なるスクリプトを使用し、もう 1 つは両方のプラットフォームで同じスクリプトを使用します。

異なるパス名または異なるスクリプトの使用

UNIX か Windows、またはその両方で起動し、トリガ スクリプトを示すために異なるパス名を使用するトリガを定義する場合、**mktrtype** コマンドに **-execunix** と **-execwin** オプションを付けて実行します。これらのオプションは、適切なプラットフォーム (UNIX または Windows) で起動されると、**-exec** と同じ振る舞いをします。適切ではない方のプラットフォームでは何もありません。この方法を使用すると、1 つのトリガタイプで同一スクリプトの別のパスを使用したり、UNIX と Windows コンピュータ上で完全に異なるスクリプトを使用することができます。例を次に示します。

```
cleartool mktrtype -element -all -nc -preop checkin ¥  
-execunix /public/scripts/precheckin.sh -execwin ¥¥neon¥scripts¥precheckin.bat ¥  
pre_ci_trig
```

UNIX 上では、スクリプト **precheckin.sh** だけが実行されます。Windows 上では、**precheckin.bat** だけが実行されます。

新しいプラットフォームでユーザーがこのトリガ プロセスを回避しないように、**-execunix** だけを指定したトリガは Windows 上では常に失敗します。同様に、**-execwin** だけを指定したトリガは UNIX 上では失敗します。

同一スクリプトの使用

Windows と UNIX プラットフォームの両方で同一のトリガ スクリプトを使用する場合、両方のオペレーティング システムで実行できるバッチ コマンド インタープリタを使用する必要があります。このため、ClearCase には **ccperl** プログラムが付属しています。**ccperl** は、UNIX 上で使用できる Perl プログラムの Windows バージョンです。

次の `mktrtype` コマンドは、`pre_ci_trig` というサンプル トリガ タイプを作成し、実行可能トリガ スクリプトとして `precheckin.pl` という名前を付けます。

```
% cleartool mktrtype -element -all -nc -preop checkin ¥  
    -execunix 'Perl /public/scripts/precheckin.pl' ¥  
    -execwin  'ccperl ¥¥neon¥scripts¥precheckin.pl' ¥  
pre_ci_trig
```

メモ

- オペレーティング システムに基づいて、スクリプトの実行の条件を設定するために、Perl スクリプト内で環境変数を使用します。
- 対話式に情報の収集と表示を行うために、`clearprompt` コマンドを使用することができます。
- `-execunix` と `-execwin` オプションの使用法の詳細については、`mktrtype` のリファレンス ページを参照してください。

ベース ClearCase と ClearQuest の統合の設定

13

この章では、ベース ClearCase と ClearQuest の統合の概要と、統合の設定方法について説明します。統合作業の詳細については、『Rational ClearCase ソフトウェア開発ガイド』を参照してください。

統合の概要

Rational ClearQuest は、プロジェクトまたは製品の障害や要求変更をレポートする変更依頼を管理します。Rational ClearCase は、プロジェクトまたは製品を表すエレメントのバージョンを管理します。各バージョンは、エレメントに対する 1 つ以上の変更を示します。

ClearQuest とベース ClearCase の統合では、1 つ以上の ClearQuest 変更依頼を 1 つ以上の ClearCase バージョンに関連付けます。

1 つの変更依頼が複数のバージョンと関連付けられることもあります。依頼された変更を実装する一連のバージョンは、その依頼の変更セットと呼ばれます。

1 つのバージョンが複数の変更依頼と関連付けられることもあります。これらの変更依頼は、そのバージョンの要求セットと呼ばれます。

統合には、次のインターフェイスがあります。

- ClearCase プロジェクト マネージャーは、バージョンと変更依頼を関連付けるようユーザーに要求する際の条件を指定します。ユーザーが変更依頼を関連付けることが可能であるか、関連付ける必要のある VOB、ブランチ、エレメント タイプを指定できます。
- ClearQuest 管理者は、ClearQuest スキーマに ClearCase 定義を追加します。これらの定義により、スキーマを使用するデータベース内の変更依頼は、関連する変更セットの取得と表示を行うことができます。
- ClearCase 開発者は、次のことが可能です。
 - バージョンのチェックインまたはチェックアウト時に、バージョンと 1 つ以上の変更依頼を関連付けることができます。
 - 依頼の変更セットを参照できます。
 - クエリーを登録し、一定期間プロジェクトと関連付けられている変更依頼を識別できます。

ClearQuest と ClearCase の設定

開発者が ClearCase バージョンと ClearQuest 変更依頼を関連付けるには、以下の手順に従って ClearQuest と ClearCase を設定する必要があります。

- 1 ClearCase 定義を ClearQuest スキーマに追加します。ClearQuest 2.0 (またはそれ以上のリリース) がインストールされている場合、ClearQuest Designer のパッケージ ウィザードを使用してこれらの定義を追加します。それ以前のリリースの ClearQuest がインストールされている場合、Windows の ClearQuest 統合の構成アプリケーションを使用して定義を追加します。ClearCase 定義を 1 つ以上のレコードタイプとそれに関連するフォームに関連付けます。各フォームには、変更依頼の変更セットを表示する [ClearCase] タブがあります。
- 2 ClearQuest Designer を使用して、新しいバージョンのスキーマでデータベースをアップグレードします。ClearQuest Designer ヘルプの「既存のデータベースのアップグレード」を参照してください。統合を別のデータベースに移行する場合は、必ずそのデータベースでこの手順を反復します。たとえば、実働データベースでの採用を決定する前に、サンプルデータベースで試験的に統合を使用できます。
- 3 ClearQuest 統合の構成アプリケーションを使用して、バージョンと変更依頼を関連付けるようユーザーに要求する際の条件を決定する各 VOB のポリシーを設定します。バージョンのチェックアウト時かチェックイン時、またはその両方でユーザーに要求するよう指定できます。この要求は、一部のブランチ タイプまたはエレメント タイプのみを対象にするよう指定することもできます。チェックイン バージョンと変更依頼の関連付けは、オプションまたは必須のいずれでもかまいません。
- 4 統合では、開発者は ClearCase のトリガを使用して変更依頼とバージョンを関連付けることが可能になります。V2 トリガを使用する場合、構成ファイルを変更してデータベース接続情報と追加のポリシー パラメータを設定する必要があります。トリガの詳細については、203 ページの「ClearCase VOB へのトリガのインストール」を参照してください。

ClearQuest スキーマへの ClearCase 定義の追加

ClearQuest スキーマには、レコードタイプ、フィールド、フォームの定義を含む一連の ClearQuest ユーザー データベースに関連する属性が含まれます。ClearCase バージョンと ClearQuest 変更依頼をユーザー データベース内で関連付ける前に、一部の ClearCase 定義をデータベースが使用するスキーマに追加する必要があります。これを行うには、ClearQuest Designer で次のようにパッケージ ウィザードを使用します。

- 1 [スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearQuest] の順にポイントし、[ClearQuest Designer] をクリックします。
- 2 ClearQuest Designer で、[パッケージ] メニューの [パッケージ ウィザード] をクリックします。

- 3 パッケージ ウィザードで、ClearCase 1.0 パッケージと ClearCase Upgrade 1.0 パッケージを見つけます。これらのパッケージが表示されていない場合、[その他のパッケージ] をクリックして、[パッケージのインストール] ダイアログ ボックスからリストにパッケージを追加します。
- 4 ClearCase と ClearQuest の統合で使用するスキーマを初めて有効にする場合、ClearCase 1.0 を選択して [次へ] をクリックします。ClearCase と ClearQuest の統合で現在使用しているスキーマを ClearQuest リリース 1.1 からリリース 2.0 にアップグレードする場合、ClearCase Upgrade 1.0 を選択して [次へ] をクリックします。
- 5 ClearCase との統合に使用する ClearQuest ユーザー データベースのスキーマを選択します。[次へ] をクリックします。
- 6 V1 トリガを使用する場合、ClearCase バージョンと関連付ける ClearQuest レコードのレコード タイプを選択します。このレコード タイプは、ClearQuest 統合の構成アプリケーションの [ClearCase] タブで指定するレコード タイプと一致する必要があります。V2 トリガを使用する場合、複数のレコード タイプを指定できます。[完了] をクリックします。
- 7 [ファイル] メニューの [チェックイン] をクリックして、新しいバージョンのスキーマを保存します。
- 8 [データベース] の [データベースのアップグレード] をクリックして、ClearQuest ユーザー データベースを新しいバージョンのスキーマでアップグレードします。

リリース 2.0 より前の ClearQuest を使用している場合、ClearQuest 統合の構成アプリケーションを使用して、ClearCase 定義を ClearQuest スキーマに追加します。アプリケーションを開始するには、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[管理]、[統合]、[ClearQuest 統合の構成] の順にクリックします。または、コマンド プロンプトで `cqconfig` と入力してもアプリケーションを開始できます。[ClearQuest] タブをクリックします。タブのフィールドを完成するための指示を参照するには、[ヘルプ] をクリックします。

ClearCase VOB へのトリガのインストール

統合では、`cleartool checkin`、`checkout`、`uncheckout` コマンドに ClearCase のトリガを使用して、バージョンと ClearQuest 変更依頼を関連付けることができます。これらのトリガを VOB にインストールするには、ClearQuest 統合の構成アプリケーションを使用します。

製品メモ: ClearQuest 統合の構成を開始するには

- ClearCase の場合は、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[管理]、[統合]、[ClearQuest 統合の構成] の順にクリックします。
- ClearCase LT の場合は、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase LT] の順にポイントし、[ClearQuest 統合の構成] の順にクリックします。

または、コマンドプロンプトで **cqconfig** と入力してもアプリケーションを開始できます。

ClearCase v2002.05.00 より前のバージョンの場合、統合には Windows で Visual Basic トリガを使用し、UNIX で Perl トリガを使用していました。ClearCase v2002.05.00 では、Windows と UNIX 上で実行される新しい Perl トリガが追加されました (このトリガは、ClearCase リリース 4.2 に対するパッチでも利用可能です)。アプリケーションの [Windows のトリガ選択] と [UNIX トリガ選択] フィールドにある [V1] または [V2] をクリックして、使用するトリガを指定します。[V1] は、従来の Visual Basic トリガと Perl トリガです。[V2] は、異なるプラットフォーム間で使用する新しい Perl トリガです。

V2 トリガは、cleartool コマンド行インターフェイスを使用する開発者にはテキストベースのユーザー インターフェイスを、ClearCase エクスプローラ (Windows) または xclearcase (UNIX) などの ClearCase GUI のいずれかを使用する開発者には GUI を提供します。初めて統合を設定する場合は、V2 トリガを使用することをお勧めします。現在 V1 トリガを使用している場合は、V2 トリガを評価して、可能であれば移行することをお勧めします。

V2 トリガは、ローカルの構成パラメータを指定する構成ファイルを使用します。V2 トリガを選択すると、構成アプリケーションによって、構成ファイルへのパスである CQCC/config.pl が [パス] フィールドに指定されます。このパスでは、CQCC が各ローカル クライアント上の ccase-home-dir/lib/perl5/CQCCTrigger/CQCC に変換されます。このパスを UNC パス名に変更し、統合で 1 つの中心的な構成ファイルを使用するようにできます。

アプリケーションのほかのフィールドを完成する場合の情報については、アプリケーション内で [ヘルプ] をクリックしてください。

評価用のクイック スタート (V2 トリガのみ)

デフォルトの構成ファイルは、ClearQuest が評価用に提供する SAMPL ユーザー データベースを使用するように設定されています。ClearQuest の SAMPL ユーザー データベースを使用して統合をテストすることができます。ClearQuest がクライアント コンピュータにインストールされている場合、統合では ClearQuest ユーザー データベースとの通信に ClearQuest Perl API を使用します。ClearQuest がクライアント コンピュータにインストールされていない場合、統合では ClearQuest ユーザー データベースとの通信に ClearQuest Web インターフェイスを使用します。

ClearQuest Web インターフェイスの環境変数の設定

クライアントで ClearQuest Web インターフェイスを有効にするには、コマンド行プロンプトまたは構成ファイルで次のように環境変数を設定します。

- CQCC_SERVER: ClearQuest Web サーバーの存在するホストの名前
- CQCC_SERVERROOT: ClearQuest Web インターフェイス ファイルがインストールされているルート ディレクトリ
- CQCC_WEB_DATABASE_SET: データベース セットの名前

データベース セットにより、ユーザーは、ClearQuest を開始するか、ベース ClearCase と ClearQuest の統合を開始するときに複数のスキーマ リポジトリから選択を行うことが可能になります。CQCC_WEB_DATABASE_SET 環境変数は、サイトで複数のデータベース セットを使用する場合にのみ設定します。

ClearQuest Perl API の環境変数の設定

クライアントで ClearQuest Perl API を有効にするには、場合により CQCC_DATABASE_SET 環境変数を設定する必要があります。データベース セットにより、ユーザーは、ClearQuest を開始するか、ベース ClearCase と ClearQuest の統合を開始するときに複数のスキーマ リポジトリから選択を行うことが可能になります。CQCC_DATABASE_SET 環境変数は、サイトで複数のデータベース セットを使用する場合にのみ設定します。

構成ファイルの編集 (V2 トリガのみ)

構成ファイルには、ローカル ポリシー選択と ClearQuest へのアクセス方法を定義するパラメータが含まれます。構成ファイルは、SAMPL ユーザー データベースにアクセスし、defect レコードタイプを使用するように設定されています。別の ClearQuest ユーザー データベースまたはレコードタイプとの統合を使用するには、パラメータを変更する必要があります。構成ファイルには、パラメータの設定方法を記述した文書が含まれます。構成ファイルに関する追加の情報は、構成ファイルと同じフォルダにある README ファイルで参照可能です。構成ファイルを編集するには、権限を変更してファイルを変更可能にします。

統合のテスト (V2 トリガのみ)

1 つ以上の VOB にトリガをインストールし、構成ファイルを編集した後、次のコマンドを入力して ClearCase と ClearQuest 間の接続をテストします。

Windows では次のようになります。

```
cqcc_launch CQCC\config.pl -test
```

UNIX では次のようになります。

```
cqcc_launch CQCC/config.pl -test
```

メモ: これらのコマンドでは、構成ファイルのデフォルトのパスを使用しています。トリガのインストール時に異なるパスを指定した場合、cqcc_launch コマンドを起動するときにはそのパスが使用されます。

このコマンドは、ターゲットの ClearQuest ユーザー データベースに接続可能であるかどうかを示す出力を表示します。詳細な出力メッセージを取得するには、CQCC_DEBUG 環境変数を 2 に設定します。

パフォーマンスのチェック (V2 トリガのみ)

統合トリガのパフォーマンスは、ClearCase と ClearQuest へのアクセス方法によって変化することがあります。CQCC_TIMER 環境変数を 1 に設定すると、トリガセッションに関するタイミング情報を記録できます。統合では、標準出力と、TMPDIR 環境変数により定義されるディレクトリ内の cqcc_output.log に情報が書き込まれます。

統合クエリー ウィザードの使用法

ClearCase バージョンと ClearQuest 変更依頼の間に関連付けを確立した後は、Windows の ClearQuest 統合クエリー ウィザードを使用して、一定期間プロジェクトと関連付けられている変更依頼を識別できます。ウィザードを使用すると、たとえば「プロジェクト X のリリース 3.1 に関連付けられていた変更依頼はどれか」というような質問に答えることができます。

Windows の [スタート] メニューからウィザードを開始するには、[スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[管理]、[ClearQuest 統合のクエリー] の順にクリックします。または、コマンドプロンプトで **cqquery** と入力してもウィザードを開始できます。ウィザードの各ページを完成するための指示を参照するには、[ヘルプ] をクリックします。

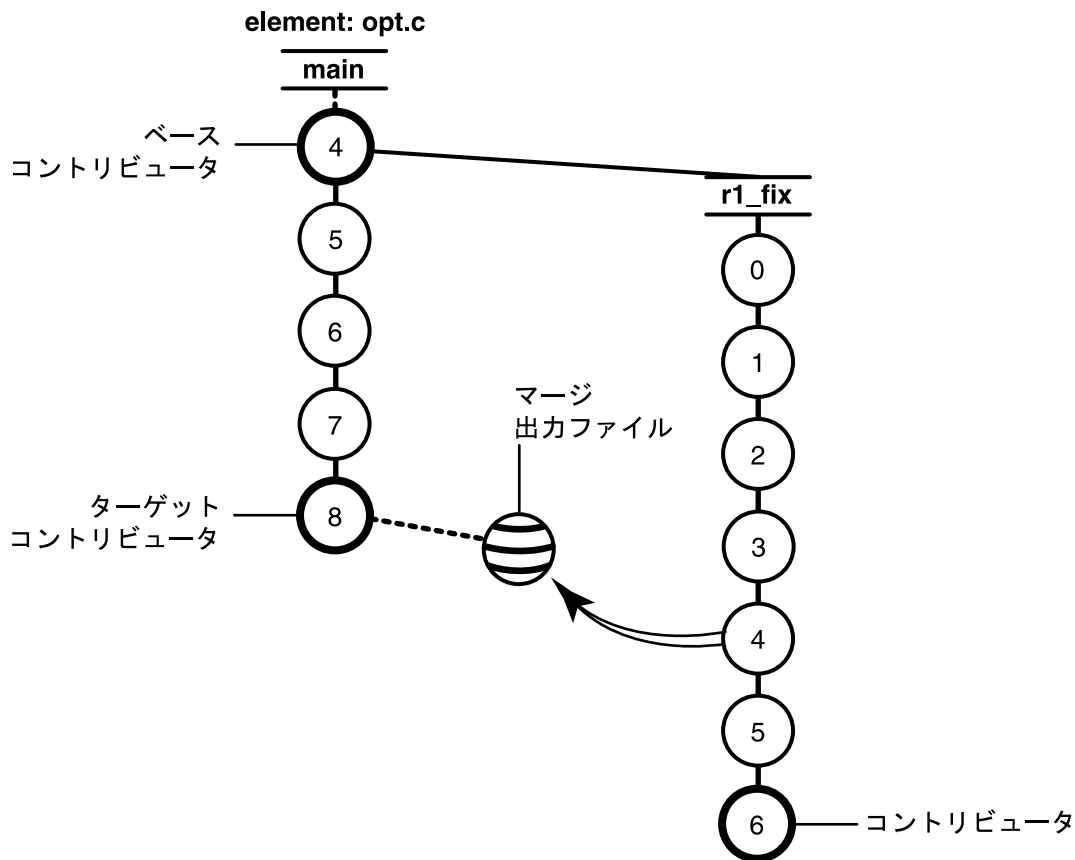
並行開発環境では、ブランチ作成の逆がマージになります。最も単純なシナリオでは、マージによってサブブランチ上の変更が **main** ブランチに組み込まれます。任意のブランチからほかのブランチにマージすることもできます。この章では、エレメントとブランチのバージョンをマージする方法といくつかのシナリオについて説明します。**ClearCase** には、ほとんどすべてのシナリオを扱える自動マージ機能があります。

マージの仕組み

マージは、2 つ以上のファイルまたはディレクトリの内容を、1 つの新規ファイルやディレクトリに組み込みます。**ClearCase** のマージアルゴリズムでは、マージ中に次のファイルを使用します (図 44 を参照)。

- コントリビュータ: 通常はマージする各ブランチのバージョンの 1 つです (15 のコントリビュータまでマージできます)。どのバージョンをコントリビュータにするかを指定します。
- ベース コントリビュータ: 通常はコントリビュータ群の最も近い共通の祖先です (選択マージ、減法マージ、複雑なブランチ構造環境でのマージでは、ベース コントリビュータが最も近い共通の祖先でない場合があります)。**ClearCase** によってベース コントリビュータが決定されます。
- ターゲット コントリビュータ: 通常はマージ結果を含むブランチの最新バージョンです。どのコントリビュータをターゲット コントリビュータにするかを決定します。
- マージ出力ファイル: これはマージの結果であり、通常はターゲット コントリビュータの後続バージョンになります。デフォルトでは、マージ出力ファイルはターゲット コントリビュータのチェックアウト バージョンです。別のファイルをマージ出力に選択することもできます。

図 44 一般的なマージに関連するバージョン

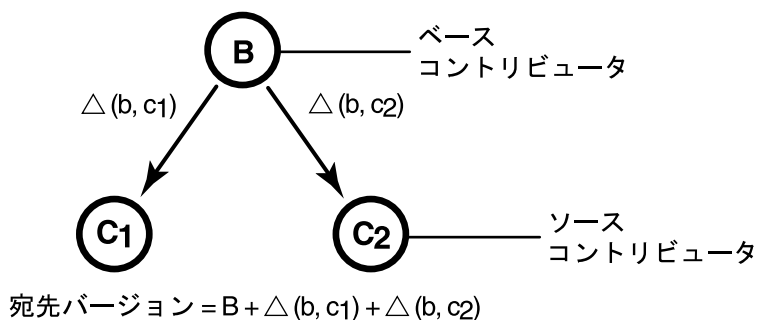


ファイルとディレクトリをマージするために、ClearCase は次の手順を実行します。

- 1 ベース コントリビュータを識別します。
- 2 それぞれのコントリビュータをベース コントリビュータと比較します (図 45 を参照)。
- 3 ベース コントリビュータとほかのコントリビュータとの間で変更されていない行については、マージ出力ファイルにコピーします。
- 4 ベース コントリビュータとほかの 1 つのコントリビュータとの間で変更された行については、コントリビュータの変更を受け入れます。これは、マージ操作の開始方法に依存します。ClearCase によって自動的にマージ出力ファイルに変更をコピーすることができますが、任意のマージ操作に対するこの自動マージ機能を無効にすることもできます。この機能を無効にした場合は、マージ出力ファイルへの変更を個別に承認する必要があります。

- 5 ベース コントリビュータと 2 つ以上のコントリビュータとの間で変更された行については、ClearCase からその矛盾を解決するように要求されます。

図 45 ClearCase のマージ アルゴリズム



バージョンのマージには、次の項で簡単に説明する GUI ツールや、210 ページの「コマンド行を使用したエレメントのマージ」に説明のあるコマンド行インターフェイスを使用することができます。

GUI を使用したエレメントのマージ

ClearCase には、エレメントのマージを支援する 3 つのグラフィカル ツールが付属しています。

- マージ マネージャ
- 差分マージ
- バージョン ツリー ブラウザ

マージ マネージャは、1 つ以上の ClearCase エレメントをマージするプロセスを管理します。これは、マージのための情報収集、マージの開始、マージの追跡の各プロセスを自動化します。また、一連のエレメントのマージ状況の保存と取得を行うこともできます。

マージ マネージャは、次のさまざまな方向のマージに使用することができます。

- あるブランチから main ブランチ
- main ブランチから別のブランチ
- あるブランチから別のブランチ

UNIX: マージ マネージャを起動するには、コマンドプロンプトで `clearmrgman` と入力します。

Windows: マージ マネージャは、次のいくつかの方法で起動することができます。

- [スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[マージ マネージャ] をクリックします。
- ClearCase エクスプローラ で、[ベース ClearCase] をクリックしてから、[マージ マネージャ] をクリックします。

差分マージユーティリティは、ファイルまたはディレクトリ エLEMENTの 2 つ以上のバージョン間の相違を表示します。このツールを使用すると、一度に 16 バージョンまでの比較、バージョン間の移動、バージョンのマージ、バージョン間の相違の解決を行うことができます。

UNIX: 差分マージユーティリティを起動するには、**xcleardiff** と入力するか、コマンドプロンプトで **cleartool merge -graphical** コマンドを使用します。

Windows: 差分マージは、次のいくつかの方法で起動することができます。

- Windows エクスプローラのショートカット メニューで [比較] をクリックします。
- マージマネージャで、[比較] をクリックします。

バージョン ツリー ブラウザ は、ELEMENTのバージョン ツリーを表示します。バージョン ツリーは、マージの際に次のことを行う場合に役立ちます。

- マージ元やマージ結果のバージョンまたはブランチの特定
- 適切なシンボルをクリックすることによるマージの開始

マージは、マージ矢印を使用して記録することができます。これは、**Merge** タイプのハイパーリンクとして実装されています。

UNIX: バージョン ツリー ブラウザを起動するには、次のいずれかの方法を使用します。

- コマンド プロンプトで **cleartool lsvtree -graphical** と入力します。
- ClearCase ファイル ブラウザで、ELEMENTをクリックして、[バージョン]、[バージョン ツリーの表示] の順にクリックします。

Windows: バージョン ツリー ブラウザは、次のいくつかの方法で起動することができます。

- [スタート] メニューから、[プログラム]、[Rational Software]、[Rational ClearCase] の順にポイントし、[バージョン ツリー ブラウザ] をクリックします。
- Windows エクスプローラのショートカット メニューで [バージョン ツリー] をクリックします。

コマンド行を使用したELEMENTのマージ

コマンド行からマージを実行するには、次のコマンドを使用します。

- **cleartool merge**
- **cleartool findmerge**
- **cleardiff**

これらのコマンドの詳細については、『Rational ClearCase リファレンス ガイド』を参照してください。

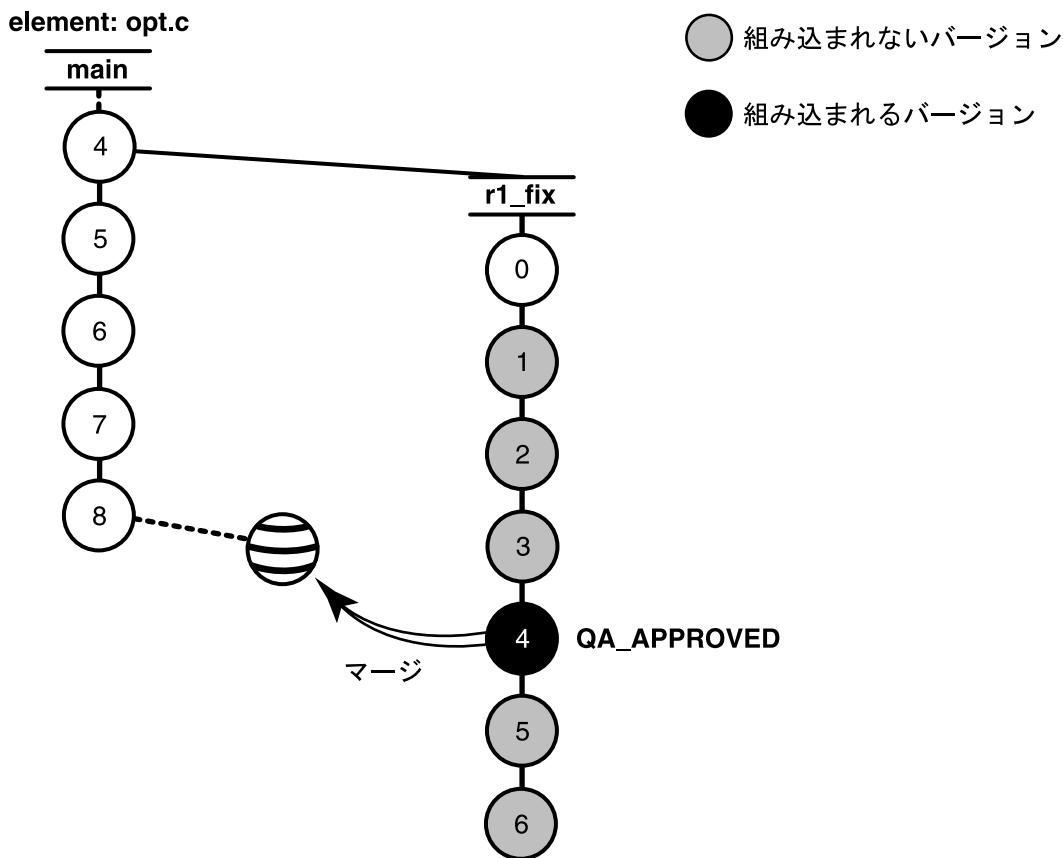
一般的なマージのシナリオ

これから、あるエレメントのブランチ上での作業を、別のブランチに組み込む場合に必要なマージに関する一連のシナリオを説明します。各シナリオでは、マージに必要なエレメントのバージョンツリーと、マージを実行するための適切なコマンドを示しています。

シナリオ: サブブランチからの選択マージ

このシナリオでは、バージョン /main/r1_fix/4 の変更を新規開発に組み込みます。マージを実行するには、r1_fix ブランチ上のどのバージョンを組み込むかを指定します。図 46 を参照してください。

図 46 サブブランチからの選択マージ



デフォルト構成仕様で構成されたビューで、次のコマンドを入力して選択マージを実行します。

```
% cleartool checkout opt.c
% cleartool merge -to opt.c -insert -version /main/r1_fix/4
```

連続する範囲のバージョンを指定してマージすることもできます。たとえば、次のコマンドはバージョン /main/r1_fix/2 から /main/r1_fix/4 までの変更のみをマージします。

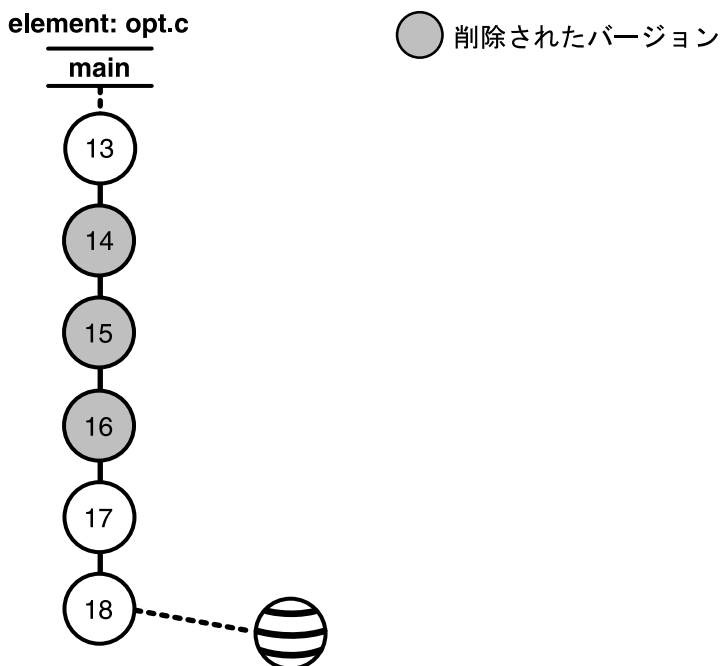
```
% cleartool merge -to opt.c -insert -version /main/r1_fix/2 /main/r1_fix/4
```

選択マージでは、マージ矢印は作成されません。

シナリオ：特定バージョンからの変更内容の削除

main ブランチのバージョン 14 から 16 までの間に実装された新機能は、製品に組み込みません。これらのバージョンで加えられた変更は、削除する必要があります。図 47 を参照してください。

図 47 特定バージョンからの変更内容の削除



この減法マージを実行するには、次のコマンドを入力します。

```
% cleartool checkout opt.c
% cleartool merge -to opt.c -delete -version /main/14 /main/16
```

減法マージでは、マージ矢印は作成されません。

シナリオ: すべてのプロジェクト作業のマージ

チームがあるブランチ上で作業をして、すべての変更を **main** ブランチにマージすることになりました。

findmerge コマンドは、ほとんどの状況に簡単に対応できます。このコマンドでは、プロジェクトの作業を隔離するために以下の方式が提供されます。

すべてのプロジェクト作業のブランチ上への隔離

並行開発では、すべてのプロジェクト作業を同一のブランチ上に隔離することが標準的なアプローチです。より厳密に言えば、ソース ファイルのすべての新規バージョンを、それぞれのエレメントの同じ名前のブランチ上 (つまり、同一のブランチ タイプのインスタンスであるブランチ上) に作成します。こうすると、1 回の **findmerge** コマンドで、すべての変更を検索して組み込むことが可能になります。共通のブランチが **gopher** という名前であるとしします。デフォルト構成仕様によって構成されたビューで、次のコマンドを入力します。

```
% cd root-of-source-tree
% cleartool findmerge . -fversion .../gopher/LATEST -merge -graphical
```

-merge -graphical 構文を使用すると、マージが可能であれば自動的に行われます。エレメントのマージにユーザーの確認が必要な場合には、マージユーティリティが起動します。プロジェクトにより複数の **VOB** が変更された場合、複数のパス名を指定するか、**findmerge** に **-avobs** オプションを使用することで、すべてのマージを一度に実行することができます。

すべてのプロジェクト作業のビュー内への隔離

プロジェクトの中には、すべての変更を単一のビュー (通常、共有ビュー) 内で行うように編成されているものがあります。このようなプロジェクトでは、**findmerge** に **-ftag** オプションを使用します。プロジェクトの作業が **goph_vu** というビュー タグを持つビュー内で行われたとすると、次のコマンドはそのマージを実行します。

```
% cd root-of-source-tree
% cleartool findmerge . -ftag goph_vu -merge -graphical
```

メモ: 単一の共有ビュー内で作業することは、システムのパフォーマンスが低下する場合がありますのでお勧めしません。

シナリオ: ソース ツリー全体の新規リリースのマージ

開発チームが外部から提供されたソース コード製品を使用しており、**VOB** でそのソースを管理しているとしします。製造元から供給される後続バージョンは、**main** ブランチにチェックインされ、**VEND_R1**、**VEND_R2**、**VEND_R3** というラベルが関連付けられます。チームによる修正と拡張は、サブブランチ **enhance** 上に作成されます。チームが作業するビューは、**VEND_R3** ベースラインからブランチを作成するように構成されています。

```

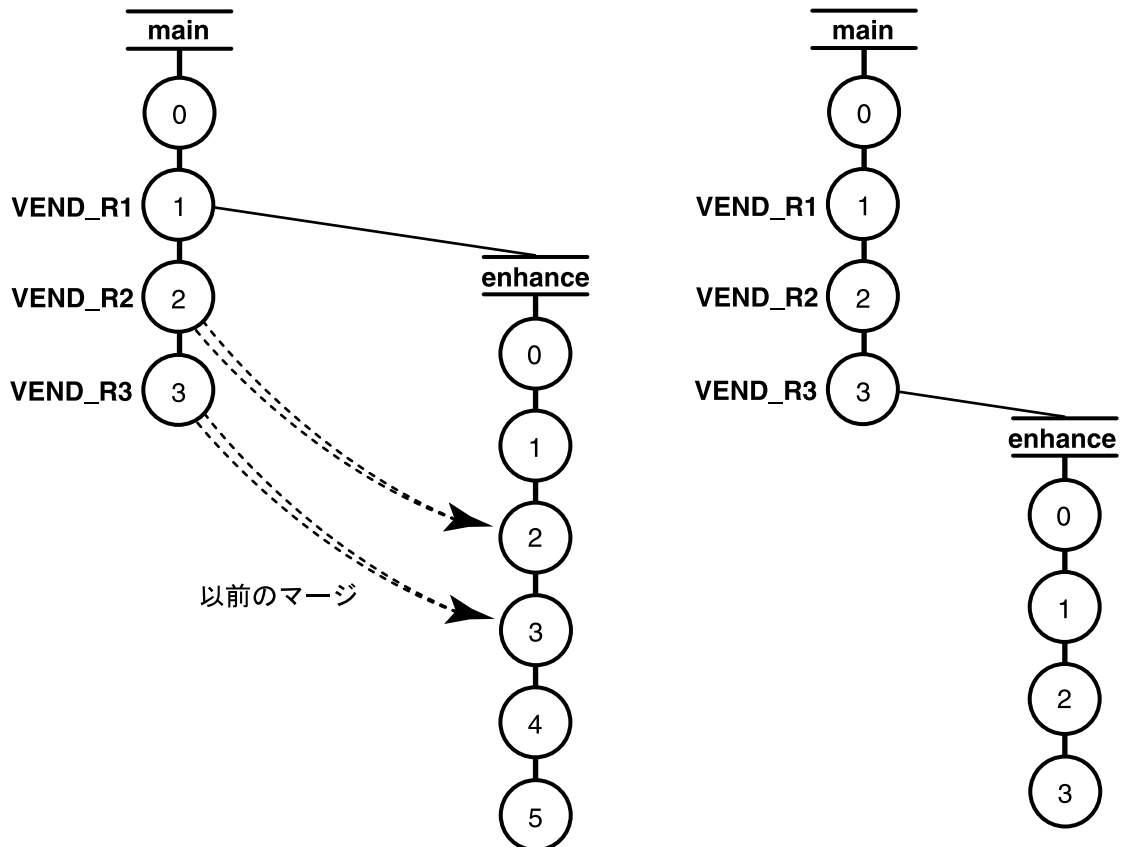
element * CHECKEDOUT
element * .../enhance/LATEST
element * VEND_R3 -mkbranch enhance
element * /main/LATEST -mkbranch enhance

```

図 48 のバージョンツリーは、次のようなさまざまな想定される状況を示しています。

- リリース 1 でチームが変更を開始したエレメント (リリース 1 は VEND_R1 というラベルのバージョンで作成された enhance ブランチ)
- リリース 3 でチームが変更を開始したエレメント
- チームが変更していないエレメント

図 48 ソース ツリー全体の新規リリースのマージ



リリース 4 が到着して、このリリースをチームの変更に統合する必要があるとします。

マージの準備として、新規リリースを **main** ブランチに追加し、**VEND_R4** というバージョンラベルを付けます。ソース ツリーをマージするには、ラベル **VEND_R4** の付いたバージョンから **enhance** ブランチの最新バージョンにマージします。エレメントに **enhance** ブランチがない場合、マージは行われません。

次の手順で統合を行います。

- 1 製造元のリリース 4 のメディアを標準ディレクトリ ツリーにロードします。

```
%cd /usr/tmp  
% tar -xv
```

mathlib_4.0 というディレクトリ ツリーが作成されます。

- 2 VOB 所有者として、**clearexport_ffile** を実行し、新規バージョンの説明を含むデータファイルを作成します。

```
% cd ./mathlib_4.0  
% clearexport_ffile  
. (省略)  
.
```

- 3 デフォルト構成仕様で構成されたビューで、**clearexport_ffile** で作成したファイルに **clearimport** を実行します。これにより、エレメントの **main** ブランチ上にリリース 4 バージョンが作成されます (必要に応じて新規エレメントも作成されます)。

```
% cleartool setview mainline  
% cd /vobs/proj/mathlib  
% clearimport /usr/tmp/mathlib_4.0/cvt_data
```

- 4 新規バージョンにラベルを付けます。

```
% cleartool mklbtype -c "Release 4 of MathLib sources" VEND_R4  
ラベル タイプ "VEND_R4" を作成しました。  
% cleartool mklabel -recurse VEND_R4 /vobs/proj/mathlib  
. (省略)  
.
```

- 5 チームの構成仕様で構成されたビューに移動し、**enhance** ブランチ上のバージョンを選択します。

```
% cleartool setview enh_vu
```

- 6 **VEND_R4** の構成をビューにマージします。

```
% cleartool findmerge -nback /vobs/proj/mathlib -fver VEND_R4 -merge -graphical
```

-merge -graphical 構文は、可能であれば自動的にマージを行い、可能でない場合はマージツールを起動するように **findmerge** を調整します。

- 7 マージを検証し、変更されたエレメントをチェックインします。

これで、リリース 4 を新規ベースラインとして設定できました。チームの開発者は、次のようにして自分のビュー構成を更新することができます。

```
element * CHECKEDOUT
element * .../enhance/LATEST

element * VEND_R4 -mkbranch enhance          (VEND_R3 から VEND_R4 への変更)
element * /main/LATEST -mkbranch enhance
```

アクティブであったエレメントは、**enhance** ブランチで開発を続けることができます。エレメントが最初に更新されたときには、そのエレメントの **enhance** ブランチが **VEND_R4** パージョンで作成されます。

シナリオ: ディレクトリ バージョンのマージ

ClearCase の最も強力な機能の 1 つに、ディレクトリに対するバージョン付けがあります。ディレクトリ エレメントの各バージョンは、一連のファイルエレメント、ディレクトリ エレメント、VOB シンボリック リンク (UNIX) をカタログします。開発プロジェクトでは、ディレクトリの変更はファイルと同様に頻繁に発生します。別のブランチへの変更のマージは、ファイルのマージと同じように簡単です。

前の項にあるソース ツリーのシナリオを詳細に説明します。製造元がディレクトリ `/vobs/proj/mathlib/src` に次のようないくつかの変更を加えたとしてします。

- ファイルエレメントの `makefile`、`getcwd.c`、`fork3.c` が更新されました。
- ファイルエレメントの `readln.c` と `get.c` が削除されました。
- 新規ファイルエレメントの `newpaths.c` が作成されました。

`findmerge` を使用して **VEND_R4** のソース内で加えられた変更を **enhance** ブランチにマージすると、ファイルとディレクトリに対する変更は自動的に処理されます。次の `findmerge` の抜粋は、ディレクトリのマージアクティビティを示しています。

```
*****
<<< ディレクトリ 1: /vobs/proj/mathlib/src@@/main/3
>>> ディレクトリ 2: .@@/main/enhance/1
>>> ディレクトリ 3: .
*****

----- [ ディレクトリ 1 を削除しました ]-----|----- [ ディレクトリ 2 ]-----
get.c 19-Dec-1991 drp                               |-
*** 自動: REMOVE をディレクトリ 2 から適用しています
----- [ ディレクトリ 1 ]-----|----- [ ディレクトリ 2 を追加しました ]-----
                                   |- newpaths.c 08-Mar.21:49 drp

*** 自動: ADDITION をディレクトリ 2 から適用しています
----- [ ディレクトリ 1 を削除しました ]-----|----- [ ディレクトリ 2 ]-----
readln.c 19-Dec-1991 drp                               |-
*** 自動: REMOVE をディレクトリ 2 から適用しています
". ." のマージを記録しました。
```

ファイルとディレクトリの両方をマージする変更がある場合には、**findmerge** を 2 回実行することをお勧めします。最初はディレクトリをマージして、次にファイルをマージします。**findmerge** コマンドに **-print** オプションを使用しても、マージされるすべての内容はレポートされません。ディレクトリのマージが終了するまで、**findmerge** はマージ元バージョンの新規ファイルやサブディレクトリを参照しないからです。実行されるすべてのマージをレポートするには、**findmerge** を使用してディレクトリだけをマージし、次に **findmerge -print** を使用して必要なファイル マージに関する情報を取得します。その後で、ディレクトリに対し **uncheckout** コマンドを使用すれば、ディレクトリのマージをキャンセルすることができます。

独自のマージ ツールの使用法

エレメントのマージされたバージョンの作成は、手動によるか、利用可能な解析ツールと編集ツールを使用して行います。ターゲットバージョンをチェックアウトし、更新してからチェックインします。チェックインの直前 (または直後) に、**merge** コマンドを **-ndata (no data)** オプション付きで使用して、アクティビティを記録します。

```
% cleartool checkout nextwhat.c
```

```
"nextwhat.c" のチェックアウトの説明を入力してください :  
merge enhance branch
```

```
.
```

```
バージョン "/main/1" から "nextwhat.c" をチェックアウトしました。
```

%<チェックアウト バージョンにデータをマージするための独自のツールを起動します>

```
% cleartool merge -to nextwhat.c -ndata -version .../enhance/LATEST
```

```
"nextwhat.c" のマージを記録しました。
```

この形式の **merge** コマンドでは、ファイル システムのデータを変更せずに、単にマージ矢印 (**Merge** タイプのハイパーリンク) を指定したバージョンに関連付けます。この注釈を作成した後には、独自のマージは **ClearCase** ツールを使用して実行されたものと区別できなくなります。

エレメント タイプを使用したファイル エレメントの処理のカスタマイズ

15

ほとんどのプロジェクトでは、多くの異なるファイル タイプを使用します。たとえば、一般的なソフトウェア リリースでは、開発者は C 言語のソース ファイル、C 言語のヘッダー ファイル、バイナリ形式の文書ファイル、ライブラリ ファイルを使用して作業を行います。

ClearCase の VOB に格納されたすべてのファイルは、エレメント タイプに関連付けられます。ClearCase には、さまざまな種類のファイル タイプ用に定義済みのエレメント タイプがあります。すべてのエレメント タイプには、関連するタイプ マネージャがあり、各エレメントのバージョンに対する操作を処理します。

プロジェクト内のあるファイル タイプに対して、独自のエレメント タイプを作成し、そのファイルの処理をカスタマイズしたい場合があります。また、独自のタイプ マネージャを作成する場合もあります。

この章では、ClearCase がエレメント タイプとタイプ マネージャを使用してどのようにファイルの分類と管理を行うかについて説明します。また、ファイルの分類と管理をカスタマイズする方法についても説明します。

一般的なプロジェクトでのファイル タイプ

表 5 は、一般的な開発プロジェクトで使用されるファイルのリストを示しています。

表 5 一般的なプロジェクトで使用されるファイル (1/2)

ファイルのタイプ	識別のための特徴
ソース ファイル	
C 言語ソース ファイル	.c のファイル名拡張子
C 言語ヘッダー ファイル	.h のファイル名拡張子
FrameMaker バイナリ ファイル	.doc または .mif のファイル名拡張子 (ファイルの最初の行は <Maker で始まる)
UNIX: マニュアル ページ ソース ファイル	.1 - .9 のファイル名拡張子

表 5 一般的なプロジェクトで使われるファイル (2/2)

ファイルのタイプ	識別のための特徴
派生ファイル	
UNIX: ar(1) アーカイブ (ライブラリ)	.a のファイル名拡張子
Windows: ライブラリ、共有ライブラリ	.lib、.dll のファイル名拡張子
コンパイルされた実行可能ファイル	UNIX: システム アーキテクチャによって異なる。 Windows: .exe のファイル名拡張子

ClearCase によるエレメント タイプの割り当て方法

さまざまな状況で、ClearCase では、既存のファイル システム オブジェクトか、新規オブジェクトに使用される名前に対して、1 つ以上のファイル タイプが決定されます。新規にエレメントを作成してエレメント タイプを指定しない場合、ClearCase によりそのエレメントに対するファイル タイプが決定されます。

cc.magic のリファレンス ページに説明されているとおり、ファイル タイプの決定には、定義済みまたはユーザー定義のマジック ファイルが使用されます。マジック ファイルでは、ファイル名のパターン マッチング、stat(2) データ、標準 UNIX マジック ナンバーなどの多くの異なる方法を使用してファイル タイプを決定します。

たとえば、次のマジック ファイルは、表 5 に示す各タイプのファイル用にさまざまなファイル タイプを指定します。

UNIX マジック ファイルの例

```
(1)  c_src src_file text_file file:      -name "*.c" ;
(2)  hdr_file text_file file:            -name "*.h" ;
(3)  frm_doc binary_delta_file doc file: -magic 0, "<MakerFile" ;
(4)  manpage src_file text_file file:    -name "*. [1-9]" ;
(5)  archive derived_file file:          -magic 32, "archive" ;
(6)  sunexec derived_file file:          -magic 40, "SunBin" ;
```

Windows マジック ファイルの例

```
(1)  c_src src_file text_file file:      -name "*.c";
(2)  hdr_file text_file file:            -name "*.h" ;
(3)  frm_doc binary_delta_file doc file: -magic 0, "<MakerFile" ;
(4)  library derived_file file:          -name "*.lib";
(5)  program compressed_file:           -name "*.exe" ;
```

エレメント タイプとタイプ マネージャ

ClearCase では、ファイルのクラスに応じて処理を変えることができます。それは、エレメントを分類する際にエレメント タイプを使用しているからです。VOB 内の各ファイル エレメントは、エレメント タイプを持つ必要があります。エレメントは、作成時にタイプを取得します。これは、後で `chtype` コマンドを使用して変更できます (属性が属性タイプのインスタンスであり、バージョン ラベルがラベル タイプのインスタンスであるように、エレメントはエレメント タイプのインスタンスです)。

それぞれのエレメント タイプには、関連するタイプ マネージャがあります。これは、記憶プールからのバージョンの格納と検索を処理する一連のプログラムです (タイプ マネージャの動作については、`type_manager` のリファレンス ページを参照してください)。このため、ファイル エレメントのデータの処理方法は、そのエレメント タイプに依存します。

メモ: それぞれのディレクトリ エレメントにもエレメント タイプがあります。ただし、ディレクトリ エレメントはタイプ マネージャを使用しません。ディレクトリ バージョンの内容は、VOB データベースに格納され、記憶プールには格納されません。

エレメント タイプを指定せずにエレメントを作成すると、ClearCase は次のようにエレメント タイプを割り当てます。

- 1 1 つ以上のマジック ファイルを読み取り、エレメントの名前に対応するファイル タイプを検索します。
- 2 名前と一致するマジック ファイル内の最初の規則に関連するファイル タイプのリストを取得します。
- 3 このリストを VOB に定義された一連のエレメント タイプと比較し、VOB 内のエレメント タイプと一致するリストの最初のエレメント タイプを使用してエレメントを作成します。

たとえば、`monet_adm.1` という新規エレメントには、次のようにエレメント タイプが割り当てられます。

- 1 開発者がエレメントを作成します。
`% cleartool mkelem monet_adm.1`
- 2 開発者が (`-eltype` オプションで) エレメント タイプを指定しなかったので、`mkelem` により 1 つ以上のマジック ファイルが使用され、指定された名前のファイル タイプが決定されます。

メモ: ClearCase は、環境変数 `MAGIC_PATH` を使用した検索パス機能をサポートしています。詳細については、`cc.magic` のリファレンス ページを参照してください。

220 ページの「UNIX マジック ファイルの例」にあるマジック ファイルを最初に (またはそれだけを) 使用するとします。この場合、(4) の規則が `monet_adm.1` という名前に最初に一致して、次のファイル タイプのリストが適用されます。

```
manpage src_file text_file file
```

- 3 このリストが、新規エレメントの **VOB** に定義された一連のエレメント タイプと比較されます。**text_file** が既存のエレメント タイプの名前を指定する最初のファイル タイプの場合、**monet_adm.1** は、**text_file** というタイプのエレメントとして作成されます。
- 4 エレメント **monet_adm.1** のバージョンに対するデータの格納と検索は、**text_file** エレメント タイプに関連付けられた、**text_file_delta** という名前のタイプ マネージャによって処理されます。

```
% cleartool describe eltype:text_file
エレメント タイプ "text_file"
...
タイプ マネージャ : text_file_delta
スーパータイプ : file
エレメントのメタタイプ : file element
```

ファイル タイプの作成メカニズムは、ユーザーごと、またはサイトごとに定義されますが、エレメント タイプは **VOB** ごとに定義されます (エレメント タイプが複数の **VOB** 間で確実に整合性を保つように、**ClearCase** 管理者はグローバル タイプを使用できます)。この場合、新規エレメントの **monet_adm.1** は、**text_file** エレメントとして作成されますが、別の一連のエレメント タイプを含む **VOB** 内では、同じマジック ファイルによって **src_file** エレメントとして作成される場合もあります。

エレメント タイプのその他の応用

エレメント タイプによって、タイプ マネージャの選択以外にも、ファイル処理をカスタマイズできます。例を以下に示します。

エレメント タイプを使用したビューの構成

C 言語のヘッダー ファイルのエレメント タイプを **hdr_file** として作成すると、ビューの構成を柔軟に行うことができます。ある開発者がプロジェクトのヘッダー ファイルを再編成したとします。チームの作業を混乱させないように、**header_reorg** というブランチで行ったとします。新しいヘッダー ファイルでコンパイルするために、ほかの開発者は、次の規則を追加して再構成されたビューを使用することができます。

```
element * CHECKEDOUT
element -eltype hdr_file * /main/header_reorg/LATEST
element * /main/LATEST
```

エレメント タイプごとのファイル処理

check_var_names というコーディング標準プログラムを、C 言語の各ソース ファイルに対して実行するとします。このようなファイルすべてが **c_src** というエレメント タイプを持つ場合、1 回の **cleartool** コマンドでこのプログラムを実行することができます。

UNIX:

```
% cleartool find -avobs -visible -element 'eltype(c_src)' ¥  
-exec 'check_var_names $CLEARCASE_PN'
```

Windows:

```
cleartool> find -avobs -visible -element 'eltype(c_src)' ^  
-exec 'check_var_names %CLEARCASE_PN%'
```

定義済みエレメント タイプとユーザー定義のエレメント タイプ

この章で説明するエレメント タイプには、定義済みのもの (たとえば `text_file`) とそれ以外のもの (たとえば `c_src` と `hdr_file`) があります。これまでに説明した例は、これらの名前を持つユーザー定義のエレメント タイプを `mkeltype` コマンドで作成したときのみ正しく機能します。

新しく作成された VOB には、定義済みのエレメント タイプのフルセットが含まれます。各エレメント タイプは、ClearCase が提供するタイプ マネージャの 1 つと関連付けられています。`mkeltype` のリファレンス ページでは、定義済みのエレメント タイプとそのタイプ マネージャについて説明しています。

`mkeltype` で新規エレメント タイプを作成する場合には、既存のエレメント タイプをそのスーパータイプに指定する必要があります。デフォルトでは、新規エレメント タイプはそのスーパータイプと同じタイプ マネージャを使用します。この場合、222 ページの「エレメント タイプのその他の応用」で説明したことをその目的として、新しいタイプと古いタイプを区別することになります。異なるデータ処理を行うには、`-manager` オプションを使用して、スーパータイプとは異なるタイプ マネージャを使用するエレメント タイプを作成します。

定義済みタイプ マネージャとユーザー定義のタイプ マネージャ

ClearCase では、定義済みのタイプ マネージャが提供されます。タイプ マネージャについては、`type_manager` のリファレンス ページに説明があります。各タイプ マネージャは、一連のプログラムとして、`ccase-home-dir/lib/mgrs` のサブディレクトリに実装されます。サブディレクトリの名前は、タイプ マネージャの名前になります。

`mkeltype -manager` コマンドは、既存のタイプ マネージャを使用するエレメント タイプを作成します。ClearCase をさらにカスタマイズするには、新規タイプ マネージャとそれを使用するエレメント タイプを作成します。アーキテクチャ上は、タイプ マネージャは互いに独立していますが、シンボリック リンクを使用して、新規タイプ マネージャに既存のタイプ マネージャの一部の機能を継承することができます。

UNIX: 新規タイプ マネージャの作成

この章の以下の部分では、UNIX 上でタイプ マネージャを作成する方法について説明します。使用する新規タイプ マネージャは、ローカル ネットワーク上でいくつでも作成できます。以下のガイドラインに従ってください。

- 1 新規タイプ マネージャの名前を選択します。データ形式との関係を示す名前をお勧めします (`bitmap_mgr` など)。この名前で `ccase-home-dir/lib/mgrs` のサブディレクトリを作成します。

メモ: ユーザー定義のタイプ マネージャの名前は、アンダースコアから始まらないようにしてください。

- 2 シンボリック リンクを作成して、新規タイプ マネージャが既存のタイプ マネージャからそのメソッドの一部 (ファイル処理操作) を継承できるようにします。
- 3 カスタマイズするメソッドに独自のプログラムを作成します。「UNIX: タイプ マネージャプログラムの作成」を参照してください。
- 4 ネットワーク上の ClearCase または ClearCase LT クライアント ホストでは、新規タイプ マネージャ ディレクトリをコピーするか、新規タイプ マネージャ ディレクトリへのシンボリック リンクを作成します。標準の記憶域、パフォーマンス、信頼性にはトレードオフの関係が適用されます。
- 5 企業で ClearCase MultiSite を使用している場合は、各サイトで手順 4 を反復します。

メモ: エレメント タイプは VOB に属しているため、その VOB をマウントするすべてのホストで利用可能ですが、タイプ マネージャはホスト固有です。タイプ マネージャは、各ホストの `ccase-home-dir/lib/mgrs/manager-name` ディレクトリにあります。

タイプ マネージャの詳細については、`type_manager` の参照ページと、ファイル `ccase-home-dir/lib/mgrs/mgr_info.h` を参照してください。

UNIX: タイプ マネージャ プログラムの作成

タイプ マネージャのメソッドが起動されると、`cleartool` は操作の実行に必要なすべての引数を ASCII 形式でそのメソッドに渡します。たとえば、多くのメソッドが、データの書き込まれるデータ コンテナのパス名を指定する `new_container_name` 引数を取得します。

1 つ以上のパラメータが無視されることも多くあります。たとえば、`create_version` メソッドは、祖先バージョンのデータ コンテナのパス名である `pred_container_name` を渡されます。タイプ マネージャがインクリメンタルな差分を実装している場合、これは必須情報ですが、実装していない場合、祖先のデータ コンテナは必要ありません。

通常、引数はオブジェクト識別子 (Object Identifier、以下 OID) です。OID がどのように生成されるかについては知る必要がありません。各 OID は、エレメント、ブランチ、バージョンのいずれかに対応する一意の名前であることを確認してください。一般的に、同じデータ コンテナに複数のバージョンを格納するタイプ マネージャにのみ OID は必要です。

引数の処理の詳細については、`ccase-home-dir/lib/mgrs/mgr_info.h` (C 言語プログラム用) と `ccase-home-dir/lib/mgrs/mgr_info.sh` (Bourne シェル スクリプト用) を参照してください。

メソッドの終了状態

ユーザーが定義するタイプ マネージャのメソッドは、コマンドがどのように完了したかを示す終了状態を `cleartool` に返す必要があります。`ccase-home-dir/lib/mgrs/mgr_info.sh` のシンボリック定数は、有効なすべての終了状態を指定します。たとえば、`create_version` を呼び出すと、新しいデータ コンテナが作成され、終了状態 `MGR_STORE_KEEP_JUST_NEW` が返されます。新しいデータ コンテナの作成に失敗すると、終了状態 `MGR_STORE_KEEP_JUST_OLD` が返されます。

マニュアル ページのソース ファイルのタイプ マネージャ

マニュアル ページのソース ファイルは、前の表 5 に示されているファイルの 1 つです。このファイルは `nroff(1)` 形式でコード化されています。この種のファイルのタイプ マネージャには、次の特徴があります。

- すべてのバージョンを圧縮された形式で個別のデータ コンテナ内に格納します。これは、`z_whole_copy` タイプ マネージャに類似しています。
- ソース バージョンではなく、フォーマットされたマニュアル ページに `diff` を実行することによるバージョン比較 (`compare` メソッド) を実装しています。

基本的な方針は、`z_whole_copy` タイプ マネージャのメソッドの大部分を使用することです。`compare` メソッドは、`nroff(1)` を使用して、相違を表示する前にバージョンをフォーマットします。

タイプ マネージャ ディレクトリの作成

このタイプ マネージャには、`mp_mgr` (manual page manager) という名前が適切です。最初の手順では、この名前を使用して `ccase-home-dir/lib/mgrs` ディレクトリの下にサブディレクトリを作成します。例を次に示します。

```
# mkdir /usr/rational/lib/mgrs/mp_mgr
```

別のタイプ マネージャからのメソッドの継承

mp_mgr の大部分のメソッドは、z_whole_copy タイプ マネージャからシンボリック リンクを介して継承します。次のコマンドを Bourne シェル内で root ユーザーとして入力します。

```
# MP=$CLEARCASEHOME/lib/mgrs/mp_mgr
# for FILE in create_element create_version construct_version ¥
    create_branch delete_branches_versions ¥
    merge xmerge xcompare get_cont_info
> do
> ln -s ../z_whole_copy/$FILE $MP/$FILE
> done
#
```

新しいタイプ マネージャでサポートしないメソッドは、このリストから省略することができます。シンボリック リンクがないと、Unknown Manager Request エラーが発生します。

以下では、create_version と construct_version という 2 つの継承メソッドについて説明します。これは、ユーザー定義メソッドのモデルになります。実際には、両方とも ccase-home-dir/lib/mgrs/z_whole_copy/Zmgr という同一ファイル内のスクリプトとして実装されています。

create_version メソッド

create_version メソッドは、checkin コマンドが入力されたときに呼び出されます。z_whole_copy タイプ マネージャの create_version メソッドは、次のことを行います。

- 1 チェックアウト バージョン内のデータを圧縮します。
- 2 圧縮データを、ソースの記憶プール内にあるデータ コンテナに格納します。
- 3 呼び出しプロセスに終了状態を返し、新規データ コンテナに対する処理を指示します。

ファイル ccase-home-dir/lib/mgrs/mgr_info.h には、呼び出しプログラム (通常 cleartool または xclearcase) から渡される引数のリストがあります。

```
/*
*****
* create_version
* 新しいバージョンのデータを格納します。
* 指定された新しいコンテナにバージョンのデータを格納し、
* 必要に応じてそれを祖先のデータと結合します (インクリメンタル デルタなどの場合)
*
* コマンド行 :
* create_version create_time new_branch_oid new_ver_oid new_ver_num
*                new_container_pname pred_branch_oid pred_ver_oid
*                pred_ver_num pred_container_pname data_pname
*/
```

特に注意する必要がある引数は、新規データ コンテナのパス名を指定する `new_container_pname` (5 番目の引数) と、チェックアウト ファイルのパス名を指定する `data_pname` (10 番目の引数) だけです。

ファイル `ccase-home-dir/lib/mgrs/mgr_info.sh` には、適切な終了状態のリストと、`create_version` メソッドのシンボリック名があります。

```
# 予期しない値はすべて失敗として処理されます。
MGR_FAILED=1
```

```
# 格納操作の戻り値
MGR_STORE_KEEP_NEITHER=101
MGR_STORE_KEEP_JUST_OLD=102
MGR_STORE_KEEP_JUST_NEW=103
MGR_STORE_KEEP_BOTH=104
.
.
MGR_OP_CREATE_VERSION="create_version"
```

次の例は、`create_version` メソッドを実装するコードです。

```
(1)      shift 1
(2)      if [ -s $4 ] ; then
(3)          echo '$0: error: new file is not of length 0!'
(4)          exit $MGR_FAILED
(5)      fi
(6)      if $gzip < $9 > $4 ; ret=$? ; then : ; fi
(7)      if [ "$ret" = "2" -o "$ret" = "0" ] ; then
(8)          exit $MGR_STORE_KEEP_BOTH
(9)      else
(10)         exit $MGR_FAILED
(11)     fi
```

Bourne シェルでは、コマンド行引数は 9 つまでしか許可されていません。1 行目の `shift 1` では 1 番目の引数 (`create_time`) は必要ないので破棄しています。このため、チェックアウトバージョンのパス名 (`data_pname`) は、元は 10 番目の引数のため、\$9 になります。

6 行目では、`data_pname` の内容が圧縮され、新規の空のデータ コンテナに追加されます。`new_container_pname` は、元は 5 番目の引数ですが、\$4 にシフトしています (2 ~ 5 行目では、新規データ コンテナが空であるかどうかを検証しています)。

最後に、`gzip` コマンドの終了状態がチェックされ、適切な値が返されます (7 ~ 11 行目)。`create_version` メソッドの終了状態は、古いデータ コンテナ (祖先バージョンがあります) と新しいデータ コンテナ (新規バージョンがあります) の両方が維持されているかを示します。

construct_version メソッド

エレメントの `construct_version` メソッドは、標準 UNIX ソフトウェアがエレメントの特定バージョンを読み取ったときに (その内容が既に `cleartext` 記憶プールにキャッシュされていなければ) 呼び出されます。たとえば、エレメント `monet_admin.1` の `construct_version` メソッドは、ユーザーが次のコマンドを入力すると `view_server` によって呼び出されます。

```
% cp monet_admin.1 /usr/tmp                (ビューで選択されているバージョンを  
                                              読み取ります)
```

```
% cat monet_admin.1@@/main/4              (指定されたバージョンを読み取ります)
```

また、`checkout` コマンド中にも呼び出されます。このコマンドは、ブランチ上の最新バージョンのビュープライベート コピーを作成します。

`z_whole_copy` タイプ マネージャの `construct_version` メソッドは、次のことを行います。

- 1 データ コンテナの内容を解凍します。
- 2 呼び出しプロセスに終了状態を返し、新規データ コンテナに対する処理を指示します。

ファイル `ccase-home-dir/lib/mgrs/mgr_info.h` には、このメソッドに渡される引数のリストがあります。

```
/*****  
*****  
* construct_version  
* バージョンのデータを取り出します。  
* 要求されたバージョンのデータを指定したパス名に抽出します。  
* または、ソース コンテナがバージョンの cleartext データとして使用可能であることを示す  
* 値を返します。  
*  
* コマンド行 :  
* construct_version source_container_pname data_pname version_oid
```

ファイル `ccase-home-dir/lib/mgrs/mgr_info.sh` には、適切な終了状態のリストと、`construct_version` メソッドのシンボリック名があります。

```
# 予期しない値はすべて失敗として処理されます。
```

```
MGR_FAILED=1
```

```
# 構築操作の戻り値
```

```
MGR_CONSTRUCT_USE_SRC_CONTAINER=101
```

```
MGR_CONSTRUCT_USE_NEW_FILE=102
```

```
.
```

```
.
```

```
MGR_OP_CONSTRUCT_VERSION="construct_version"
```

次の例は、`construct_version` メソッドを実装するコードです。

construct_version メソッド

```
(1)         if $gzip -d < $1 > $2 ; then
(2)             exit $MGR_CONSTRUCT_USE_NEW_FILE
(3)         else
(4)             exit $MGR_FAILED
(5)         fi
```

1 行目では、`source_container_pname` の内容が解凍され、`data_pname` という `cleartext` コンテナ内に格納されます。残りの行では、`gzip` コマンドの成否に応じて、適切な値を呼び出しプロセスに返します。

新規 compare メソッドの実装

`compare` メソッドは、`cleartool diff` コマンドによって呼び出されます。このメソッドは次のことを行います。

- 1 `nroff(1)` を使用してそれぞれのバージョンをフォーマットし、ASCII テキスト ファイルを作成します。
- 2 `cleardiff` または `xcleardiff` を使用して、フォーマットされたバージョンどうしを比較します。

ファイル `ccase-home-dir/lib/mgrs/mgr_info.h` には、`cleartool` または `xclearcase` からメソッドに渡される引数のリストがあります。

```
/*****
*****
* compare
* 2 つ以上のバージョンのデータを比較します。
* 詳細については、cleartool diff のマニュアル ページを参照してください。
*
* コマンド行 :
* compare [-tiny | -window] [-serial | -diff | -parallel]
* [-columns n]
* [pass-through-options] pname pname ...
```

このリストは、ユーザーによる `compare` メソッドの実装が、`ClearCase diff` コマンドのサポートするコマンド行オプションすべてを受け付ける必要があることを示しています。ここでの方針は、オプションを `cleardiff` に渡すときにはそれを解釈しないことです。すべてのオプションを処理した後に、残りの引数で比較するファイルを指定します。

ファイル `ccase-home-dir/lib/mgrs/mgr_info.sh` には、適切な終了状態のリストと、`compare` メソッドのシンボリック名があります。

```
# COMPARE/MERGE 操作の戻り値
MGR_COMPARE_NODIFFS=0
MGR_COMPARE_DIFF_OR_ERROR=1
.
.
MGR_OP_COMPARE="compare"
```

「compare メソッドのスクリプト」に示される Bourne シェル スクリプトは、compare メソッドを実装します (このスクリプトを変更して、compare と少し異なる xcompare メソッドを実装することができます)。

compare メソッドのスクリプト

```
#!/bin/sh -e
MGRDIR=${CLEARCASEHOME:-/usr/rational}/lib/mgrs

#### メソッドと終了状態を定義するファイルの読み込み
. $MGR_DIR/mgr_info.sh

#### すべてのオプションの処理 : cleardiff へのオプションの受け渡し
OPTS=""
while (expr $1 : '¥-' > /dev/null) ; do
    OPTS="$OPTS $1"
    if [ "$1" = "$MGR_FLAG_COLUMNS" ] ; then
        shift 1
    fi
    shift 1
done
#### 残りのすべての引数 ($*) は比較するファイル
#### 最初に各ファイルを NROFF でフォーマット
COUNT=1
TMP=/usr/tmp/compare.$$
for X in $* ; do
    nroff -man $X | col | ul -Tcrt > $TMP.$COUNT
    COUNT=`expr $COUNT + 1 `
done

#### 次に cleardiff を使用してファイルを比較
cleardiff -quiet $OPTS $TMP.*

#### クリーンアップ処理を行い適切な終了状態を返す
if [ $? -eq MGR_COMPARE_NODIFFS ] ; then
    rm -f $TMP.*
    exit MGR_COMPARE_NODIFFS
else
    rm -f $TMP.*
    exit MGR_COMPARE_DIFF_OR_ERROR
fi
```

タイプ マネージャのテスト

タイプ マネージャのテストは、ClearCase ホスト上でのみ行うことができます。このプロセスを気にする必要はありません。タイプ マネージャは新しい名前を持つため、既存のエレメント タイプや既存のエレメントが自動的にこのタイプ マネージャを使用することはありません。タイプ マネージャを使用するには、新規にエレメント タイプを作成し、そのタイプのテスト用エレメントを作成して、何らかのテストを行います。

以下のテスト シーケンスでは、引き続き `mp_mgr` の例を使用します。

テスト用のエレメント タイプの作成: テストされていないタイプ マネージャが誤って使用されないように、関連付けるエレメント タイプを `manpage_test` にして、自分だけに使用可能にします。

```
% cleartool mkeltype -nc -supertype compressed_file ¥  
-manager mp_mgr manpage_test  
% cleartool lock -nusers $USER eltype:manpage_test
```

テスト用のエレメントの作成と使用: 次のコマンドは、新規タイプ マネージャを使用するテスト用のエレメントを作成し、さまざまなデータ操作メソッドをテストします。

```
% cd directory-in-test-VOB  
% cleartool checkout -nc . (create_element メソッドをテストします)  
% cleartool mkelem -eltype manpage_test -nc -nco test.1  
% cleartool checkout -nc test.1 (construct_version メソッドをテストします)  
% vi test.1 (チェックアウト バージョンを編集します)  
% cleartool checkin -c "first" test.1 (create_version メソッドをテストします)  
% cleartool checkout -nc test.1 (construct_version メソッドをテストします)  
% vi test.1 (チェックアウト バージョンを編集します)  
% cleartool checkin -c "second" test.1 (create_version メソッドをテストします)  
% cleartool diff test.1@@/main/1 test.1@@/main/2 (compare メソッドをテストします)
```

タイプ マネージャのインストールと使用法

タイプ マネージャを十分にテストしたら、次の手順により、それをすべてのユーザーが利用できるようにします。

1 タイプ マネージャをインストールします。

VOB は、ネットワーク上の共有リソースとして、任意の ClearCase ホスト上にマウントすることができます。ただし、タイプ マネージャはホスト リソースなので、ClearCase のクライアント プログラムを実行している各ホスト上にコピーを個別にインストールする必要があります。タイプ マネージャのコピーがインストールされていないと、新規タイプのエレメントは使用できません (VOB かビュー、またはその両方のリポジトリとしてのみ機能しているホストには、タイプ マネージャをインストールする必要はありません)。

VOB のレプリカが作成されている場合は、すべてのサイトにタイプ マネージャをインストールしなければなりません。独自のタイプ マネージャのレプリカは作成されません。

特定のホストにタイプ マネージャをインストールするには、**ccase-home-dir/lib/mgrs** 内にサブディレクトリを作成し、メソッドを実装するプログラムを格納します。ネットワークを介して、サーバー ホスト上のマスター コピーにシンボリック リンクを作成することができます。

2 エレメント タイプを作成します。

このタイプ マネージャを使用する 1 つ以上のエレメント タイプを作成します。「タイプ マネージャのテスト」で説明した手順と同じです (ただし、エレメント タイプの名前に「test」は付けません)。たとえば、エレメント タイプの名前を **manpage** または **nroff_src** とします。

3 既存のエレメントを変換します。

通常は、既存のエレメントの一部で新規タイプ マネージャを使用することになります。**chtype** コマンドを使用すると、エレメント タイプを変更できます。

```
% cleartool chtype -force manpage pathname ...
```

エレメント タイプを変更する権限は、エレメント所有者、VOB 所有者、root ユーザーに制限されています。

4 マジック ファイルを更新します。

特定のエレメントが新規に作成された場合に、この新規エレメント タイプを自動的に適用するには、各ホストの **ccase-home-dir/config/magic** ディレクトリに **local.magic** ファイルを作成 (または更新) します。

```
manpage src_file text_file file: -name "*. [1-9]" ;
```

5 プロジェクト チーム (必要であればほかのチーム) に通知します。

新規エレメント タイプをすべてのチーム メンバーに通知して、新規タイプ マネージャの機能と利点を説明します。新しい機能への (マジック ファイル規則に一致するファイル名を使用した) 自動的なアクセス方法や、(**mkelem -eltype** を使用した) 明示的なアクセス方法について説明します。

GUI ブラウザによるアイコンの使用法

xclearcase ブラウザは、ファイル システム オブジェクトを名前で表示するか、グラフィカルに表示します。後者の場合、**xclearcase** は、各ファイル システム オブジェクトのアイコンを次のようにして選択します。

- 1 オブジェクト名またはその内容によって、ファイル タイプのリストを決定します。
220 ページの「ClearCase によるエレメント タイプの割り当て方法」に説明があります。

- 2 ファイル タイプを、定義済みまたはユーザー定義のアイコン ファイルと 1 つずつ比較します。**cc.icon** のリファレンス ページに説明があります。たとえば、ファイル タイプ **c_source** は、次のアイコン ファイル規則と一致します。

```
c_source : -icon c ;
```

一致するタイプが見つかると、検索は終了します。**-icon** に続くトークンは、表示されるアイコンのファイル名です。

- 3 **xclearcase** は、このファイルを検索します。ファイルは **bitmap(1)** 形式で、**\$HOME/.bitmaps**、**ccase-home-dir/config/ui/bitmaps**、環境変数 **BITMAP_PATH** で指定したディレクトリのいずれかにある必要があります。
- 4 有効なビットマップ ファイルが見つかった場合、**xclearcase** により表示されます。見つからなかった場合には、次のファイル タイプに対してアイコンの検索が継続されます。

アイコン ファイルの名前には、数字の拡張子を付ける必要があります。これは、アイコン ファイル規則に指定する必要はありません。この拡張子には、**xclearcase** でそのアイコンに割り当てる画面スペースを指定します。**ClearCase** に付属するそれぞれのビットマップは、**40 x 40** のアイコンを示す **.40** というサフィックスを持つファイル (たとえば **lib.40**) に格納されています。

以上の手順によって、**xclearcase** ではマニュアル ページのソース ファイルをカスタマイズしたアイコンで表示することができます。すべてのマニュアル ページには、ファイル タイプ **manpage** があります。

- 1 自分のマジック ファイル (**\$HOME/.magic** ディレクトリにあります) に規則を追加して、すべてのマニュアル ページのソース ファイルに割り当てられているファイル タイプ **manpage** を追加します。

```
manpage src_file text_file file: -name "*. [1-9]" ;
```

- 2 自分のアイコン ファイル (**\$HOME/.icon** ディレクトリにあります) に規則を追加して、**manpage** をユーザー定義のビットマップ ファイルにマッピングします。

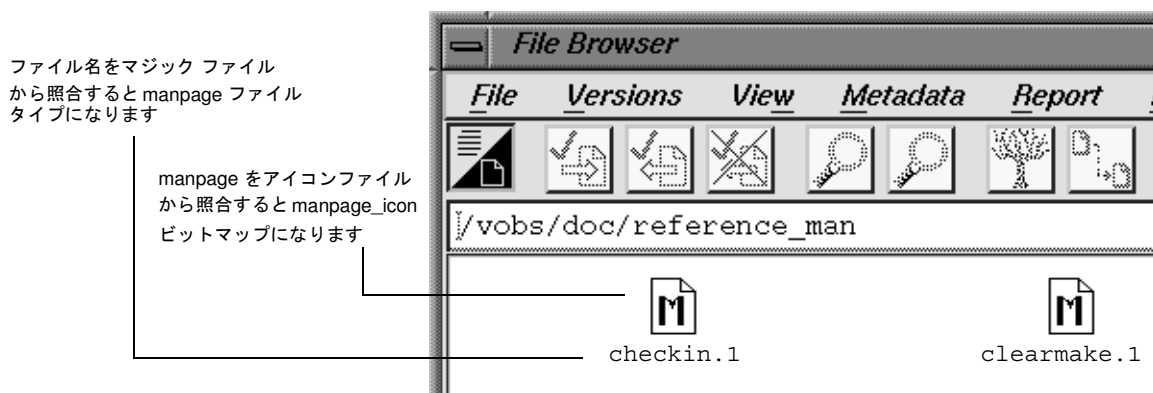
```
manpage : -icon manual_page_icon ;
```

- 3 標準アイコンの 1 つを標準 X ビットマップ ユーティリティで更新して、manpage アイコンを自分のビットマップ ディレクトリ (\$HOME/.bitmaps) に作成します。

```
% mkdir $HOME/.bitmaps
% cd $HOME/.bitmaps
% cp $RATIONALHOME/config/ui/bitmaps/c.40 manual_page_icon.40
% bitmap manual_page_icon.40
```

- 4 xclearcase ブラウザにマニュアル ページのソース ファイルを表示して、作業をテストします (図 49)。

図 49 ユーザー定義アイコンの表示



開発サイクル全体を通じた ClearCase の使用法

16

これまでの章では、ClearCase を使用したプロジェクト管理のさまざまな側面について説明しました。この章では、ClearCase を使用して開発プロジェクト全体の作業を編成する 1 つの方法について説明します。このサイクルに従って、開発者は新しいリリースの作成と以前のリリースの保守を行います。

この章では、一般的な編成作業を行うための概念と方法について説明します。ClearCase には、このほかにもさまざまなアプローチがあります。ここで説明するコマンド行ツールの代わりに、マージ マネージャなどの GUI ツールを使用することもできます。

プロジェクトの概要

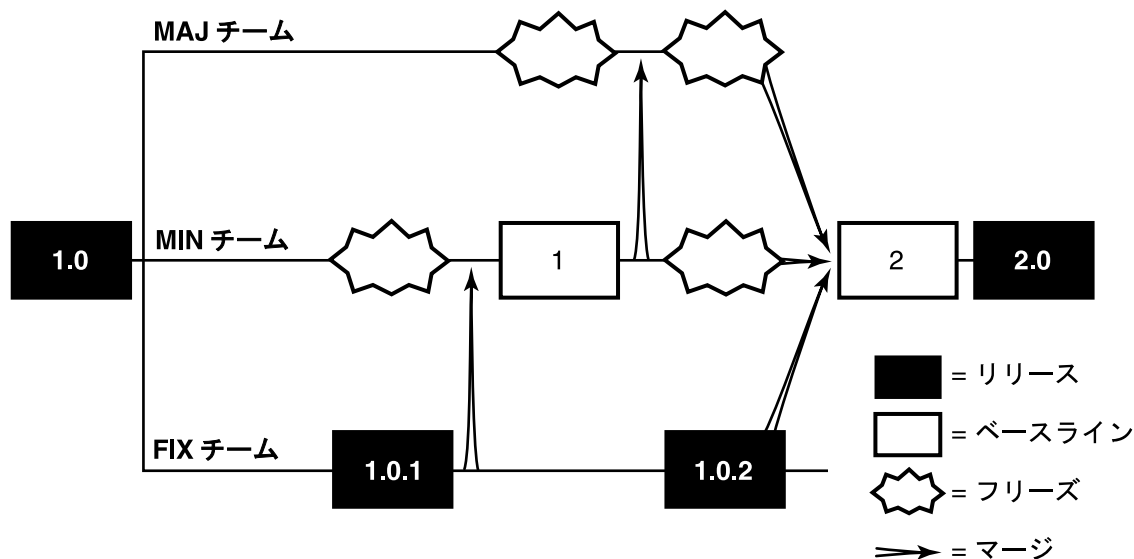
monet プロジェクトの リリース 2.0 の開発には、次のような作業があります。

- パッチ: リリース 1.0 に対する優先度の高いバグ修正が必要です。
- 小規模な機能拡張: 一部のコマンドに新しいオプションを追加する、一部のオプション名を短縮する (-recursive を -r にするなど)、一部のアルゴリズムのパフォーマンスを改善するなどです。
- 重要な新機能の追加: 多くの新規コマンドとインターナショナルライゼーションをサポートするグラフィカル ユーザー インターフェイスが必要です。

これら 3 つの開発作業の大部分を並行して進めることができます (図 50)。ただし、次のような重大な依存関係やマイルストーンを考慮する必要があります。

- リリース 1.0 のいくつかのパッチ リリースは、リリース 2.0 が完成する前に出荷します。
- 新機能の追加は、小規模な機能拡張よりも時間がかかります。
- 一部の新機能は、小規模な機能拡張に依存しています。

図 50 リリース 2.0 開発のプロジェクト計画



この計画では、ベースラインに変更を加えるアプローチを使用します。開発者は、定期的な新規コードの記述を中止し、作業の統合、ビルド、テストを行います。その結果、アプリケーションの安定して動作するバージョンとしてベースラインが作成されます。ClearCase を使用すると、製品の機能拡張をいつでもインクリメンタルに統合することができます。ベースラインを頻繁に作成するほど、作業のマージと結果のテストのタスクは簡単になります。

ベースラインを作成後、開発を再開します。すべての新規作業は、このベースライン ビルドに組み込まれた一連のソース バージョンを使用して開始します。

ベースライン ビルドに組み込まれた、またはベースライン ビルドによって作成されたすべてのバージョンに対して、同じバージョン ラベル (たとえば、リリース 2.0 のベースライン 1 では R2_BL1) を割り当てることで、ベースラインを定義します。

プロジェクト チームは、3 つのより小さなチームに分割され、それぞれ異なる開発作業を担当します。MAJ チーム (新機能の追加)、MIN チーム (小規模な機能拡張)、FIX チーム (リリース 1.0 のバグ修正とパッチ) の 3 つです。

メモ: 一部の開発者は、複数のチームに所属する場合があります。このような開発者は、各チームのタスク用に構成された複数のビュー内で作業を行います。

製品メモ: 次の例で、/vobs/monet などの複数コンポーネントの VOB タグを示す引数は、/monet などの単一コンポーネントの VOB タグのみを認識する UNIX 上の ClearCase LT には適用されません。

monet プロジェクトの開発領域を以下に示します。リリース 2.0 開発の開始時点では、main ブランチ上の最新バージョンには R1.0 というラベルが付いています。

/vobs/monet	(プロジェクトの最上位ディレクトリ)
src/	(ソース)
include/	(インクルード ファイル)
lib/	(共有ライブラリ)

開発方針

ここでは、開発を始める前に ClearCase に関して決定する必要のある内容について説明します。

プロジェクト マネージャーと ClearCase 管理者

ほとんどの開発作業では、プロジェクト マネージャーとシステム管理者の役割は別の人物が行います。プロジェクト マネージャーのユーザー名は **meister** です。管理者のユーザー名は **vobadm** で、monet と libpub の VOB を作成し、所有します。

ブランチの使用法

一般的に、異なる種類の作業は別のブランチ上で行います。たとえば、リリース 1.0 のバグ修正は、新規開発の作業から分離するために別のブランチ上で行います。これにより、FIX チームは、リリース 2.0 の機能拡張や非互換性などの問題にわずらわされることなくパッチ リリースを作成することができます。

MIN チームは、最初のベースライン リリースを独自に作成するので、プロジェクト マネージャーはこのチームに main ブランチを作成します。MAJ チームは、サブブランチ上で新機能を開発し、しばらくの間は統合の準備をしません。FIX チームは、別のサブブランチ上でリリース 1.0 のバグを修正します。この変更はいつでも統合することができます。

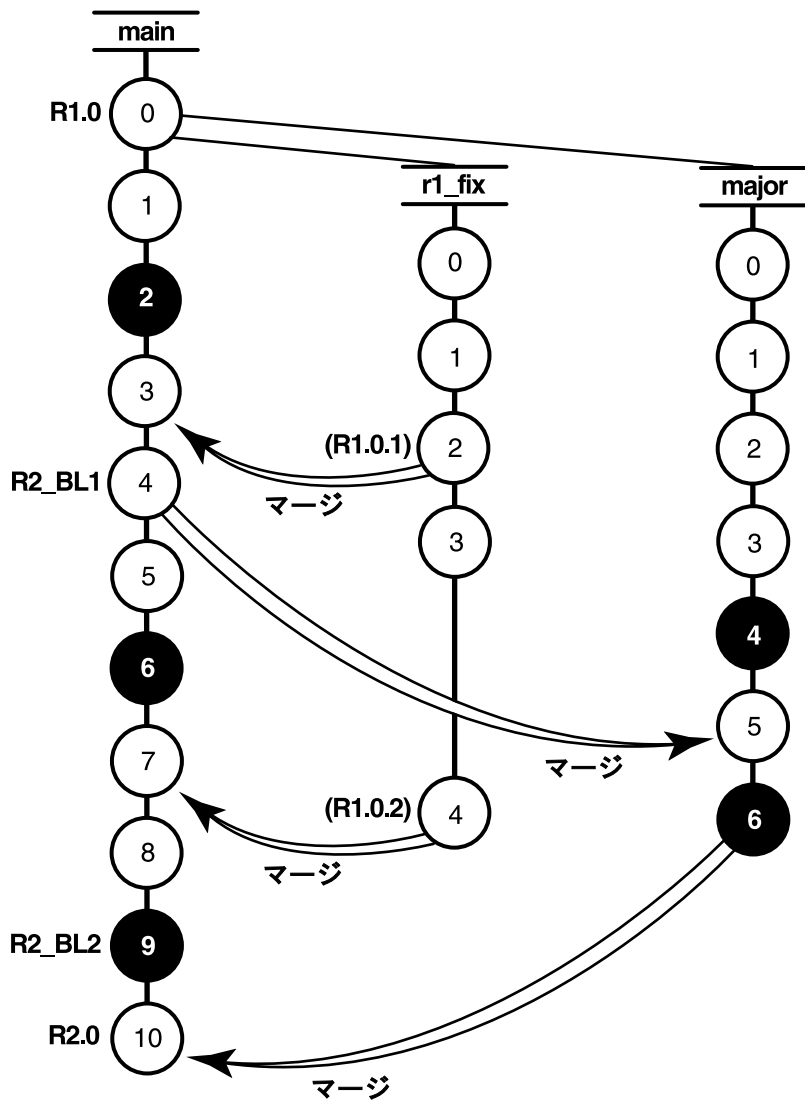
それぞれの新機能は、統合とテストの管理を簡単にするために、独自のサブブランチで開発することも可能です。この章では、説明をわかりやすくするため、新機能の作業は 1 つのブランチで行われると仮定します。

ここでは、プロジェクト マネージャーがエレメントの main ブランチ上のバージョンから最初のベースラインを作成していますが、これは必須ではありません。任意のブランチまたはブランチの組み合わせからリリースを作成することもできます。

リリース 2.0 開発での一般的なエレメントの進展 (図 51) は、以下のようになります。

- 1 R1.0 のバグ修正と並行して、小規模な機能拡張と新機能の開発を開始します (すべてのブランチ)。
- 2 小規模な機能拡張作業をフリーズします (main ブランチ)。
- 3 リリース 1.0.1 から小規模な機能拡張にバグ修正をマージします (main)。
- 4 ベースライン 1 リリースを作成します (main)。
- 5 新機能の開発作業をフリーズします (major)。
- 6 新機能の開発作業にベースライン 1 の変更をマージします (major)。
- 7 小規模な機能拡張作業をフリーズします (main)。
- 8 小規模な機能拡張に追加のバグ修正をマージします (main)。
- 9 新機能の開発作業をフリーズします (major)。
- 10 新機能の開発作業と小規模な機能拡張作業をマージします (main)。
- 11 ベースライン 2 リリースを作成します (main)。
- 12 最終テストを開始します (main)。
- 13 リリース 2.0 の開発作業が完了します (main)。

図 51 開発のマイルストーン：一般的なエレメントの進展



プロジェクト ビューの作成

MAJ チームは、**major** という名前のブランチ上で作業を行い、次の構成仕様を使用します。

- (1) `element * CHECKEDOUT`
- (2) `element * .../major/LATEST`
- (3) `element * R1.0 -mkbranch major`
- (4) `element * /main/LATEST -mkbranch major`

MIN チームは、**main** ブランチ上で作業を行い、デフォルト構成仕様を使用します。

- (1) element * CHECKEDOUT
- (2) element * .../main/LATEST

FIX チームは、**r1_fix** という名前のブランチ上で作業を行い、次の構成仕様を使用します。

- (1) element * CHECKEDOUT
- (2) element * .../r1_fix/LATEST
- (3) element * R1.0 -mkbranch r1_fix
- (4) element * /main/LATEST -mkbranch r1_fix

MAJ チームと FIX チームは、規則 (3) と規則 (4) の自動ブランチ作成機能を利用することで、常にサブブランチを使用します。これにより、開発者はブランチ作成タスクから明示的に解放され、すべてのブランチは必ず **R1.0** というラベルのバージョンで作成されるようになります。

ブランチ タイプの作成

プロジェクト マネージャーは、**major** と **r1_fix** ブランチ タイプを作成します。これは、239 ページの「プロジェクト ビューの作成」の構成仕様に必要です。

```
cleartool mkbtype -c "monet R2 major enhancements" ¥
```

```
major@/vobs/libpub major@/vobs/monet
```

ブランチ タイプ "major" を作成しました。

ブランチ タイプ "major" を作成しました。

```
cleartool mkbtype -c "monet R1 bugfixes" r1_fix@/vobs/libpub r1_fix@/vobs/monet
```

ブランチ タイプ "r1_fix" を作成しました。

ブランチ タイプ "r1_fix" を作成しました。

メモ: それぞれの VOB ごとに独自のブランチ タイプのセットが必要になるため、**monet VOB** と **libpub VOB** では別々にブランチ タイプを作成する必要があります。

標準構成仕様の作成

チーム内のすべての開発者が確実に同じ方法でビューを作成できるように、プロジェクト マネージャーは次のように標準構成仕様を含むファイルを作成します。

- /public/config_specs/MAJ には、MAJ チームの構成仕様が含まれます。
- /public/config_specs/FIX には、FIX チームの構成仕様が含まれます。

これらの構成仕様ファイルは、VOB の外部の標準ディレクトリに格納し、すべての開発者が同じバージョンを確実に取得できるようにします。

ビューの作成、構成、登録

各開発者は、自分のホーム ディレクトリにビューを作成します。たとえば、開発者 **arb** は次のコマンドを入力します。

```
% mkdir $HOME/view_store
% cleartool mkview -tag arb_major $HOME/view_store/arb_major.vws
```

ビューを作成しました。

ホスト上のローカル パス : `phobos:export/home/arb/view_store/arb_major.vws`
グローバル パス :
`/net/phobos/export/home/arb/view_store/arb_major.vws`

権限は以下のとおりです :

ユーザー :	arb	:	rwx
グループ :	user	:	rwx
その他 :		:	r-x

新しいビューには、デフォルト構成仕様が適用されます。このため、MAJ チームと FIX チームの開発者は、チーム用の標準ファイルで自分のビューを再構成する必要があります。**arb** の場合は、**cleartool edcs** コマンドを使用して自分の構成仕様を編集し、既存の行を削除して次の行を追加します。

```
/public/config_specs/MAJ
```

プロジェクト マネージャーが標準ファイルを変更した場合、**arb** はコマンド **cleartool setcs -current** を入力して、その変更を取り込む必要があります。

開発の開始

プロジェクトを開始するには、開発者は適切に構成されたビューを設定し、1 つ以上のエレメントをチェックアウトして作業を開始します。たとえば、MAJ チームの開発者 **david** は次のコマンドを入力します。

```
% cleartool setview david_major
% cd /vobs/monet/src
% cleartool checkout -nc opt.c prs.c
```

ブランチ "major" が "opt.c" バージョン "/main/6" から作成されました。
バージョン "/main/major/0" から "opt.c" をチェックアウトしました。
ブランチ "major" が "prs.c" バージョン "/main/7" から作成されました。
バージョン "/main/major/0" から "prs.c" をチェックアウトしました。

自動ブランチ作成機能によって、各エレメントは **major** ブランチ上にチェックアウトされます (239 ページの「プロジェクト ビューの作成」の MAJ チームの構成仕様にある規則 4 を参照)。MIN チームの開発者がこのコマンドを入力すると、エレメントは **main** ブランチ上にチェックアウトされ、競合は発生しません。

ClearCase は、標準的な開発ツールや作業方法と完全な互換性があります。このため、開発者は、ビュー内での作業中に編集、コンパイル、デバッグ用のツール (個人的なスクリプトや別名も含みます) を自由に使用できます。

開発者は、定期的に作業をチェックインして、ほかのチーム メンバー (そのチームのブランチ上で最新のバージョンを選択するビューを使用しているメンバー) がその作業を利用できるようにします。これにより、開発期間を通して、チーム内での統合とテストが進展します。

各自の作業を隔離するテクニック

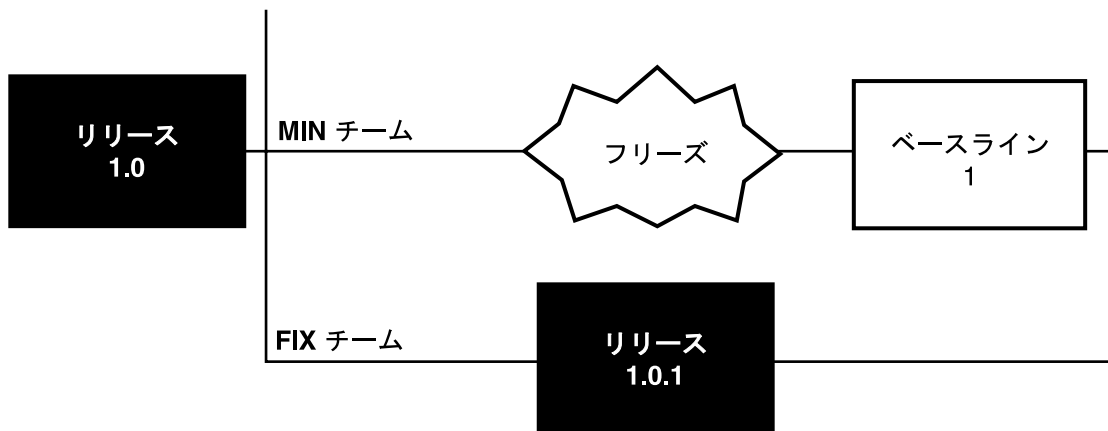
個々の開発者は、各自の作業をほかのチーム メンバーが加えた変更から隔離したい場合があります。これには、次の方法でビューを構成します。

- 時間規則: だれかが互換性のない変更をチェックインした場合、開発者はビューを再構成して、その変更が加えられる前のバージョンを選択することができます。
- プライベート サブブランチ: 開発者は、1 つ以上のエレメントにプライベート サブブランチを作成することができます (たとえば、/main/major/anne_wk)。構成仕様は、/main/major ブランチ上のバージョンの代わりに /main/major/anne_wk 上のバージョンを選択するように変更する必要があります。
- 自分のリビジョンのみの参照: 開発者は、ClearCase のクエリーを使用して、ソース ツリーに自分のリビジョンだけが選択されるビューを構成することができます。

ベースライン 1 の作成

MIN チームは、小規模な機能拡張の最初のグループの実装とテストを行いました。FIX チームは、パッチ リリースを作成し、そのバージョンにラベル R1.0.1 を付けました。次に、これらの作業を組み合わせ、リリース 2.0 のベースライン 1 を作成します (図 52)。

図 52 ベースライン 1 の作成



2 つのブランチのマージ

プロジェクト マネージャーは、MIN 開発者に対して、**r1_fix** ブランチから **R1.0.1** の変更を自分のブランチ (**main**) にマージするように指示します。すべての変更は、次のように **findmerge** コマンドを一度実行することでマージできます。

```
% cleartool findmerge /vobs/libpub /vobs/monet/src ¥  
-fversion .../r1_fix/LATEST -merge -graphical  
.  
. < 省略 >  
.
```

統合とテスト

マージが完了すると、特定のエレメントの **/main/LATEST** バージョンには、MIN チームと FIX チーム両方の作業結果が反映されます。次に、MIN チームは、**monet** アプリケーションのコンパイルとテストを行い、2 チームの作業に互換性の問題がないかをチェックし、あれば修正します。

MIN チームの開発者は、単一の共有ビュー内で変更を統合します。プロジェクト マネージャーは、すべての開発者のホストからアクセスできる場所にビュー記憶領域を作成します。

```
% umask 2  
% mkdir /netwide/public  
% cleartool mkview -tag base1_vu /netwide/public/base1_vu.vws  
ビューを作成しました。  
ホスト上のローカル パス : infinity:/netwide/public/base1_vu.vws  
グローバル パス : /net/infinity/netwide/public/base1_vu.vws.  
権限は以下のとおりです :  
ユーザー : meister : rwx  
グループ : mon : rwx  
その他 : : r-x
```

すべての統合作業は **main** ブランチ上で行われるので、新規ビューの構成を **ClearCase** のデフォルトから変更する必要はありません。MIN 開発者はこのビューを設定して (**cleartool setview base1_vu**)、**monet** アプリケーションのビルドとテストを調整します。開発者は、単一のビューを共有しているので、それぞれのビュープライベート ファイルを上書きしないように注意します。矛盾のある部分 (およびその他のバグ) を修正する新規バージョンを作成すると、そのバージョンは **main** ブランチに追加されます。

ソースへのラベルの関連付け

monet アプリケーションの小規模な機能拡張とバグ修正はこれで統合され、クリーンなビルドが **base1_vu** ビューで行われました。ベースラインを作成するために、プロジェクト マネージャーは、同じバージョン ラベルの **R2_BL1** をすべてのソース エレメントの **/main/LATEST** バージョンに割り当てます。まず、適切なラベル タイプを作成します。

```
% cleartool mklbtype -c "Release, Baseline 1" R2_BL1@/vobs/monet R2_BL1@/vobs/libpub
ラベル タイプ "R2_BL1" を作成しました。
ラベル タイプ "R2_BL1" を作成しました。
```

次に、(プロジェクト マネージャーを除く) すべての開発者が使用できないように、そのラベル タイプをロックします。

```
% cleartool lock -nusers meister lbtype:R2_BL1@/vobs/monet lbtype:R2_BL1@/vobs/libpub
ラベル タイプ "R2_BL1" をロックしました。
ラベル タイプ "R2_BL1" をロックしました。
```

ラベルを適用する前に、プロジェクト マネージャーは、**main** ブランチ上にすべてのエレメントがチェックインされているかを確認します (ほかのブランチ上のチェックアウトはこの時点でも許可されています)。

```
% cleartool lscheckout -all /vobs/monet /vobs/libpub
```

このコマンドからの出力結果がない場合は、**monet** プロジェクトのすべてのエレメントがチェックインされていることになります。次に、プロジェクト マネージャーは、2 つの VOB 内にあるすべてのエレメントの現在選択しているバージョン (/main/LATEST) に、**R2_BL1** ラベルを関連付けます。

```
% cleartool mklabel -recurse R2_BL1 /vobs/monet /vobs/libpub
ラベル "R2_BL1" を "/vobs/monet" バージョン "/main/1" 上に作成しました。
ラベル "R2_BL1" を "/vobs/monet/src" バージョン "/main/3" 上に作成しました。
<省略>
```

インテグレーション ビューの削除

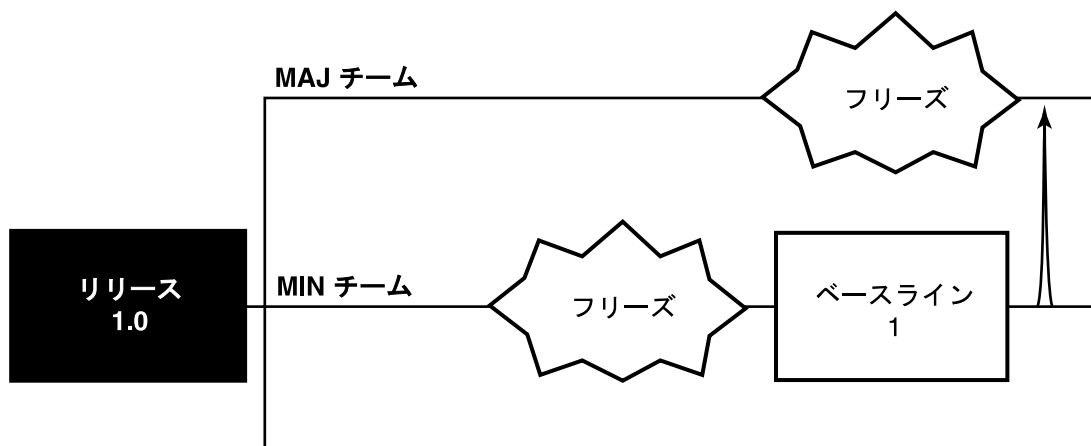
base1_vu として登録されているビューはもう必要ないので、次のようにして削除します。

```
% cleartool rmview -force -tag base1_vu
```

進行中の開発作業のマージ

ベースライン 1 の作成後、MAJ チームは、ベースライン 1 の変更を自分のチームの作業にマージします (図 53)。MAJ チームは、今後の開発に必要な小規模の機能拡張にアクセスできるようになります。また、自分のチームが行った変更と互換性の問題がないかどうかを早期に判断できます。

図 53 新機能の開発の更新



最初に、プロジェクト マネージャーは、新機能の開発におけるコードのフリーズを宣言します。MAJ チームのメンバーは、すべてのエレメントをチェックインして、**monet** アプリケーションのビルドと実行を検証し、必要に応じてソースに変更を加えます。このような変更がすべてチェックインされると、整合性のある一連の /¥main/¥¥major¥¥LATEST バージョンが完成します。

メモ: ほかの新機能開発ブランチで作業をしている開発者も、ここで説明したマージ手順を使用して随時変更をマージすることができます。

マージの準備

- 1 プロジェクト マネージャーは、**major** ブランチ上でチェックアウトされたエレメントがないか確認します。

```
% cleartool lscheckout -all /vobs/monet /vobs/libpub
```

メモ: MAJ チームに、マージしないで作業を継続したいメンバーがいる場合は、「フリーズした」バージョンでサブブランチを作成することができます (または、非予約としてチェックアウトしたバージョンで作業をします)。

- 2 プロジェクト マネージャーは、必要なディレクトリ マージを実行します。

```
% cleartool setview major_vu ( 任意の MAJ チームのビューを使用 )
```

```
% cleartool findmerge /vobs/monet /vobs/libpub -type d ¥
```

```
-fversion /main/LATEST -merge
```

```
/vobs/monet/src のマージが必要 [automatic to /main/major/3 from  
/main/LATEST]
```

```
.  
. <省略>
```

ログが “findmerge.log.04-Feb-99.09:58:25” に書き込まれました。

- 3 ファイルのチェックイン後に、プロジェクト マネージャーは、マージする必要のあるエレメントを決定します。

```
% cleartool findmerge /vobs/monet /vobs/libpub -fversion /main/LATEST -print
```

```
.  
. <省略>
```

```
A 'findmerge' log has been written to  
"findmerge.log.04-Feb-99.10:01:23"
```

この最後の findmerge ログ ファイルは、シェル スクリプトの形式です。この中には、一連の cleartool findmerge コマンドがあり、それぞれが 1 つのエレメントに対して必要なマージを実行します。

```
% cat findmerge.log.04-Feb-99.10:01:23
```

```
cleartool findmerge /vobs/monet/src/opt.c@@/main/major/1 -fver /main/LATEST  
-merge
```

```
cleartool findmerge /vobs/monet/src/prs.c@@/main/major/3 -fver /main/LATEST  
-merge
```

```
.  
.
```

```
cleartool findmerge /vobs/libpub/src/dcanon.c@@/main/major/3 -fver  
/main/LATEST -merge
```

```
cleartool findmerge /vobs/libpub/src/getcwd.c@@/main/major/2 -fver  
/main/LATEST -merge
```

```
cleartool findmerge /vobs/libpub/src/lineseq.c@@/main/major/10 -fver  
/main/LATEST -merge
```

- 4 プロジェクト マネージャーは、major ブランチをロックして、マージを実行する開発者だけが使用できるようにします。

```
cleartool lock -nusers meister,arb,david,sakai brtype:major@/vobs/monet ¥  
brtype:major@/vobs/libpub
```

ブランチ タイプ "major" をロックしました。

ブランチ タイプ "major" をロックしました。

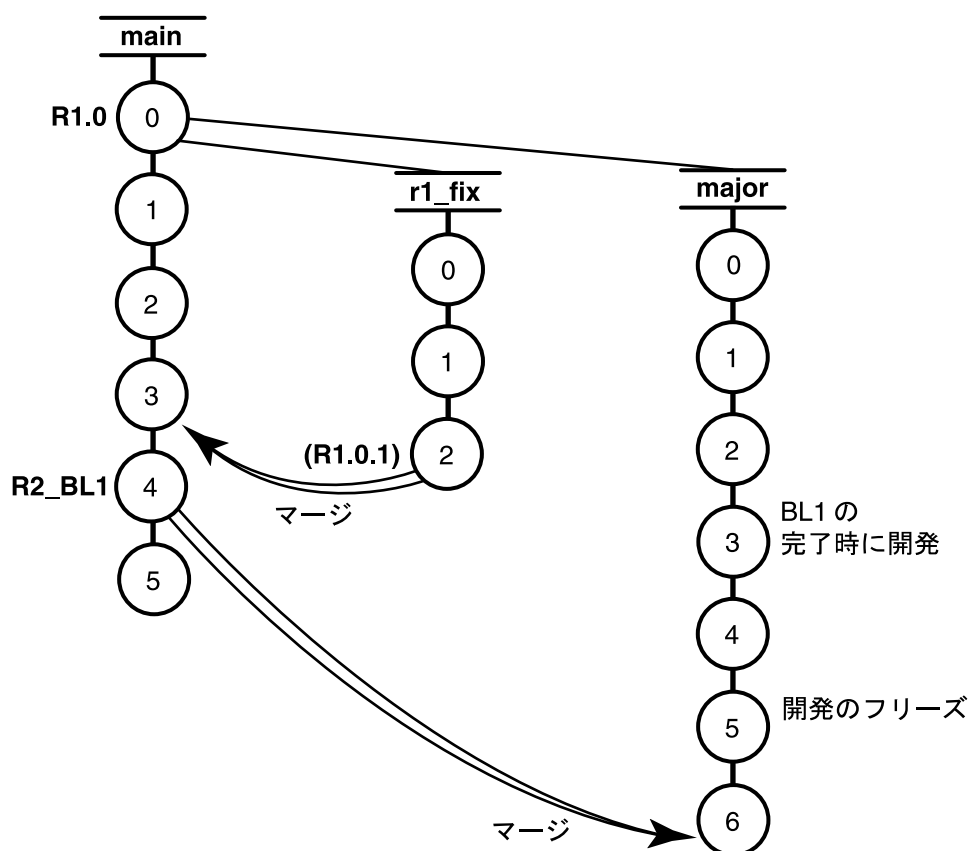
作業のマージ

MAJ チームは、ベースラインに変更内容をすぐには反映できないので、共有ビュー内での作業のマージ (および結果のテスト) は必須ではありません。MAJ 開発者は、自分のビューで作業を継続できます。

プロジェクト マネージャーは、**findmerge** ログからの抜粋を、コマンドを実行して結果を監視している個々の開発者に定期的に送ります (開発者は、マージアクティビティの確認として、結果のログ ファイルをプロジェクト マネージャーに送り返すことができます)。

エレメントのマージされたバージョンには、3 つの開発作業の変更が反映されています。つまり、リリース 1.0 のバグ修正、小規模な機能拡張、新規機能の追加の 3 つです (図 54)。

図 54 ベースライン 1 の変更の major ブランチへのマージ



プロジェクト マネージャーは、**findmerge** コマンドを **-whynot** オプション付きで実行することにより、これ以上マージが必要ないことを確認します。

```
% cleartool findmerge /vobs/monet /vobs/libpub -fversion /main/LATEST -whynot -print
```

```
.  
.
```

"/vobs/monet/src" をマージできません [/main/major/4 は既に /main/3 からマージ済みです]

"/vobs/monet/src/opt.c" をマージできません [/main/major/2 は既に /main/12 からマージ済みです]

```
.  
.
```

プロジェクト マネージャーが **major** ブランチのロックを解除した時点で、マージ作業は終了です。

```
% cleartool unlock brtype:major@/vobs/monet brtype:major@/vobs/libpub
```

ブランチ タイプ "major" をアンロックしました。

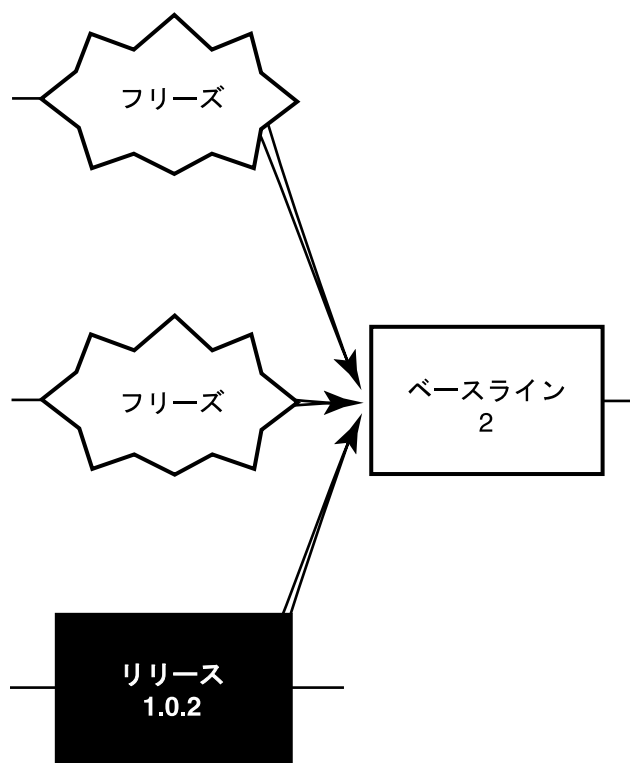
ブランチ タイプ "major" をアンロックしました。

ベースライン 2 の作成

MIN チームはベースライン 2 をフリーズする準備が整っており、MAJ チームの作業もまもなくフリーズします (図 55)。ベースライン 2 は、3 つの開発作業すべてを統合するため、2 度のマージが必要です。

- 最新のパッチ リリースからのバグ修正 (バージョン ラベルは **R1.0.2**) を **main** ブランチにマージします。
- 新機能を **major** ブランチから **main** ブランチにマージします (これは 244 ページの「進行中の開発作業のマージ」で説明したマージと逆方向に行われます)。

図 55 ベースライン 2



ClearCase は、2 方向以上のマージをサポートしているので、バグ修正と新機能の両方を同時に main ブランチにマージできます。ただし、一般的には 2 度のマージ結果を検証の方が簡単です。

r1_fix ブランチからのマージ

1 組目のマージは、243 ページの「2 つのブランチのマージ」で説明したマージとほぼ同じです。

major ブランチからのマージの準備

r1_fix ブランチの統合を完了した後、プロジェクト マネージャーは、major ブランチからのマージを準備します。このマージは厳密に管理された環境下で行います。ベースライン 2 のマイルストーンが間近であり、major ブランチはこれで破棄されるからです。

メモ: アプリケーションのビルドと検証を行い、次のマージに進む前にバージョン ラベルを適用することをお勧めします。

プロジェクト マネージャーは、**main** ブランチと **major** ブランチの両方で、すべてのエレメントがチェックインされているかを確認します。

```
% cleartool lscheckout -brtype main -recurse /vobs/monet /vobs/libpub
% cleartool lscheckout -brtype major -recurse /vobs/monet /vobs/libpub
%
```

このコマンドからの出力結果がない場合は、**main** ブランチと **major** ブランチの両方で、チェックアウトされているエレメントがないことになります。

次に、プロジェクト マネージャーは、マージする必要のあるエレメントを決定します。

```
% cleartool setview minor_vu (任意の MIN チームのビューを使用)
```

```
% cleartool findmerge /vobs/monet /vobs/libpub -fversion .../major/LATEST -print
```

```
.
. < 省略 >
.
```

```
A 'findmerge' log has been written to
"findmerge.log.26-Feb-99.19:18:14"
```

major ブランチ上でのすべての開発は、このベースライン以後停止されます。このため、プロジェクト マネージャーは、マージを実行する以外のすべてのユーザーに対して **major** ブランチをロックします。ロックすると、ClearCase が Merge タイプのハイパーリンクを使用してマージを記録するようになります。

```
% cleartool lock -nusers arb,david brtype:major@/vobs/monet brtype:major@/vobs/libpub
```

ブランチ タイプ "major" をロックしました。

ブランチ タイプ "major" をロックしました。

main ブランチは、少人数の開発者グループがベースライン 2 の統合のために使用するので、プロジェクト マネージャーは、**vobadm** に対して **main** ブランチをほかのすべての人物に対してロックするよう依頼します。

```
% cleartool lock -nusers meister,arb,david,sakai ¥
brtype:main@/vobs/monet brtype:main@/vobs/libpub
```

ブランチ タイプ "main" をロックしました。

ブランチ タイプ "main" をロックしました。

(ブランチをロックできるのは、ブランチ作成者、エレメント所有者、VOB 所有者、root ユーザー (UNIX)、ClearCase 管理者グループ (Windows) のいずれかです。lock のリファレンス ページを参照してください。)

major ブランチからのマージ

main ブランチはマージ先であるため、開発者はデフォルト構成仕様によるビューで作業します。この状況は、245 ページの「マージの準備」で説明した内容に似ています。今回は、マージが major ブランチから main ブランチへと逆方向に行われます。そのため、findmerge コマンドの使用法もよく似ています。

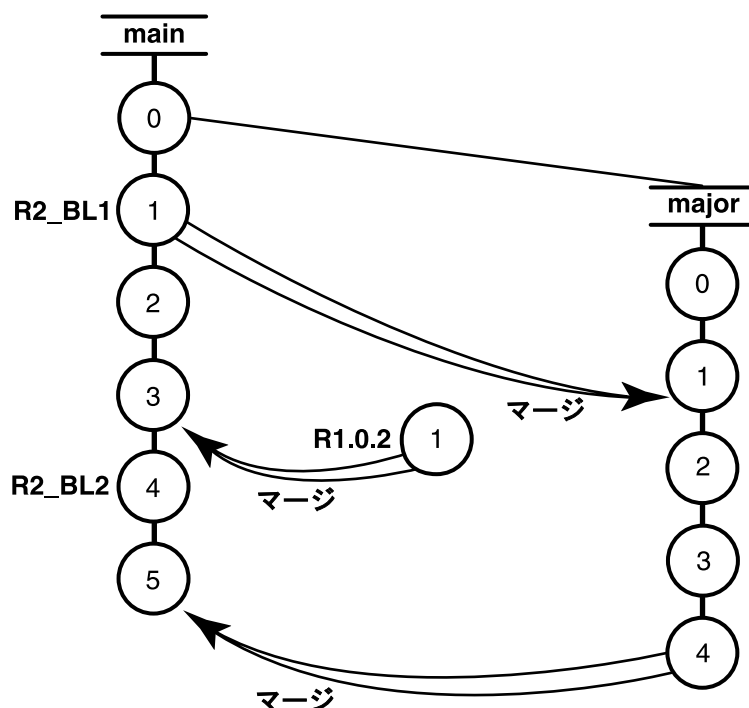
```
% cleartool findmerge /vobs/monet /vobs/libpub -fversion /main/major/LATEST ¥  
-merge -graphical
```

```
.  
.< 省略 >  
.
```

```
A 'findmerge' log has been written to  
"findmerge.log.23-Mar-99.14:11:53"
```

チェックイン後、マージされたエレメントのバージョンツリーは、通常、図 56 のように表示されます。

図 56 ベースライン 2 に先行するマージ後のエレメント構造



major ブランチの破棄

すべてのデータを main ブランチにマージした後、major ブランチの開発者は作業を停止します。プロジェクト マネージャーは、major ブランチを不要にするため、次の手順を実施します。

```
% cleartool lock -replace -obsolete brtype:major@/vobs/monet brtype:major@/vobs/libpub
```

ブランチ タイプ "major" をロックしました。
ブランチ タイプ "major" をロックしました。

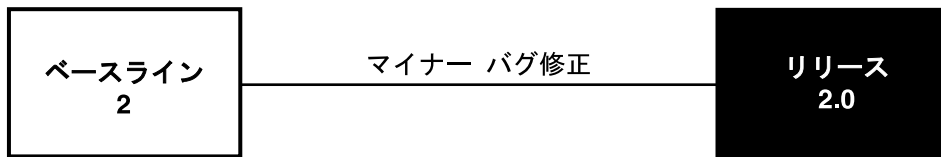
統合とテスト

ベースライン 2 の統合とテストのフェーズは、構造的にベースライン 1 の場合と同様です (243 ページの「統合とテスト」を参照)。統合処理の最後に、プロジェクト マネージャーは、バージョン ラベル R2_BL2 を monet と libpub VOB 内の各要素の /main/LATEST バージョンに関連付けます (ベースライン 1 のバージョン ラベルは R2_BL1 でした)。

最終確認：リリース 2.0 の作成

ベースライン 2 は、社内でリリースされ、その後のテストで小さなバグが発見されました。これらのバグは、main ブランチ上に新規バージョンを作成して修正されました (図 57)。

図 57 最終テストとリリース



顧客に出荷する前に、monet アプリケーションに対して必要な確認フェーズは次のとおりです。

- すべての編集、ビルド、テストは、単一の共有ビュー内に制限します。
- すべてのビルドは、特定のバージョン ラベル (R2.0) のソースを使用して行います。
- このラベルに変更を加える権限があるのは、プロジェクト マネージャーだけです。
- すべてのラベルは、手動で移動する必要があります。
- 優先度の高いバグのみを、次の手順で修正します。
 - a プロジェクト マネージャーは、特定の開発者に新規バージョンを作成する権限を与え、バグの修正を許可します (main ブランチ上)。
 - b 開発者のチェックイン アクティビティは、ClearCase のトリガによって追跡されます。
 - c バグの修正後に、プロジェクト マネージャーは、R2.0 のバージョン ラベルを修正したバージョンに移動し、開発者の新規バージョンを作成する権限を無効にします。

ソースへのラベルの関連付け

デフォルト構成仕様によるビューで、プロジェクト マネージャーは、R2.0 ラベル タイプの作成とロックを行います。

```
cleartool mklbtype -c "Release 2.0" R2.0@/vobs/monet R2.0@/vobs/libpub
```

ラベル タイプ "R2.0" を作成しました。

ラベル タイプ "R2.0" を作成しました。

```
cleartool lock -nusers meister lbtype:R2.0@/vobs/monet lbtype:R2.0@/vobs/libpub
```

ラベル タイプ "R2.0" をロックしました。

ラベル タイプ "R2.0" をロックしました。

プロジェクト マネージャーは、monet と libpub の開発ツリー全体を通して、/main/LATEST バージョンにラベルを添付します。

```
cleartool mklabel -recurse R2.0 /vobs/monet /vobs/libpub
```

<多くのラベル メッセージ>

最終テスト フェーズ中、プロジェクト マネージャーは、新規バージョンが作成された場合、mklabel -replace コマンドを使用してラベルを先に移動します。

main ブランチの使用制限

この時点で、main ブランチの使用は、ベースライン 2 に至るまでのマージと統合を実行した少数のユーザーのみに制限されています (251 ページの「major ブランチからのマージ」を参照)。次に、プロジェクト マネージャーは、vobadm に main ブランチを自分 (meister) 以外のすべてのユーザーに対して使用不可にするよう依頼します。

```
% cleartool lock -replace -nusers meister brtype:main
```

ブランチ タイプ "main" をロックしました。

main ブランチは、最終的なバグ修正の段階にのみ使用されます (254 ページの「最終バグの修正」を参照)。

テスト ビューのセットアップ

プロジェクト マネージャーは、1 つの規則の構成仕様で構成された新規共有ビュー r2_vu を作成します。

```
% umask 2
```

```
% cleartool mkview -tag r2_vu /public/integrate_r2.vws
```

```
% cleartool edcs -tag r2_vu
```

以下が構成仕様です。

```
element * R2.0
```

この構成仕様によって、適切なラベルが添付されたバージョンのみが最終確認ビルドに含まれるようになります。

バグ修正を監視するトリガの設定

プロジェクト マネージャーは、**monet** と **libpub** の VOB 内のすべてのエレメントにトリガを設定します。トリガは、任意のエレメントの新規バージョンがチェックインされると起動します。まず、メールを送信するスクリプトを作成します (スクリプトの例は、190 ページの「関連する変更のチーム メンバーへの通知」を参照してください)。

次に、**vobadm** に対して、**monet** と **libpub** の VOB 内にすべてのエレメントのトリガ タイプを作成し、このスクリプトをトリガ アクションとして指定するよう依頼します。

```
% cleartool mktrtype -nc -element -all -postop checkin -brtype main ¥  
-exec /public/scripts/notify_manager.sh ¥  
r2_checkin@vobs/monet r2_checkin@vobs/libpub  
トリガ タイプ "r2_checkin" を作成しました。  
トリガ タイプ "r2_checkin" を作成しました。
```

トリガ タイプを作成できるのは、VOB 所有者、**root** ユーザー (UNIX)、ClearCase 管理者グループのメンバー (Windows) だけです。

最終バグの修正

ここでは、最終確認環境でのシナリオを説明します。開発者 **arb** は、重大なバグを発見し、それを解決する権限を要求しました。プロジェクト マネージャーは、この開発者に **main** ブランチ上で新規バージョンを作成する権限を与えます。権限を与えるには、**vobadm** に次のコマンドを入力してもらいます。

```
% cleartool lock -replace -nusers arb,meister brtype:main  
ブランチ タイプ "main" をロックしました。
```

arb はデフォルト構成仕様によるビュー内でバグを修正し、そこで修正をテストします。これにより、エレメント **prs.c** の新規バージョンが 2 つと、エレメント **opt.c** の新規バージョンが 1 つ作成されました。**arb** が **checkin** コマンドを使用するたびに、**r2_checkin** トリガによりプロジェクト マネージャーにメールが送られます。例を次に示します。

```
Subject: Checkin /vobs/monet/src/opt.c by arb  
/vobs/monet/src/opt.c@@/main/9  
Checked in by arb.
```

```
Comments:  
fixed bug #459: made buffer larger
```

リグレッションテストによりバグが修正されていることを確認した後に、プロジェクト マネージャーは **arb** の新規バージョンを作成する権限を取り消します。コマンドを実行するのは、この場合も **vobadm** です。

```
% cleartool lock -replace -nusers meister brtype:main  
ブランチ タイプ "main" をロックしました。
```

次に、プロジェクト マネージャーは、メールのメッセージに示されているように、バージョン ラベルを **prs.c** と **opt.c** の新規バージョンに移動します。例を次に示します。

```
% cleartool mklabel -replace R2.0 /vobs/monet/src/opt.c@@/main/9
```

ラベル "R2.0" ("prs.c" 上) をバージョン "/main/8" から "/main/9" に移動しました。

ラベルからの再ビルド

ラベルの移動後に、開発者は再度 **monet** アプリケーションをビルドして、ラベル **R2.0** のバージョンだけを使用して正常にビルドが実行できるか確認します。

まとめ

r2_vu 内での最終ビルドが最終テストに合格したら、**monet** のリリース 2.0 の出荷準備は完了です。**r2_vu** の派生オブジェクトからデリバー メディアを作成した後に、プロジェクト マネージャーは **ClearCase** 管理者に次のリリースのためのクリーンアップと準備を依頼します。

- **ClearCase** 管理者は、すべてのエレメントのトリガ タイプを削除して、すべてのエレメントからチェックイン トリガを削除します。

```
cleartool rmtree trtype:r2_checkin@/vobs/monet trtype:r2_checkin@/vobs/libpub
```

トリガ タイプ "r2_checkin" を削除しました。

トリガ タイプ "r2_checkin" を削除しました。

- **ClearCase** 管理者は、**main** ブランチを再度開きます。

```
cleartool unlock brtype:main
```

ブランチ タイプ "main" をアンロックしました。

ビュー プロファイルから UCM への移行

A

この付録では、ビュー プロファイルの機能を UCM の機能と比較して、プロジェクトをビュー プロファイルから UCM に移行する方法について説明します。

製品ノート: ビュー プロファイルは、Windows 上の Rational ClearCase でのみ利用できます。Rational ClearCase LT はビュー プロファイルをサポートしていません。

ビュー プロファイルと UCM

ベース ClearCase には、ビュー プロファイルと呼ばれる一連の機能があります。ビュー プロファイルを使用すると、開発チームの共有 ClearCase 構成の設定と管理に必要な作業の大部分を自動化することができます。統一変更管理 (UCM) プロセスには、ソフトウェア開発チームを管理するための、より完全なソリューションが用意されています。現在ビュー プロファイルを使用している場合は、UCM に移行することをお勧めします。

機能の比較

ここでは、ビュー プロファイルと UCM の機能を比較します。

ブランチとストリーム

UCM では、ビュー プロファイルの代わりにプロジェクトとそのインテグレーション ストリームが使用されます。ビュー プロファイルの構成仕様は、共通の共有ブランチを選択します。同様に、インテグレーション ストリームに付随するビューは、プロジェクトの共有インテグレーション ブランチを選択するように構成されます。

ビュー プロファイルでは、開発作業用のプライベート ブランチを設定することによって、開発者が独立して作業することができます。UCM では、チームメンバーがプロジェクトに参加したときに、自分用の開発作業空間を作成します。開発作業空間は、開発ストリームと開発ビューから構成されます。

ブランチ間またはストリーム間での作業の移動

ビュー プロファイル内のプライベート ブランチ上で作業を行う場合、ほかの開発者による変更をプライベート ブランチ上に自動的に取り込むことはできません。UCM を利用する開発者は、リベース操作を使用することで、ほかの開発者によってインテグレーション ストリームにデリバーされ、ベースラインに組み込まれた最新の作業状態に基づいて、自分の開発作業空間を更新することができます。

ビュー プロファイルを利用する開発者は、タスクの完了時にプライベート ブランチを終了します。プライベート ブランチを終了すると、そのブランチは閉じ、作業結果はインテグレーション ブランチ上でほかのソースとマージされます。UCM では、開発タスクを完了するために開発者が作成したバージョンが、アクティビティに変更セットとして記録されます。デリバー操作によって、アクティビティが開発ストリームからインテグレーション ストリームに、または機能固有の開発ストリームに移動します。デリバー操作後も開発ストリームは元のままなので、開発者は作業を続けることができます。

VOB とコンポーネント

ビュー プロファイルには、プロジェクト データを保持する VOB のリストがあります。UCM プロジェクトでは、ディレクトリ エLEMENT とファイル エLEMENT がコンポーネントに編成されます。各ストリームには、コンポーネントのリストが保持されます。

チェックポイントとベースライン

ビュー プロファイルでは、ラベル付けされた一連のバージョンであるチェックポイントを使用して、プロジェクトの安定した構成を取得します。UCM では、ベースラインを使用して、コンポーネントごとに一連のバージョンを取得します。

ビュー プロファイルと UCM のさまざまな機能の違いを表 6 にまとめます。

表 6 ビュー プロファイルと UCM のさまざまな機能の比較

ビュー プロファイルの構成要素	UCM の対応要素
ビュー プロファイル	プロジェクトとインテグレーション ストリーム
インテグレーション ブランチ	インテグレーション ストリーム
プライベート ブランチ	開発ストリーム
プライベート ブランチのセットアップ	開発ストリームの作成/プロジェクトに参加
プライベート ブランチの終了	インテグレーション ストリームへの作業のデリバー

表 6 ビュー プロファイルと UCM のさまざまな機能の比較

ビュー プロファイルの構成要素	UCM の対応要素
作業を完了したときにブランチを閉じて、インテグレーション ブランチにマージする	デリバー操作の後でも、開発ストリームは存続する
ほかの開発者の作業結果でプライベート ブランチを自動的に更新できない	リベース操作により、インテグレーション ストリームから個人用作業空間に変更を反映する
ビューはプロファイルからの情報で構成される	ビューはストリームからの情報で構成される

ビュー プロファイル情報を UCM に移行する方法

ここでは、プロジェクトをビュー プロファイルから UCM に移行する際の一般的なガイドラインを示します。

ビュー プロファイル プロジェクトの準備

作業結果を UCM に移行する前に、すべてのプライベート ブランチを終了します。プライベート ブランチ上の作業を UCM プロジェクトに直接移行することはできません。作業結果をインテグレーション ブランチにマージした後で、UCM プロジェクトに移行するすべてのバージョンにラベル付けをしたチェックポイントを作成します。

ビュー プロファイル情報の移行

- 1 ビュー プロファイル プロジェクトの各 VOB をコンポーネントに変換します。
- 2 各コンポーネントごとに、手順 1 で作成したチェックポイントに使用されているラベルをインポートします。ラベルをインポートすることによって、各コンポーネントの新しいベースラインを作成します。
- 3 UCM プロジェクトを作成し、手順 2 で作成した各ベースラインを追加します。

これで、プロジェクト チームのメンバーがプロジェクトに参加して、自分の開発ストリームと開発ビューを作成できます。

UCM プロジェクトの作成の詳細については、「第 6 章 プロジェクトのセットアップ」を参照してください。

ClearCase と ClearQuest の統合

B

ClearCase は、ClearQuest との 2 種類の統合をサポートしています。この付録では、同じ開発環境で両方の統合を管理するために必要な情報について説明します。

ClearCase と ClearQuest の統合の理解

ClearQuest と ClearCase の統合では、1 つ以上の ClearQuest のレコードと、1 つ以上の ClearCase のバージョンを関連付けます。これにより、それぞれの製品の機能を使用することができます。ClearCase は、ClearQuest との統合に 2 つの異なる方法をサポートしています。

- ベース ClearCase と ClearQuest の統合
- UCM と ClearQuest の統合

ベース ClearCase と ClearQuest の統合の詳細については、「第 13 章 ベース ClearCase と ClearQuest の統合の設定」を参照してください。ただし、この統合は UCM プロジェクトには使用できません。

UCM と ClearQuest の統合の詳細については、本書の第一部を参照してください。

一般的に、ベース ClearCase と UCM の統合は別々に行い、共通の ClearQuest ユーザー データベースを使用しないことをお勧めします。ただし、両方の統合に同じ ClearQuest ユーザー データベースを使用することも可能です。この方法は、プロジェクトを UCM に移行中で、ベース ClearCase と ClearQuest の統合で作成された大量の情報が ClearQuest ユーザー データベースに存在する場合には有用です。UCM での新しい作業を、同じ ClearQuest ユーザー データベースの新しい ClearQuest レコードに反映させることが可能です。

この付録の残りの部分では、ベース ClearCase と ClearQuest の統合と、UCM と ClearQuest の統合が共存する場合の管理について説明します。

統合の共存の管理

以前に ClearCase と統合された ClearQuest ユーザー データベースを UCM との統合用に構成する場合、既存の変更セットはそのまま ClearQuest ユーザー データベースに保存されていますが、UCM との統合に移行することはできません。

ClearQuest ユーザー データベース内の既存レコードの変更セットは保存されており、ClearQuest からアクセス可能です。UCM に移行したプロジェクトのタスクを継続するには、対応する UCM アクティビティを新規作成し、そこで作業を継続します。

関連情報については、49 ページの「UCM と ClearQuest の統合の使用法の計画」を参照してください。

スキーマ

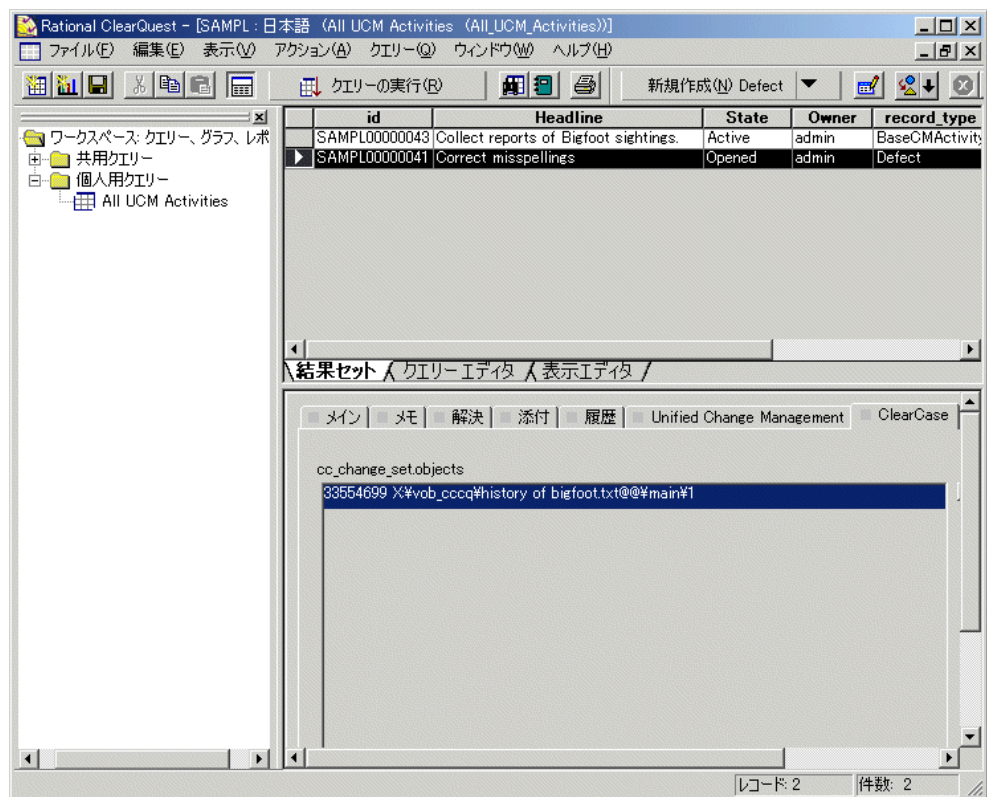
ClearQuest のスキーマは、ベース ClearCase と ClearQuest の統合と、UCM と ClearQuest の統合の両方からの変更を含むことができます。このようなスキーマ内のレコード タイプは、ClearCase パッケージと統一変更管理 (UCM) パッケージの両方を含む場合があります。

このレコード タイプの個々のレコードは、ClearCase または UCM の変更セット情報のいずれかを格納できますが、両方は格納できません。

画面表示

両方の統合で使用するレコード タイプのフォームには、それぞれの統合に関連した変更セット情報を表示する図 58 のような 2 つのタブがあります。[Unified Change Management] タブは、UCM アクティビティの変更セットを表示します。[ClearCase] タブは、ClearQuest レコードに関連する変更セットを表示します。

図 58 ClearQuest GUI 内の変更セット



ClearCase レポートの カスタマイズ



この付録では、ClearCase レポートをカスタマイズする方法について説明します。具体的には、ClearCase レポート プログラミング インターフェイスを紹介し、レポート プロシージャとユーザー インターフェイスをカスタマイズする場合の例を示します。ClearCase レポートは、Windows バージョンの ClearCase と ClearCase LT でのみ利用できます。

ClearCase レポートの仕組み

ClearCase レポートは、以下の 2 つの部分から構成されます。

- ユーザーが変更可能なレポート プロシージャ
- ユーザーが変更できない ClearCase レポート アプリケーション (Report Builder と Report Viewer)

レポート プロシージャは、アプリケーションと密接に関連します。特定のレポートの生成と参照に必要なすべての操作は、レポート プロシージャによって実装されます。アプリケーションは、ユーザーから入力データを受け取り、それを解釈して適切なレポート プロシージャを実行します。実際には、ClearCase レポートは、ClearCase Report Builder と ClearCase Report Viewer の 2 つのアプリケーションに分けて実行されます。Report Builder はレポート パラメータの選択と定義に使用し、Report Viewer は出力レポートの参照に使用します。

すべてのレポート プロシージャには、インターフェイス仕様が必要です。この仕様によって、Report Builder と Report Viewer に表示されるユーザー インターフェイス情報が決定します。ユーザーがフォルダを選択すると、Report Builder によって関連するサブディレクトリ内の各レポートのインターフェイス仕様が走査され、その内容が一時バッファに格納されます。ユーザーが特定のレポートを選択すると、そのレポートに関連するインターフェイス情報がそのバッファから抽出され、Report Builder と Report Viewer 内に表示されます。必要なレポート パラメータを指定すると、Report Builder によってレポートが生成され、そのデータが Report Viewer に送られます。

Report Builder で使用するコマンドに `-i` オプションがあります。このオプションは、レポート プロシージャからインターフェイス仕様を抽出します。レポート プロシージャにインターフェイス仕様が指定されていないか、仕様の構造と内容が Report Builder に適合しない場合、レポート 作成処理は停止します。

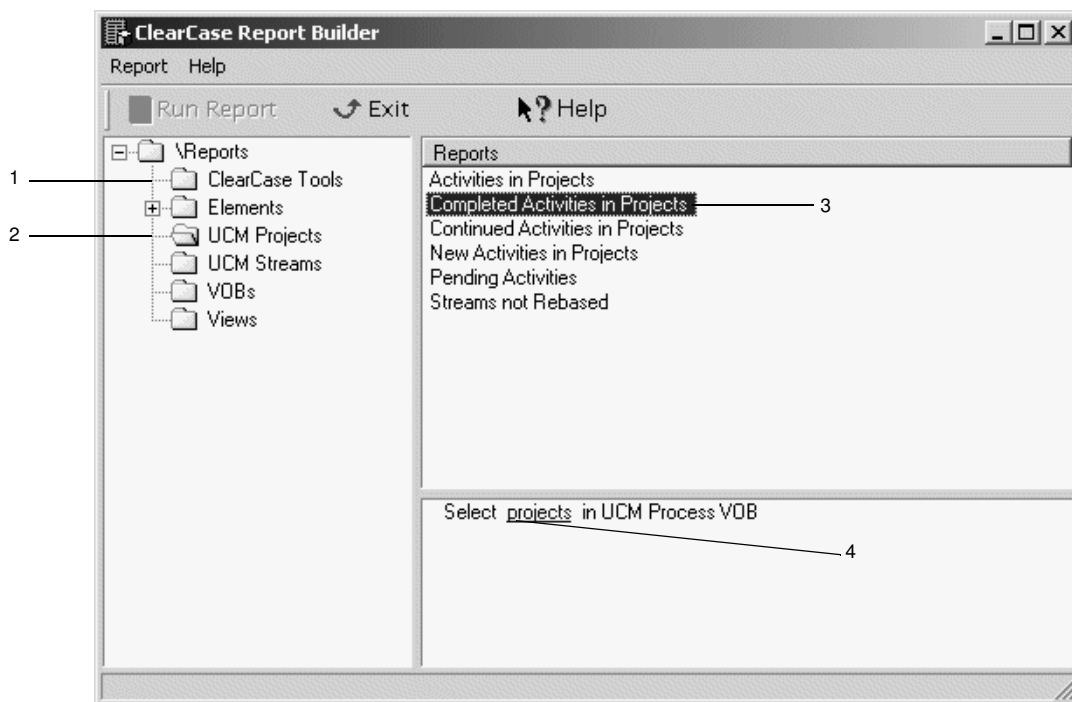
ClearCase レポート アプリケーションとレポート プロシージャ間の処理順序の詳細については、268 ページの「レポート プログラミング インターフェイスの実行時の処理順序」を参照してください。

ClearCase レポートのカスタマイズ可能な内容

ClearCase のプログラミング インターフェイスを通じて、Report Builder ユーザー インターフェイスの 4 つの部分と、Report Viewer ユーザー インターフェイスの 2 つの部分のカスタマイズできます。Report Builder の次の領域で情報を追加、変更、削除することによって、カスタマイズを行います (図 59)。

- 領域 1: ツリー ペインのフォルダの名前
- 領域 2: ツリー ペインに表示されるディレクトリの構造
- 領域 3: レポートの説明
- 領域 4: レポートのパラメータ

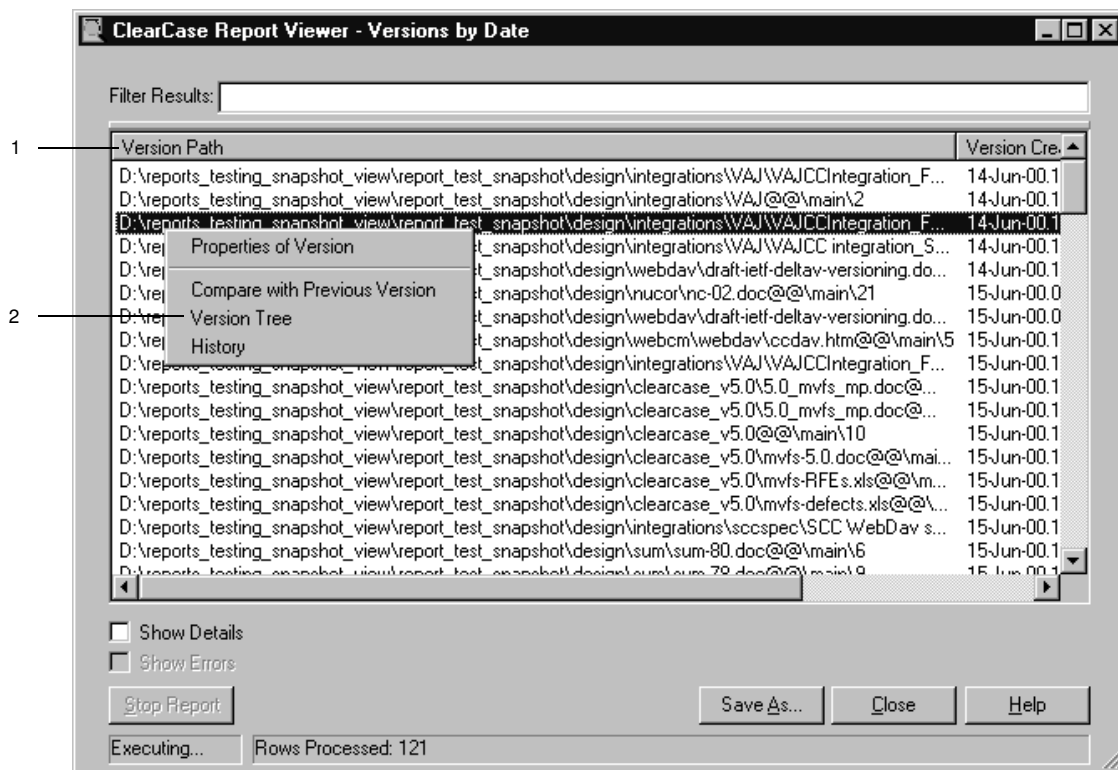
図 59 Report Builder インターフェイスのカスタマイズ可能な領域



Report Viewer の次の領域で情報を追加、変更、削除することによって、カスタマイズを行います (図 60)。

- 領域 1: 列見出しの位置の移動、列見出しの名前の追加、変更、削除、任意の列見出しを対象とするデフォルトのソート順序の追加や削除が可能
- 領域 2: ショートカット メニュー上のコマンド

図 60 Report Viewer ウィンドウ内のカスタマイズ可能なインターフェイス



これらのカスタマイズを行う方法を示すプログラミングの例については、284 ページの「レポートプログラミングの例」を参照してください。

レポート プログラミング インターフェイスの実行時の処理順序

レポート プロシージャのカスタマイズを開始する前に、**Report Builder** と **Report Viewer** の実行時の処理フローを理解することが重要です。処理は 3 つのフェーズで行われます。

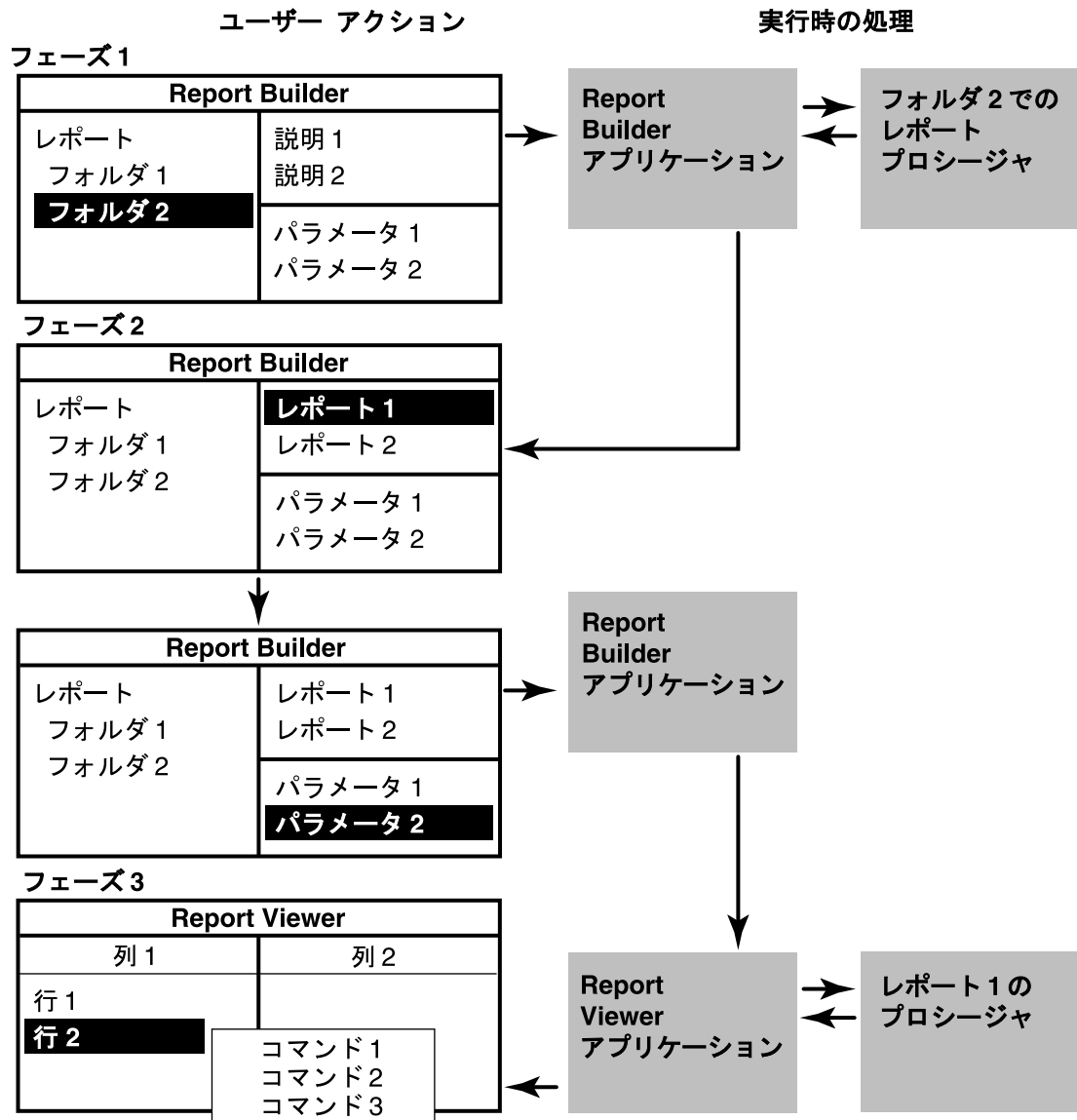
フェーズ 1 で、ユーザーはレポート フォルダのサブフォルダのいずれかを開きます。**Report Builder** は、このサブフォルダ内のレポートに関連するすべてのレポート プロシージャのインターフェイス仕様を処理し、各レポートの説明を **Report Builder** のレポート ペインに表示します。一覧表示された最初のレポートに関連するパラメータが、パラメータ ペインに表示されます。この処理は、**-i** オプションを使用するコマンドで行われます。

フェーズ 2 で、ユーザーはレポート ペインのレポートのいずれかを選択します。そのレポートに必要なパラメータが、**Report Builder** によってパラメータ ペインに表示されます。パラメータをクリックすると、関連するパラメータ セレクタによって、値の入力が要求されます。すべてのパラメータに値を入力すると、ユーザーはレポートの作成を実行することができます (すべてのパラメータの値を指定するまで、**[Run Report]** ボタンは使用可能になりません)。

フェーズ 3 で、レポートが生成されます。インターフェイス仕様内で定義されているパラメータを含むコマンド行が、パラメータ値と共に **Report Viewer** に渡されます。**Report Viewer** はレポート プロシージャを実行し、**cleartool** または **ClearCase Automation Library (CAL)** インターフェイスのいずれかを使用して VOB から情報を取得します。レポート プロシージャは情報を **Report Viewer** に返し、**Report Viewer** は返された情報のソート、フォーマット、表示を行います。レポート内のすべての行で右クリックが (インターフェイス仕様内の定義に応じて) 可能になり、レポート データを操作できるようになります。

図 61 に、この処理順序を示します。

図 61 実行時の処理順序



この処理手順を正しく実行するには、レポートプロシージャが以下の条件を満たしている必要があります。

- レポートプロシージャを格納するディレクトリは、適切な場所にある必要があります。
Report Builder は、`¥reports¥scripts` ディレクトリを参照してレポートプロシージャのファイル名を判定します。関連するディレクトリフォルダをユーザーがクリックすると、そのファイルが呼び出されます。
- レポートプロシージャには、有効なインターフェイス仕様が必要です。適切なフォーマットがない場合、レポートの作成は実行されません。
- レポートプロシージャ内のインターフェイス仕様では、**ClearCase** レポートの所定のパラメータとセクタを使用する必要があります。276 ページの「**ClearCase** レポートのパラメータ」を参照してください。
- レポートプロシージャでは、ユーザー定義のパラメータ値をレポートプロシージャに渡すために、**Report Viewer** で使用可能なコマンド行インターフェイスがサポートされる必要があります。

共有レポートディレクトリの構成

ClearCase をクライアント上にインストールすると、**ClearCase** レポート用のファイルが `ccase-home-dir¥reports` にインストールされます。このディレクトリの内容を変更する前に、共有の場所にそのコピーを作成します。その後で、フォルダの削除または名称変更と、レポートプロシージャの追加または変更を行います。

コピーを作成するには、以下のいずれかを行います。

- ファイルを新しいディレクトリにコピーします。
- ファイルのコピーをソース管理下に置き、共有の場所として機能する **ClearCase** のビューを作成します。

レポートプロシージャのコピーをソース管理下に置くことをお勧めします。

カスタマイズディレクトリから `.dll` と `.exe` ファイルを削除する必要があります。`¥scripts`、`¥script_tools`、`¥scripts_rightclick` のサブディレクトリは必須です。`¥scripts` ディレクトリは、**Report Builder** のツリーペインのルートノード **Reports** になります。このディレクトリツリーは変更することができます。`¥script_tools` と `¥scripts_rightclick` 内のファイルは削除しないことをお勧めします。独自のファイルを追加することはできます。

レポートによって使用されるヘルプファイルは、変更が不可能であり、`¥reports` ディレクトリには含まれません。**ClearCase** レポート用のヘルプファイルは、`ccase-home-dir¥bin¥cc_reports.hlp` にあります。

ソース管理へのレポート プロシージャの追加

ccase-home-dir¥reports のコピーをソース管理下に置くには

- 1 すべてのファイルを一時ディレクトリにコピーします。
- 2 一時ディレクトリで、以下のコマンドを入力します。
`clearexport_ffile -o name-of-data-file`
- 3 既存の VOB 内の ¥reports ディレクトリで、以下のコマンドを入力します。
`clearimport -verbose -directory ¥reports¥name-of-data-file`
- 4 ソース管理下に置かれた ClearCase レポート データ用の動的ビューまたはスナップショットビューを作成します。

カスタマイズしたディレクトリに対する Report Builder の設定

ClearCase レポート用にインストールしたファイルを ccase-home-dir¥reports から共有のディレクトリの場所にコピーした後で、その場所を使用するように Report Builder を設定します。

- 1 Report Builder ウィンドウで、[Report] メニューの [Set Scripts Location] をクリックし、[Configure Reports Directory] ダイアログ ボックスを開きます。
- 2 表示されるダイアログ ボックス内で、以下のいずれかを行います。
 - カスタマイズしたディレクトリのパスをテキスト ボックスに入力します。
 - [...] をクリックして [Browse for scripts location] ダイアログ ボックスを開き、ディレクトリの場所を選択します。

メモ: 変更を有効にするには、ClearCase レポートのユーザー インターフェイスを変更した後で、Report Builder を再起動する必要があります。

ClearCase レポートのデフォルトのディレクトリ構造

ClearCase レポートのすべてのファイルは、ccase-home-dir¥reports にあります。このディレクトリ構造を以下に示します。

```
reports¥
  ccreportbuilder.exe
  ccreportviewer.exe
  cctypechooser.dll
  ccpathchooser.dll
  scripts¥
    ClearCase_Tools¥
      Elements¥
        Attributes¥
        Branches¥
        Labels¥
```

```

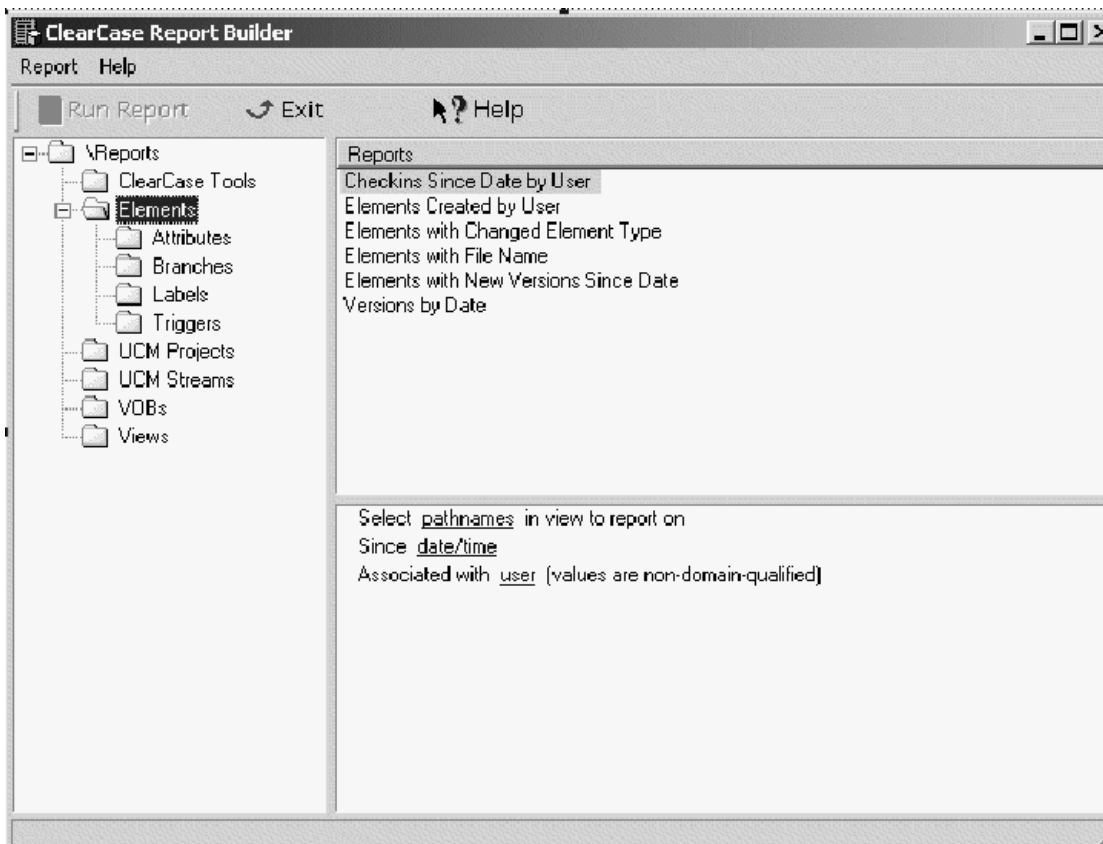
Triggers¥
UCM_Projects¥
UCM_Streams¥
Views¥
VOBs¥
scripts_rightclick¥
script_tools¥

```

Report Builder のツリー ペインへのデータ指定

図 62 に示すように、Report Builder ウィンドウには 3 つのペインがあります。左側はツリー ペイン、右上はレポート ペイン、右下はパラメータ ペインです。ツリー ペインのいずれかのフォルダをクリックすると、関連するレポート プロシージャが Report Builder によってコマンド 行から実行されます。コマンド行の `-i` オプションにより、Report Builder でユーザー インター フェイス情報を収集するための検出アルゴリズムを使用できるようになります。

図 62 Report Builder のユーザー インターフェイス



Report Builder は、¥scripts サブディレクトリ内を走査します。ツリー ペイン内では、ツリーのディレクトリはフォルダとして表示されます。以下のリストに一致する拡張子を持つファイルがあると、レポート ペインに表示されます。

.exe 通常 ClearCase Automation Library (CAL) を使用してデータを抽出する Visual C++ アプリケーション

.pl ユーザーの PATH 環境変数から perl.exe で実行される Perl (ActiveState Perl など)

.prl ccperl

.js Windows スクリプティング ホスト (cscript.exe) で実行される JavaScript

.vbs Windows スクリプティング ホスト (cscript.exe) で実行される VBScript

その他のファイルはすべて無視されます。ClearCase レポートに付随するレポート プロシージャのファイル名拡張子は、.prl です。この拡張子は、Report Builder によって ccperl.exe と関連付けられます。

実行時には、すべてのフォルダ名が表示されます。その際、アンダースコアはスペースに置き換えられ、ファイル名の拡張子は表示されません。ただし、例外としてルートディレクトリは常に Reports と表示されます。この命名規則を変更することはできません。

たとえば、ディスク上に以下のディレクトリ ツリーがあるとします。

```
scripts¥
    Admin_Reports
        view_aging.prl
        all_views.prl
    UCM_Reports¥
        lagging_streams.prl
        completed_acts.prl
```

Report Builder によってツリー ペインに表示されるテキストは以下のようになります。

```
¥Reports
    Admin Reports¥
    UCM Reports¥
```

レポート プロシージャのインターフェイス仕様

カスタマイズされたディレクトリ内でレポート プロシージャを検出すると、Report Builder は各レポート プロシージャのインターフェイス仕様を調べます。この動作は、CreateProcess() で個別のプロセスを開始することにより行います。有効なレポート プロシージャは、インターフェイス仕様を実装しており、この仕様に適合するフォーマット済みのテキストを STDOUT に返します。

```

description : ["<text to display in description pane for this
report>"]

id : <numeric help id>

helpfile : ["<full path to user-written help file for what's this
report help>"]

parameters : [<parameter_spec_1>] [<parameter_spec_2>] ...
[<parameter_spec_N>]

rightclick : [<rightclick_spec_1>] [<rightclick_spec_2>] ...
[<rightclick_spec_N>]

fields : [<field_spec_1>] [<field_spec_2> ... [<field_spec_N>]

```

インターフェイス仕様を処理する際に重大な解析上のエラーが発生すると、レポートはレポートペインに表示されません。**helpfile** 仕様は、将来使用するために予約されており、このリリースでは使用できません。解析エラーのトラブルシューティングについては、306 ページの「トラブルシューティング」を参照してください。

特定のレポート プロシージャ内でインターフェイス仕様がどのように定義されているかを示す例を、以下の項で説明します。

All_Views.prl のインターフェイス仕様

Report Builder は、次のコマンドを使用して All_Views.prl を実行します。

```
ccperl "D:\Program Files\Rational\ClearCase\Reports\scripts\Views\All_Views.prl" -i
```

この場合のインターフェイス仕様は、以下のとおりです。

```

description : "All Views"
id : 2001
helpfile :
parameters :
rightclick : Properties_of_View(single)
fields : "View Tag"(view_tag, rightclick, initial_width 30, sort 1)
"View Owner"(user_dq)

```

このレポート インターフェイスは、Report Viewer を View Tag フィールドと View Owner フィールドに関連付けます。Report Viewer ウィンドウ内で右クリックすると、View Tag フィールドからのデータ ストリームに基づく Properties_of_View.prl が呼び出されます。

description 仕様

description だけがインターフェイス仕様の中で必須の部分になります。**description** のみ定義されている場合、レポート プロシージャは、ほかのグラフィカル ユーザー インターフェイス (たとえば、**clearprompt**) を実行するか、ユーザーと対話します。¥ClearCase_Tools フォルダ内のレポートには、**description** だけが定義されています。

description には、区切り記号である二重引用符 (") 以外の文字を指定できます。長さに制限はありませんが、長い文字列の場合でもレポート ペインでは改行されません。

ヘルプ ID 仕様

Report Builder のユーザー インターフェイスには、**description** フィールドと **parameter** フィールドのためのヘルプ ID が用意されています。**help id** 仕様は、Report Builder のユーザー インターフェイス内の **description** または **parameter** フィールドに関する状況依存のヘルプをサポートするための ID です。この ID は、次の範囲の整数です。

0-999 パラメータに関する状況依存のヘルプ

2000-2999 レポートの説明に関する状況依存のヘルプ

ClearCase レポート用のヘルプ ファイルは、**ccase-home-dir¥bin¥cc_reports.hlp** にあります。**parameter** と **description** フィールドのヘルプ ID は、このファイルに含まれています。このバージョンの ClearCase レポートでは、ユーザー独自のレポート用の ID を追加することはできません。

parameters 仕様

parameters には、ClearCase レポートで提供されているものだけを指定できます。各パラメータには、関連するセレクトア コントロール、パラメータ テキスト、ヘルプ ID があります (表 7)。これらのパラメータを使用する場合、パラメータ名を指定するだけでかまいません。たとえば、**Elements Changed Between Two Labels** レポートには、次の **parameters** 仕様が使用されています。

parameters : LOOKIN LABEL LABEL

パラメータの順序は重要です。パラメータは、指定した順にパラメータ ペインに表示されます (各パラメータはリンクとして表示され、ユーザーがそのリンクをクリックするとパラメータ値の入力を要求されます)。実行時に、Report Viewer はレポート プロシージャを呼び出し、レポート プロシージャは仕様に定義されている順序でパラメータ値を処理します。

表 7 のパラメータ内でタイプ セレクトアに関連するものには、インターフェイス仕様に **LOOKIN** パラメータを指定する必要があります。タイプ セレクトアを使用するその他のパラメータ値を指定する前に、**LOOKIN** パラメータに値を設定します。**LOOKIN** パラメータの値であるパスは、タイプの読み取り先になる一連の VOB を作成するために使用されます。実行時に、これとは違う順序でパラメータを設定しようとすると、以下のエラー メッセージが表示されます。

このパラメータを設定する前に、まず [Select pathnames in view to report on] パラメータの値を設定する必要があります。

表 7 ClearCase レポートのパラメータ (1/2)

パラメータ	パラメータ ペインに表示されるデフォルトのテキスト	ヘルプ ID	セレクト	選択数
PROJECTS	UCM プロセス VOB の projects を選択します	1	パス (UCM)	複数
STREAMS	UCM プロセス VOB の streams を選択します	2	パス (UCM)	複数
ACTIVITIES	UCM プロセス VOB の activities を選択します	3	パス (UCM)	複数
PROJECT	UCM プロセス VOB の project を選択します	4	パス (UCM)	1 つ
STREAM	UCM プロセス VOB の stream を選択します	5	パス (UCM)	1 つ
ACTIVITY	UCM プロセス VOB の activity を選択します	6	パス (UCM)	1 つ
ISTREAM	UCM プロセス VOB の Integration Stream を選択します	7	パス (UCM)	1 つ
PVOB	1 つの Process VOB タグを選択します	8	パス (ファイル選択)	1 つ
COMPONENT	1 つの UCM component オブジェクトセレクトを入力します (検証は実行されません)	9	テキスト	1 つ
BASELEVEL	1 つの UCM baseline オブジェクトセレクトを入力します (検証は実行されません)	10	テキスト	1 つ
ISTREAMS	UCM プロセス VOB の Integration Streams を選択します	11	パス (UCM)	複数
PVOBS	Process VOBs タグを選択します	12	パス (ファイル選択)	複数

表 7 ClearCase レポートのパラメータ (2/2)

パラメータ	パラメータ ペインに表示されるデフォルトのテキスト	ヘルプ ID	セクタ	選択数
COMPONENTS	UCM components オブジェクト セクタのリストを入力します (検証は実行されません)	13	テキスト	複数
BASELEVELS	UCM baselines オブジェクト セクタのリストを入力します (検証は実行されません)	14	テキスト	複数
LOOKIN	レポートするビューの pathnames を選択します	15	パス (ファイル選択)	複数
USER	user との関連付け (値は非ドメイン修飾)	17	テキスト	1 つ
GROUP	group との関連付け (値は非ドメイン修飾)	18	テキスト	1 つ
LABEL	label と	19	タイプ	1 つ
ATTRIBUTE	attribute と	20	タイプ	1 つ
ATTRIBUTE_ VALUE	value for attribute と	21	テキスト	1 つ
TRIGGER	trigger と	22	タイプ	1 つ
BRANCH	branch と	23	タイプ	1 つ
ELTYPE	element type と	24	テキスト	1 つ
HLTYPE	hyperlink type と	25	タイプ	1 つ
CCTIME	date/time から	26	日付/時刻	1 つ
BRANCHLEVELS	integer レベルのブランチ	27	テキスト	1 つ
FILE_NAME	filename と	28	テキスト	1 つ
PATH	pathname を入力	29	テキスト	1 つ
STRING	string と	30	テキスト	1 つ
INTEGER	integer を入力	31	テキスト	1 つ
REGULAR_ EXPRESSION	regular expression を入力	32	テキスト	1 つ

rightclick 仕様

rightclick 仕様は、Report Viewer 内のショートカット メニュー上で利用可能なコマンドのリストです。すべての右クリック イベントは、¥scripts_rightclick ディレクトリ内のスクリプトのリストによってサポートされています。この仕様により、ショートカット メニュー上のテキストを指定することができます。実行時には、次のような文字列のアンダースコアがスペースで置き換えられます。

```
rightclick : properties_of_view delete_view
```

デフォルトの設定では、Report Viewer 内の結果レコードから 1 つまたは複数をコマンドで選択することができます。この動作は、single 変更子を使用して制御可能です。

```
rightclick : properties_of_view(single) delete_view(single)
```

特殊な文字列の sep を使用すると、コマンドのグループ間を視覚的に区切ることができます。ショートカット メニュー上のコマンドは、実行時に指定された順序で表示されます。

fields 仕様

fields 仕様は、フィールドの見出し名と、レポート プロシージャから Report Viewer に返される結果を記述するいくつかの変更子を定義します。サポートされる変更子を表 8 に示します。

表 8 フィールド変更子

変更子	説明
sort N	オプション: 返されるレコードのソート順を指定します。この変更子は、1 で始まる一連の整数を指定する必要があります。この変更子を指定しない場合、レコードの順序はレポート プロシージャから返されたままになります。
Initial_width N	オプション: フィールドのデフォルトの幅をオーバーライドします。
<field_type>	必須
hidden	オプション: Report Viewer 内でこのフィールドの値が表示されないようにします。この変更子を使用する場合、通常はこのフィールドに関連する sort N 変更子が存在します。
rightclick	オプション: Report Viewer 内で右クリック操作が行われた場所に送られるフィールド値ストリームです。rightclick フィールドとして指定できるフィールドは 1 つだけです。

たとえば、以下の **fields** 仕様では、単一のフィールドに最小限の仕様を指定しています。**field_type** 変更子は必須です。

```
fields: "view tag"(view_tag)
```

以下の **fields** 仕様の例では、**view tag** と **last mod time** の 2 つのフィールドにすべての可能な変更子を指定しています。

```
fields: "view tag"(view_tag, rightclick, initial_width 10) "last mod time"(time_t, hidden, sort 1)
```

field_type 規則

表 9 に、**field_type** の名前とそれによって表されるデータの種類を示します。レポートプロシージャにこれらの定義を使用するようお勧めします。ただし、独自の定義を使用することもできます。

ユーザー定義の **field_type** の表示に必要な列幅に応じて、レポートプロシージャの **fields** 仕様で **Initial_width N** 変更子を使用し、列の表示サイズを調整する必要があります。

表 9 ClearCase レポートで指定可能なフィールド タイプ (1/2)

フィールド名	データの説明	例
project	UCM プロジェクトのヘッドライン名	V4.1
project_objsel	UCM プロジェクトのオブジェクトセクタ	Project:v4.1@¥projects
stream	UCM ストリームのヘッドライン名	George_v4.1
stream_objsel	UCM ストリームのオブジェクトセクタ	George_v4.1@¥projects
activity	UCM アクティビティのヘッドライン名	My activity
activity_objsel	UCM アクティビティのオブジェクトセクタ	Activity:my_act@¥projects
view_tag	lsview などから返されるビュー タグ	main_latest_view
time_t	1970 年 1 月 1 日以降の秒数 (整数値)	946934277
cctime	%dfmt_ccase 形式の読み取り可能な時間	20-Dec-99.16:01:12

表 9 ClearCase レポートで指定可能なフィールド タイプ (2/2)

フィールド名	データの説明	例
User	ユーザー名	georgem
User_dq	ドメイン修飾された ユーザー名	rational¥georgem
string	文字列	hello world
Host	ホスト名	georgemnt
Hpath	ビュー /VOB ディレクトリへのローカル コンピュータパス	D:¥ClearCase_Storage¥views¥jet
View_sttrs	ビュー属性	snapshot、ucmview
Element_xpn	@@ で終わるエレメントへのフル パス	S:¥frontpage¥accts¥web¥photo.htm@@
Element_pn	@@ のないエレメントへのフル パス	S:¥frontpage¥accts¥web¥photo.htm
Version_pn	@@ の後ろのバージョン指定子	¥main¥v4.0.bl5_main¥2
label	ラベルのインスタンス名	V4.0
Integer	整数値	5
Yes_no	yes または no 列挙型文字列	Yes
Branch_xpn	ブランチへのフル パス	S:¥frontpage¥accts¥web¥photo.htm@@¥main
version_xpn	バージョンへのフル パス	S:¥frontpage¥accts¥web¥photo.htm@@¥main¥3
branch	ブランチ名	main
Attribute	属性名	normalize_html
Objset	オブジェクトセレクタ	VOB:¥my_vob
Trigger	トリガ名	post_ci
Eltype	エレメント タイプ	text_file
Vob_tag	VOB タグ	¥projects

パラメータ セレクタ

Report Builder のツリー ペインにあるフォルダを開くと、インターフェイス仕様から Report Builder によって読み取られた説明のリストがレポート ペインに表示されます。レポートを選択すると、関連するパラメータが Report Builder に読み取られます。インターフェイス仕様の各パラメータには、関連するパラメータ テキスト、ヘルプ ID、セレクタがあります。すべてのパラメータには、対応するセレクタがあります (表 7)。

ClearCase レポートには、以下のセレクタが用意されています。

- パス セレクタ
- UCM ターゲット セレクタ
- タイプ セレクタ
- 日付/時刻セレクタ
- テキスト セレクタ

ユーザー情報については、各セレクタ ダイアログ ボックス内の [Help] をクリックしてください。

パス セレクタ

パス セレクタは、LOOKIN パラメータに関連付けられます。このセレクタにより、ビュー パス名のリストが表示されて、選択可能になります。選択されたパス名はレポート プロシージャに送られます。このセレクタは、UCM プロジェクト VOB の VOB タグを選択するために、PVOB パラメータと PVOBS パラメータにも使用されます。

UCM ターゲット セレクタ

UCM ターゲット セレクタは、PROJECT、PROJECTS、STREAM、STREAMS、ACTIVITY、ACTIVITIES、ISTREAM、ISTREAMS の各パラメータに関連付けられます。このセレクタにより、UCM オブジェクトが選択可能になります。

タイプ セレクタ

タイプ セレクタは、BRANCH、ATTRIBUTE、LABEL、HYPERLINK、TRIGGER の各パラメータの値を表します。タイプ セレクタがサポートするすべてのパラメータでは、最初に LOOKIN パラメータの値が必要です。

日付/時刻セレクタ

日付/時刻セレクタは、CCTIME パラメータの日付/時刻の値を選択するために使用されます。

テキスト セレクタ

テキスト セレクタは、COMPONENT、COMPONENTS、BASELINE、BASELINES、USER、GROUP、ATTRIBUTE_VALUE、ELTYPE、BRANCHLEVELS、FILE_NAME、PATH、STRING、INTEGER、REGULAR_EXPRESSION の各パラメータの値を表します。

テキスト セレクタに入力されたデータは、Report Builder や Report Viewer で検証または解析されません。パラメータ値を受け取るレポート プロシージャ側で、必要な確認を行う必要があります。

大部分のパラメータでは、テキスト ボックスの上に [Enter value for user] というテキストが表示されます。ベースライン、コンポーネント、エレメント タイプの名前が必要なパラメータの場合、パラメータごとにテキストの表示が変化します。たとえば、[Enter value for baseline] になります。

レポートの表示

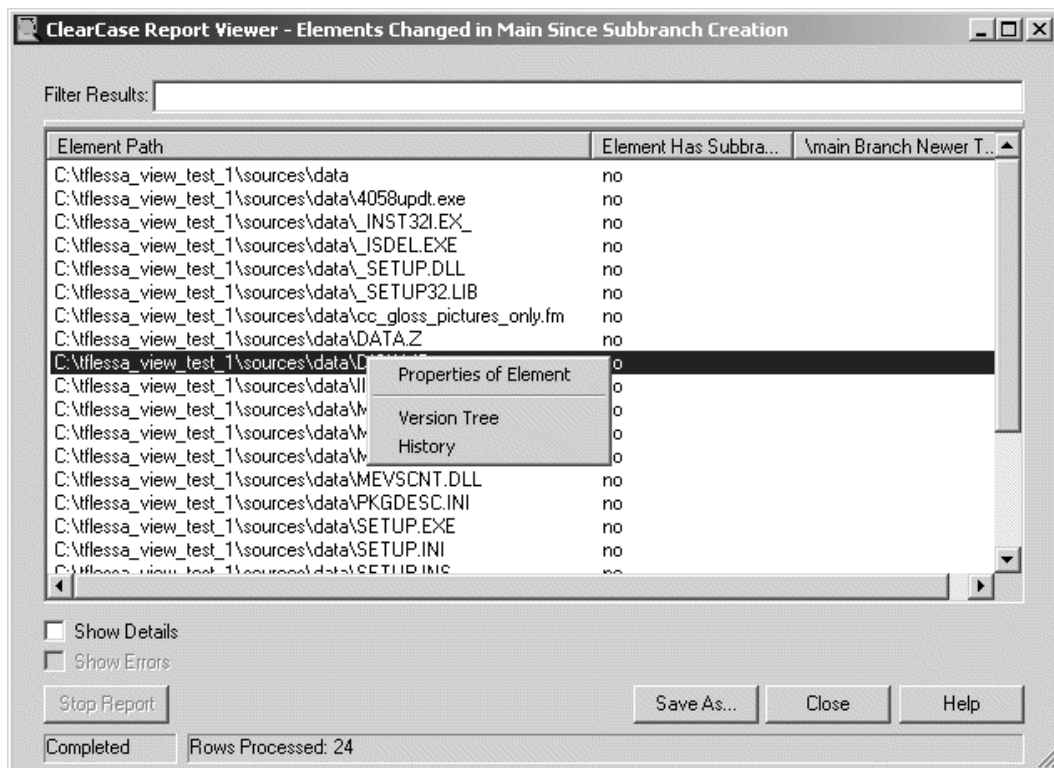
すべてのパラメータに値を指定し、[Run Report] をクリックすると、Report Viewer ウィンドウ (図 63) が表示されます。Report Builder は、ユーザー定義のパラメータをインターフェイス仕様に定義されている順序で引き渡すためのコマンド行を生成します。たとえば、レポート プロシージャがパラメータ LOOKIN LABEL を必要とする場合、Report Viewer は次の値を引き渡します。

```
ccperl elements_with_label.prl %LOOKIN='s:¥frontpage¥acctst';%LABEL=V4.0;
```

Report Viewer は、レポート プロシージャを実行するためのプロセスを生成します。その際、.prl には ccperl.exe を、.pl には perl を、.js と .vbs には cscript.exe を、.exe にはデフォルトの実行プログラムをそれぞれ使用します。レポート プロシージャは、結果を STDOUT に返します。結果はセミコロンで区切られます。その順序、数、タイプは、インターフェイス仕様内の fields 定義で指定されたとおりになります。

レポート プロシージャは、すべてのデータを収集すると、処理を終了します。次に、レポート プロシージャは最も効率のよい方法で、結果のレコードを STDOUT に返します。その結果を Report Viewer がソートし、表示用にフォーマットを整えます。実行時に、ユーザーは、Report Viewer 内の列見出しをクリックしてデフォルトのソート順序を変更することができます。通常のフィールドをソートするには、簡単なテキスト ソートが使用されます。field_type が time_t、integer、cctime であるフィールドは例外で、数値ソートが使用されます。

図 63 Report Viewer ウィンドウ



レポート データの保存

Report Viewer ウィンドウ内で [Save As] をクリックすると、標準のファイル選択ダイアログボックスが表示されます。以下のいずれかの出力フォーマットを選択して、結果を保存します。

- .CSV Access または Excel へのインポート用のカンマ区切り値
- .HTML Web ブラウザでの参照用
- .XML XSL スタイル シートを使用した Internet Explorer 5 での参照用

ファイルの保存は、¥cript_tools にある save_results.prl スクリプトによって実行されます。このスクリプトでは、2 つのスイッチ (-html と -csv) とヘッダーを指定し、その後にセミコロンで区切ったデータ行を続けます。このスクリプトでは、-out オプション用の pathname 値も必要です。この pathname は、Report Viewer がパス セクタから引き渡す値です。

XML 出力は、Report Viewer によって直接サポートされています。save_result.prl を変更することによって、.CSV と .HTML の出力を再実装することができます。XML 出力内で参照可能な追加の XSL スタイル シートを定義することもできます。最初は ClearCase レポート (¥script_tools¥table.xsl) に付属するスタイル シートを使用して、XSL ファイルをカスタマイズすることをお勧めします。

レポート プログラミングの例

ClearCase レポートで提供されるすべてのレポート プロシージャは、ccperl で記述されています。ここでは、それらのレポート プロシージャを変更したものを、プログラミング例として紹介します。レポート プロシージャは、その他の多くのスクリプト言語とプログラミング言語で記述することができます。ほかのプログラミング言語を使用したレポート プロシージャは、ClearCase ユーザー専用 Web サイト (<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp>) にある T0046 パッケージから入手できます (ただし、英語のみのご利用となります)。ここでは、以下のプログラミング例を示します。

- 例 1: Versions_byDate.prl のレポートへの新しい列の追加
- 例 2: Elements_with_New_Versions_Since_Date.prl のディレクトリ構造とレポート説明の変更、別のフィールド名を使用したバージョン パスの変更、レポート出力へのエレメント タイプ列の追加
- 例 3: Elements_Created_by_User.prl のレポート説明、パラメータ タイプ、レポート出力の変更
- 例 4: Element_with_Labels.prl のコマンド順序の変更とショートカット メニューへのコマンドの追加
- 例 5: Element_with_Branches.prl のショートカット メニューへのユーザー定義コマンドの追加

各例に伴うソース コード リストで、### customization change という文字列は、タスクを実行する元のレポートに変更を加えたことを示します。

例 1: レポート出力への列の追加

Versions by Date レポートは、ユーザーの指定したパス名に存在するすべてのバージョンを一覧表示します。このレポート プロシージャには、以下の列が含まれます。

- Version Path (バージョン パス)
- Version Creation Time (バージョン作成時刻)

ここでは、各バージョンに関連するユーザー名を表示する列を追加するように、このレポートを変更します。このレポート プロシージャは、以下の場所にあります。

ccase-home-dir¥Reports¥Scripts¥Elements¥ Versions_by_Date.prl

処理ロジック

Versions_by_Date.prl の処理ロジックは、以下のとおりです。

- 1 この関数の唯一のパラメータである **LOOKIN** パラメータは、次の形式の文字列として取得されます。

```
LOOKIN = "<path1> [<path2> ...]"
```

このパラメータは、**cleartool find** コマンドを呼び出す際に使用するパスのリストを指定します。

- 2 呼び出されたルーチンは、**LOOKIN** の文字列からパスを抽出して、**check_lookin()** ルーチン (**common_script.prl** 内) に渡します。
- 3 **check_lookin()** は、そのパスをグローバル変数の **\$ctfind_paths** に格納し、各パスを二重引用符で囲みます。受け取ったパスの簡単な確認も行います。
- 4 レポート プロシージャは、**cleartool lshistory** を呼び出して、パス パラメータの **\$ctfind_paths** と必要な情報を返すための **-fmt** パラメータを渡します。
- 5 レポート プロシージャは、**print** ステートメントを実行します。その際、インターフェイス仕様の処理中に渡されたリストと同じ数と順序のパラメータ (出力アイテム) を使用します。**Report Builder** には、列見出しを設定するために必要な情報が含まれます。出力を行うには、レポート プロシージャがこの仕様に準拠している必要があります。

インターフェイス仕様

Versions_by_Date.prl の既存のインターフェイス仕様を以下に示します。

```
if (/^-i/) {
    print "description : 'Versions by Date'¥n";
    print "id : 2018¥n";
    print "helpfile :¥n";
    print "parameters : ";
    print "LOOKIN ";
    print "¥n";
    print_version_rightclick();
    print "fields : ";
    print "¥"Version Path¥"(version_xpn, rightclick, sort 2) ";
    print "¥"Version Creation Time¥"(cctime) ";
    print "¥"Version Creation Time¥"(time_t, sort 1, hidden) ";
    print "¥n";
    exit(0);
}
```

必要な変更

レポート出力に列を追加するには

- 1 適切にコーディングした **print** ステートメントを、**Report Builder** が **Report Viewer** に渡すことのできるインターフェイス仕様に追加します。
- 2 **ClearCase** からその情報を取得するために、**cleartool lshist** の呼び出し内の **-fmt** パラメータに **%Fu**; を追加します。
- 3 表示ができるように、**cleartool lshist** の呼び出しから出力が返された後で、ユーザー情報を何らかの変数に適切に抽出します。
- 4 ユーザー変数を出力します。その際、インターフェイス仕様内と順序を同じにして、正しい列見出しの下に表示されるようにします。

変更後のレポート プロシージャ

Versions_by_Date.prl の変更バージョンを以下に示します。このレポート プロシージャは、<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp> (ただし、英語のみのご利用となります) にある **T0046** パッケージの **example1.prl** です。

```
$start_dir = $0; $start_dir =~ s/¥¥scripts¥¥.*/¥¥scripts/;
$common_dir = $start_dir;
$common_dir =~ s/(.*)¥¥scripts/$1¥¥script_tools/;
```

```
$cc = ""; if ($cc) {;}
$ct = ""; if ($ct) {;}
$debug = ""; if ($debug) {;}
$skip_path_checks = ""; if ($skip_path_checks) {;}
$CLEARCASE_XN_SFX = ""; if ($CLEARCASE_XN_SFX) {;}
$ctfind_paths = ""; if ($ctfind_paths) {;}
$skip_path_checks = "yes"; if ($skip_path_checks) {;}
$debug = "no"; if ($debug) {;}
sub do_exit {
    $err = join(" ", @_);
    if ("$err" != "") {
        print STDERR "$err¥n";
    }
    sleep(2);
    if ("$err" != "") {
        exit(1);
    } else {
        exit(0);
    }
}
```

```

open(INCLUDE, "<$common_dir¥¥common_script.prl") or do_exit("error
opening include file '$common_dir¥¥common.prl'");
$buf = "";
while(<INCLUDE>) {
    $buf = $buf . $_;
}
close(INCLUDE);
eval $buf || do_exit("error on eval of include file
'$common_dir¥¥common.prl'");

my $args = $ARGV[0];
$args =~ s/%/ /g;
@args = split(";", $args);
$required_args = 0;
foreach(@args) {

    s/^[ ]+//;
    s/[ ]+$//;
    validate_arg_length($_);
    if (/^-i/) {
        print "description : 'Versions by Date'¥n";
        print "id : 2018¥n";
        print "helpfile :¥n";
        print "parameters : ";
        print "LOOKIN ";
        print "¥n";
        print_version_rightclick();
        print "fields : ";
        print "¥"Version Path¥"(version_xpn, rightclick,
sort 2) ";
        print "¥"Version Creation Time¥"(cctime) ";
        print "¥"Version Creation Time¥"(time_t, sort 1,
hidden) ";
        ### customization change *** added following line
        print "¥"User'(user) ";
        print "¥n";
        exit(0);
    }
    if (/^LOOKIN[ ]*=[ ]*('.*')/) {
        check_lookin($1);
        $required_args++;
        next;
    }
    print STDERR "unrecognized argument: $_¥n";
    print STDERR "  ccperl $0 -i¥n";
    print STDERR "    for script's interface.¥n";
}

```

```

do_exit("¥n");
}
if ($required_args != 1) {
    print STDERR "usage: not all required arguments specified.¥n";
    print STDERR "    ccperl $0 -i¥n";
    print STDERR "        for script's interface.¥n";
    do_exit("¥n");
}
$ENV{"d;"} = "%d;%";
open(CTHIST, "cleartool lshist -fmt '%d;%e;%n¥¥n' -recurse -nco
$ctfind_paths |");
while(<CTHIST>) {
    chomp;
    if (/create directory version/ || /create version/) {
        ($date, $event, $xpn) = split /;/, $_, 3;
        if ($date) {;}
        if ($event) {;}
        if ($xpn) {;}
        $timet = time_to_ticks($date);
### customization change *** added following line
        $user = 'cleartool desc -fmt '%Fu' '$xpn' ';
### customization change *** added ";$user" to following line
        print "$xpn;$date;$timet;$user¥n";
    }
}
do_exit();

```

例 2: レポートのディレクトリ構造、説明、出力の変更

Elements with New Versions Since Date レポートは、ユーザーの指定したパス名と、指定した日時以降の新しいバージョンをすべて一覧表示します。このレポート プロシージャには、以下の列が含まれます。

- Element Path (エレメントパス)
- Version Path (バージョンパス)
- Version Creation Time (バージョン作成時刻)

レポート プロシージャに以下の変更を加えます。

- Report Builder のツリー ペインに、ccase-home-dir¥Reports¥Scripts¥Elements¥New_Versions という名前の新しいディレクトリを表示します。
- Types of Elements with New Versions Since Date という新しいレポート説明を表示します。
- version_xpn フィールドに、異なるフォーマットでバージョンパス情報を表示します。
- Element Type 用の新しい列を表示するために、レポート出力に列を追加します。

このレポートは、ccase-home-dir¥Reports¥Scripts¥Elements¥Elements_with_New_Versions_Since_Date.prl にあります。

処理ロジック

Elements_with_New_Versions_Since_Date.prl の処理ロジックは、以下のとおりです。

- 1 Report Builder がインターフェイス仕様を処理すると、レポート プロシージャから次の 2 つのパラメータが得られます。

LOOKIN

CCTIME

LOOKIN パラメータの機能は、「例 1: レポート出力への列の追加」に説明があります。レポート プロシージャが取得する CCTIME は、次の形式の文字列です。

```
CCTIME = "time"
```

このパラメータは、cleartool find コマンドで使用する時刻を指定します。

- 2 完全修飾されたコマンド行により Report Viewer から呼び出されたレポート プロシージャは、CCTIME 文字列から値を抽出して、chooser_time_to_cctime() サブルーチン (common.prl 内) に渡します。このルーチンは、その文字列を (cleartool に渡すための) 適切なフォーマットに変換して返します。
- 3 レポート プロシージャ は、cleartool find -print コマンドからパイプを開き、変換された cctime の値を created_since(<cctime>) 文字列として渡します。created_since は、query_language(1) の述語であり、通常 find コマンドと組み合わせて使用されます。
- 4 cleartool find から値が返されると、レポート プロシージャはその出力に対して cleartool describe を呼び出し、バージョン作成時刻を取得します。このルーチンは、(common.prl 内の) time_to_ticks() ルーチンを呼び出して、その時刻に相当する秒数を取得します。
- 5 レポート プロシージャは、cleartool find の出力からパスとバージョン ID を取得し、ホスト用の \$CLEARCASE_XN_SFX 拡張命名シンボルの値に基づいて分割します。最後に、レポート プロシージャ は、インターフェイス仕様に定義されている順序で情報を出力します。

インターフェイス仕様

Elements_with_New_Versions_Since_Date.prl の既存のインターフェイス仕様を以下に示します。

```
if (/^-i/) {  
    print "description : 'Elements with New Versions Since  
Date'¥n";  
    print "id : 2017¥n";  
    print "helpfile :¥n";
```

```

        print "parameters : ";
        print "LOOKIN CTIME";
        print "¥n";
        print_element_rightclick();
        print "fields : ";
        print "¥"Element Path¥"(element_pn, sort 2, rightclick) ";
        print "¥"Version Path¥"(version_pn) ";
        print "¥"Version Creation Time¥"(cctime) ";
        print "¥"Version Creation Time¥"(time_t, hidden, sort 1) ";
        print "¥n";
        exit(0);
    }
}

```

必要な変更

ディレクトリ構造とレポート説明を変更、別のフィールド名を使用するようにバージョンパスを修正、レポート出力にエレメントタイプの列を追加するには

- 1 新しいフォルダ **New_Versions** を作成し、レポートプロシージャをそこに移動します。
- 2 以下の処理を行うための適切にコーディングした **print** ステートメントをインターフェイス仕様に追加します。
 - Report Builder 内にレポート説明情報を表示する方法を指定する
 - Report Viewer 内にレポートを表示する方法を指定する
- 3 エレメントタイプに関して必要な情報を取得するために、必要に応じて **cleartool find** の出力に別の処理を追加します。
- 4 エレメントタイプに関する新しい情報を、変数に適切に抽出します。
- 5 新しい情報を正しい列見出しの下に出力します。

変更後のレポート プロシージャ

Elements_with_New_Versions_Since_Date.prl の変更バージョンを以下に示します。
 このレポートプロシージャは、
<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp>
 (ただし、英語のみのご利用となります) にある **T0046** パッケージの **example2.prl** です。

```

$start_dir = $0; $start_dir =~ s/¥¥scripts¥¥.*/¥¥scripts/;
$common_dir = $start_dir;
$common_dir =~ s/(.*)¥¥scripts/$1¥¥script_tools/;

$cc = ""; if ($cc) {;};
$ct = ""; if ($ct) {;};
$debug = ""; if ($debug) {;};
$skip_path_checks = ""; if ($skip_path_checks) {;};

```

```

$CLEARCASE_XN_SFX = ""; if ($CLEARCASE_XN_SFX) {;};
$ctfind_paths = ""; if ($ctfind_paths) {;};
$skip_path_checks = "yes"; if ($skip_path_checks) {;};
$debug = "no"; if ($debug) {;};

sub do_exit {
    $err = join(" ", @_);
    if ("$err" != "") {
        print STDERR "$err¥n";
    }
    sleep(2);
    if ("$err" != "") {
        exit(1);
    } else {
        exit(0);
    }
}

open(INCLUDE, "<$common_dir¥¥common_script.prl") or do_exit("error
opening include file '$common_dir¥¥common.prl'");
$buf = "";
while(<INCLUDE>) {
    $buf = $buf . $_;
}
close(INCLUDE);
eval $buf || do_exit("error on eval of include file
'$common_dir¥¥common.prl'");

my $args = $ARGV[0];
$args =~ s/%/ /g;
@args = split(";", $args);
my $cctime = "";
$required_args = 0;
foreach(@args) {

    s/^[ ]+//;
    s/[ ]+$//;
    validate_arg_length($_);
    if (/^-i/) {
### customization change *** changed following line
        print "description : 'Types of Elements with New Versions
Since
        Date'¥n";
        print "id : 2017¥n";
        print "helpfile :¥n";
        print "parameters : ";
        print "LOOKIN CCTYPE";
    }
}

```

```

        print "¥n";
        print_element_rightclick();
        print "fields : ";
            print "¥"Element Path¥"(element_pn, sort 2,
rightclick) ";
### customization change *** changed following line
            print "¥"Version Path¥"(version_xpn) ";
            print "¥"Version Creation Time¥"(cctime) ";
            print "¥"Version Creation Time¥"(time_t, hidden,
sort 1) ";
### customization change *** added following line
            print "¥"Element Type¥"(eltype) ";
            print "¥n";
            exit(0);
    }

    if (/^LOOKIN[ ]*=[ ]*('.*')/) {
        check_lookin($1);
        $required_args++;
        next;
    }

    if (/^CCTIME[ ]*=[ ]*'*([']*)'*/) {
        $cctime = chooser_time_to_cctime($1);
        $required_args++;
        next;
    }

    print STDERR "unrecognized argument: $_¥n";
    print STDERR "  ccperl $0 -i¥n";
    print STDERR "    for script's interface.¥n";
    do_exit("¥n");
}

if ($required_args != 2) {
    print STDERR "usage: not all required arguments specified.¥n";
    print STDERR "  ccperl $0 -i¥n";
    print STDERR "    for script's interface.¥n";
    do_exit("¥n");
}

open(CTFIND, "cleartool find $ctfind_paths -version
'created_since($cctime)' -print |");
while(<CTFIND>) {
    chomp;
    if (/CHECKEDOUT/) {next;}
    $vertime = 'cleartool desc -fmt '%d' '$_' '
### customization change *** added following line
    $eltype = 'cleartool desc -fmt '%[type]p' '$_' '
    $vertime_t = time_to_ticks($vertime);
    ($path, $verid) = split $CLEARCASE_XN_SFX, $_, 2;

```

```
### customization change *** changed following line
    print "$_;$verid;$vertime;$vertime_t;$eltype¥n";
    #print "$path;$verid;$vertime;$vertime_t¥n";
}
do_exit();
```

例 3: レポート説明、パラメータ タイプ、レポート出力の変更

Elements Created by User レポートは、ユーザー定義のユーザー名によって作成されたすべてのエレメントを一覧表示します。このレポート プロシージャには、以下の列が含まれます。

- Element Path (エレメントパス)
- Creating User (作成ユーザー)

このレポートに以下の変更を加えます。

- 新しいレポート説明の **Elements with Group** を表示します。
- 既存のユーザー パラメータを削除し、グループ用の新しいパラメータを追加します。
- エレメントに関連付けられているグループを、ユーザー定義のグループ パラメータ内に指定されているグループと比較します。
- レポート出力に **Group** と **Yes/No** の列を追加します。**Yes/No** 列は、各エレメントに関連するグループが、ユーザー定義のグループ パラメータの値と同じであるかどうかを比較した結果を反映します。

このスクリプトは、以下の場所にあります。

ccase-home-dir¥Reports¥Scripts¥Elements¥Elements_Created_by_User.prl

処理ロジック

Elements_Created_by_User.prl の処理ロジックは、以下のとおりです。

- 1 Report Builder がインターフェイス仕様を処理すると、レポート プロシージャから次の 2 つのパラメータが得られます。

LOOKIN

USER

LOOKIN パラメータの機能は、284 ページの「例 1: レポート出力への列の追加」に説明があります。レポート プロシージャが取得する USER は、次の形式の文字列です。

USER= "user-name"

このパラメータは、cleartool サブコマンドで使用されるユーザー名を指定します。

- 2 USER 文字列は、\$ccuser として抽出され、格納されます。その後、created_by(\$ccuser) に渡されます。

- 3 **created_by (\$ccuser)** クエリー言語プリミティブは、**cleartool find** に指定されたパスをフィルタし、述語に一致するものだけを返します。ここでは、**USER** のパラメータ値を設定することで、該当ユーザーが作成したものだけが返されます。
- 4 ユーザー変数を出力します。その際、インターフェイス仕様内と順序を同じにして、正しい列見出しの下に表示されるようにします。

インターフェイス仕様

Elements_Created_by_User.prl の既存のインターフェイス仕様を以下に示します。

```
if (/^-i/) {
    print "description : 'Elements Created by User'¥n";
    print "id : 2016¥n";
    print "helpfile :¥n";
    print "parameters : ";
    print "LOOKIN USER";
    print "¥n";
    print_element_rightclick();
    print "fields : ";
    print "¥"Element Path¥"(element_xpn, sort 2, rightclick) ";
    print "¥"Creating User¥"(user, sort 1) ";
    print "¥n";
    exit(0);
}
```

必要な変更

ユーザー パラメータを削除、グループと日付/時刻用のパラメータを追加、グループと日付/時刻の情報に関するレポート出力を調整するには

- 1 必要なインターフェイスの変更に応じて、レポート プロシージャのインターフェイス仕様を変更します。
- 2 グループ情報に関するデータ要求を処理するように、レポート プロシージャ内のロジックを変更します。ClearCase からグループ情報を取得するために、**cleartool describe** の呼び出し内の **-fmt** パラメータに **%Gu;** を追加します。
- 3 表示ができるように、**cleartool describe** の呼び出しから出力が返された後で、グループ情報を変数に適切に抽出します。
- 4 エレメントのグループが、ユーザーによって入力されたグループ パラメータ値と同じであるかどうかを判定して、その比較結果を列見出しとして出力します。
- 5 グループ変数を出力します。その際、インターフェイス仕様内と順序を同じにして、正しい列見出しの下に表示されるようにします。

変更後のレポート プロシージャ

Elements_Created_by_User.prl の変更バージョンを以下に示します。このレポート プロシージャは、<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp> (ただし、英語のみのご利用となります) にある T0046 パッケージの example3.prl です。

```
$start_dir = $0; $start_dir =~ s/¥¥scripts¥¥.*¥¥scripts/;
$common_dir = $start_dir;
$common_dir =~ s/(.*)¥¥scripts/$1¥¥script_tools/;

$cc = ""; if ($cc) {;};
$ct = ""; if ($ct) {;};
$debug = ""; if ($debug) {;};
$skip_path_checks = ""; if ($skip_path_checks) {;};
$CLEARCASE_XN_SFX = ""; if ($CLEARCASE_XN_SFX) {;};
$ctfind_paths = ""; if ($ctfind_paths) {;};
$skip_path_checks = "yes"; if ($skip_path_checks) {;};
$debug = "no"; if ($debug) {;};

sub do_exit {
    $err = join(" ", @_);
    if ("$err" != "") {
        print STDERR "$err¥n";
    }
    sleep(2);
    if ("$err" != "") {
        exit(1);
    } else {
        exit(0);
    }
}

open(INCLUDE, "<$common_dir¥¥common_script.prl") or do_exit("error
opening include file '$common_dir¥¥common.prl'");
$buf = "";
while(<INCLUDE>) {
    $buf = $buf . $_;
}
close(INCLUDE);
eval $buf || do_exit("error on eval of include file
'$common_dir¥¥common.prl'");

my $args = $ARGV[0];
$args =~ s/%/ /g;
@args = split(";", $args);
my $ccuser = "";
$required_args = 0;
```

```

foreach(@args) {

    s/^[ ]+//;
    s/[ ]+$//;
    validate_arg_length($_);
    if (/^-i/) {
### customization change *** changed following line
        print "description : 'Elements With Group'¥n";
        print "id : 2016¥n";
        print "helpfile :¥n";
        print "parameters : ";
### customization change *** changed following line
        print "LOOKIN GROUP";
        print "¥n";
        print_element_rightclick();
        print "fields : ";
        print "¥"Element Path¥"(element_xpn, sort 2,
rightclick) ";
### customization change *** added following 2 lines
        print "¥"Element's Group¥"(group, sort 1) ";
        print "¥"Same¥"(yes_no) ";
### customization change *** deleted following line
        #print "¥"Creating User¥"(user, sort 1) ";
        print "¥n";
        exit(0);
    }
    if (/^LOOKIN[ ]*=[ ]*('.*')/) {
        check_lookin($1);
        $required_args++;
        next;
    }
### customization change *** deleted following 2 lines
    #if (/^USER[ ]*=[¥t]*¥"*(^¥")*¥"*/) {
        #bccuser = $1;
### customization change *** added following 2 lines
    if (/^GROUP[ ]*=[¥t]*¥"*(^¥")*¥"*/) {
        $ccgroup = $1;
        $required_args++;
### customization change *** deleted following line
        #validate_user($bccuser);
        next;
    }
    print STDERR "unrecognized argument: $_¥n";
    print STDERR "    ccperl $0 -i¥n";
    print STDERR "        for script's interface.¥n";
    do_exit("¥n");
}

```

```

}
if ($required_args != 2) {
    print STDERR "usage: not all required arguments specified.¥n";
    print STDERR "  ccperl $0 -i¥n";
    print STDERR "      for script's interface.¥n";
    do_exit("¥n");
}
### customization change *** deleted following 3 lines
#if ($ccuser =~ /[ ]+\/) {
#     do_clearprompt("cleartool find does not allow spaces in user
names;
#     cannot proceed.");
#}

### customization change *** changed following line
open(CTFIND, "cleartool find $ctfind_paths -nxname -print |");
while(<CTFIND>) {
    chomp;
### customization change *** added following 6 lines
    $grp = 'cleartool desc -fmt '%Gu' '$_' ';
    if ($grp eq $ccgroup) {
        $same = "yes";
    } else {
        $same = "no";
    }
}
### customization change *** changed following line
print "$_;$grp;$same¥n";
#print "$_;$ccuser;¥n";
}
do_exit();

```

例 4: 右クリック操作のショートカットメニューの変更

Elements with Labels レポートは、ユーザー定義のパス名に関するラベル付きのすべてのエレメントを一覧表示します。このレポートには、以下の列が含まれます。

- Element Path (エレメントパス)

ここでは、ショートカットメニューに[前バージョンと比較]コマンドを追加するように、このレポートを変更します。現在、ショートカットメニューには以下のコマンドが組み込まれています。

- Properties of Element
- Version Tree
- History

このレポート プロシージャは、ccase-home-dir¥Reports¥Scripts¥Elements¥Labels¥Elements_with_Labels.prl にあります。

インターフェイス仕様

Elements_with_Labels.prl の既存のインターフェイス仕様を以下に示します。

```
if (/^-i/) {
    print "description : ";
    print "'Elements with Labels'";
    print "¥n";
    print "id : 2003¥n";
    print "helpfile :¥n";
    print "parameters : ";
    print "LOOKIN ";
    print "LABEL ";
    print "¥n";
    print_element_rightclick();
    print "fields : ";
    print "¥"Element Path¥"(element_pn, rightclick, sort 1)";
    print "¥n";
    exit(0);
}
```

インターフェイス仕様にある `print_element_rightclick()` への呼び出しルーチンのコードは、¥script_tools¥common.prl 内にあります。

```
sub print_element_rightclick {
    print "rightclick : ";
    print "Properties_of_Element(single) ";
    print "sep ";
    print "Version_Tree(single) ";
    print "History(single) ";
    print "¥n";
}
```

必要な変更

レポート プロシージャでは、メインのソート キーに同じフィールドを使用するすべてのレポートには、ショートカット メニューに同じコマンドを組み込むという規則が適用されます。たとえば、`element` または `element_xpn` をメインのソート キーとするすべてのレポートでは、同じコマンドが表示されます。

メインのソート キーが `element` または `element_xpn` のすべてのレポートで追加のコマンドを使用可能にするには、ClearCase レポートに付属する ¥script_rightclick 内のルーチンを変更し、¥script_tools¥common.prl 内の関連するルーチンを編集します。

レポートプロシージャを変更するには、(`¥script_tools¥common.prl` にある) `sub print_element_rightclick` の内容をコピーし、インターフェイス仕様の適切な部分に貼り付けます。次に、新しいコマンドを表示するための宣言を追加します。

変更後のレポート プロシージャ

`Elements_with_Labels.prl` の変更バージョンを以下に示します。このレポート プロシージャは、<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp> (ただし、英語のみのご利用となります) にある T0046 パッケージの `example4.prl` です。

```
$start_dir = $0; $start_dir =~ s/¥¥scripts¥¥.*¥¥scripts//;
$common_dir = $start_dir;
$common_dir =~ s/(.*)¥¥scripts/$1¥¥script_tools/;

$cc = ""; if ($cc) {;}
$ct = ""; if ($ct) {;}
$debug = ""; if ($debug) {;}
$skip_path_checks = ""; if ($skip_path_checks) {;}
$CLEARCASE_XN_SFX = ""; if ($CLEARCASE_XN_SFX) {;}
$ctfind_paths = ""; if ($ctfind_paths) {;}
$skip_path_checks = "yes"; if ($skip_path_checks) {;}
$debug = "no"; if ($debug) {;}

sub do_exit {
    $err = join(" ", @_);
    if ("$err" != "") {
        print STDERR "$err¥n";
    }
    sleep(2);
    if ("$err" != "") {
        exit(1);
    } else {
        exit(0);
    }
}

open(INCLUDE, "<$common_dir¥¥common_script.prl") or do_exit("error
opening include file '$common_dir¥¥common.prl'");
$buf = "";
while(<INCLUDE>) {
    $buf = $buf . $_;
}
close(INCLUDE);
eval $buf || do_exit("error on eval of include file
'$common_dir¥¥common.prl'");

my $args = $ARGV[0];
```

```

$args =~ s/%/ /g;
@args = split(";", $args);
my $cclabel = "";
$required_args = 0;
foreach(@args) {
    s/^[ ]+//;
    s/[ ]+$//;
    validate_arg_length($_);
    if (/^-i/) {
        print "description : ";
        print "'Elements with Labels'";
        print "¥n";
        print "id : 2003¥n";
        print "helpfile :¥n";
        print "parameters : ";
        print "LOOKIN ";
        print "LABEL ";
        print "¥n";
    }
    ### customization change *** deleted following line
    #print_element_rightclick();
    ### customization change *** added following 7 lines
    print "rightclick : ";
    print "Properties_of_Element(single) ";
    print "sep ";
    print "Compare_with_Previous_Version(single) ";
    print "Version_Tree(single) ";
    print "History(single) ";
    print "¥n";
    print "fields : ";
    print "¥"Element Path¥"(element_pn, rightclick, sort
1)";
    print "¥n";
    exit(0);
}
if (/^LOOKIN[ ]*=[ ]*('.*')/) {
    #print "paths are $1¥n";
    check_lookin($1);
    $required_args++;
    next;
}
if (/^LABEL[ ]*=[ ]*'([^']*')'/) {
    $cclabel = $1;
    #print "label is $cclabel¥n";
    $required_args++;
    next;
}
}

```

```

print STDERR "unrecognized argument: $_\n";
print STDERR "  ccperl $0 -i\n";
print STDERR "    for script's interface.\n";
do_exit("\n");
}
if ($required_args != 2) {
    print STDERR "usage: not all required arguments specified.\n";
    print STDERR "  ccperl $0 -i\n";
    print STDERR "    for script's interface.\n";
    do_exit("\n");
}
open(CTFIND, "cleartool find $ctfind_paths -element
'lbtype_sub($cclabel)' -print |");
while(<CTFIND>) {
    chomp;
    ($path, $rest) = split $CLEARCASE_XN_SFX, $_, 2;
    if ($rest) {;}
    print "$path;\n";
}
do_exit();

```

例 5: Report Viewer のショートカット メニューへの新しいコマンドの追加

Elements with Branches レポートは、ユーザーの指定したブランチとパス名に関連付けられたすべてのエレメントを一覧表示します。このレポート プロシージャには、以下の列が含まれます。

- Element Path (エレメント パス)
- Branch (ブランチ)

このレポート プロシージャは、`ccase-home-dir¥Reports¥Scripts¥Elements¥Branches¥Elements_with_Branches.prl` にあります。

ここでは、ショートカット メニューに [マージ マネージャ] コマンドを追加するように、このレポートを変更します。このコマンドは、ClearCase レポートではサポートされていません。したがって、この変更で必要な作業は、「例 4: 右クリック操作のショートカット メニューの変更」の場合とは異なります。

現在、ショートカット メニューには以下のコマンドが組み込まれています。

- Properties of Element
- Version Tree
- History

インターフェイス仕様

Elements_with_Branches.prl の既存のインターフェイス仕様を以下に示します。

```
if (/^-i/) {
    print "description : 'Elements with Branches'¥n";
    print "id : 2013¥n";
    print "helpfile :¥n";
    print "parameters : ";
    print "LOOKIN BRANCH";
    print "¥n";
    print_element_rightclick();
    print "fields : ";
    print "¥"Element Path¥"(element_xpn, sort 1, rightclick) ";
    print "¥"Branch¥"(branch) ";
    print "¥n";
    exit(0);
}
```

必要な変更

この変更を行うには、新しいコマンド関数用の新しいスクリプトが必要です。

このスクリプトは、¥scripts_rightclick ディレクトリに配置する必要があります (サポートされる任意のプログラミング言語でスクリプトを記述することができます)。STDIN からの入力時に、レポート プロシージャのインターフェイス仕様内の **rightclick** 変更子によって指定されたフィールドからストリームを取得するように、スクリプトをコーディングする必要があります。たとえば、clearmrgman.exe (マージ マネージャ) を起動する my_rc.prl を作成する場合、¥scripts_rightclick 内に my_rc.prl を配置する必要があります。

変更後のレポート プロシージャ

Elements_with_Branches.prl の変更バージョンを以下に示します。このレポート プロシージャは、<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp> (ただし、英語のみのご利用となります) にある T0046 パッケージの example5.prl です。

```
$start_dir = $0; $start_dir =~ s/¥¥scripts¥¥.*/¥¥scripts/;
$common_dir = $start_dir;
$common_dir =~ s/(.*)¥¥scripts/$1¥¥script_tools/;

$cc = ""; if ($cc) {;};
$ct = ""; if ($ct) {;};
$debug = ""; if ($debug) {;};
$skip_path_checks = ""; if ($skip_path_checks) {;};
$CLEARCASE_XN_SFX = ""; if ($CLEARCASE_XN_SFX) {;};
$ctfind_paths = ""; if ($ctfind_paths) {;};
```

```

$skip_path_checks = "yes"; if ($skip_path_checks) {;;}
$debug = "no"; if ($debug) {;;}

sub do_exit {
    $err = join(" ", @_);
    if ("$err" != "") {
        print STDERR "$err¥n";
    }
    sleep(2);
    if ("$err" != "") {
        exit(1);
    } else {
        exit(0);
    }
}

open(INCLUDE, "<$common_dir¥¥common_script.prl") or do_exit("error
opening include file '$common_dir¥¥common.prl'");
$buf = "";
while(<INCLUDE>) {
    $buf = $buf . $_;
}
close(INCLUDE);
eval $buf || do_exit("error on eval of include file
'$common_dir¥¥common.prl'");

my $args = $ARGV[0];
$args =~ s/%/ /g;
@args = split(";", $args);
my $ccbranch = "";
$required_args = 0;
foreach(@args) {
    s/^[ ]+//;
    s/[ ]+$//;
    validate_arg_length($_);
    if (/^-i/) {
        print "description : 'Elements with Branches'¥n";
        print "id : 2013¥n";
        print "helpfile :¥n";
        print "parameters : ";
        print "LOOKIN BRANCH";
        print "¥n";
    }
}
### customization change *** deleted following line
#print_element_rightclick();
### customization change *** added following 8 lines
print "rightclick : ";
print "my_rc(single) ";

```

```

        print "Properties_of_Element(single) ";
        print "sep ";
        print "Compare_with_Previous_Version(single) ";
        print "Version_Tree(single) ";
        print "History(single) ";
    print "¥n";
    print "fields : ";
    print "¥"Element Path¥"(element_xpn, sort 1,
rightclick) ";
    print "¥"Branch¥"(branch) ";
    print "¥n";
    exit(0);
}

if (/^LOOKIN[ ]*=[ ]*('.*')/) {
    #print "paths are $1¥n";
    check_lookin($1);
    $required_args++;
    next;
}

if (/^BRANCH[ ]*=[ ]*'*(^[']*)'*/) {
    $ccbranch = $1;
    $required_args++;
    next;
}

print STDERR "unrecognized argument: $_¥n";
print STDERR "  ccperl $0 -i¥n";
print STDERR "    for script's interface.¥n";
do_exit("¥n");
}

if ($required_args != 2) {
    print STDERR "usage: not all required arguments specified.¥n";
    print STDERR "  ccperl $0 -i¥n";
    print STDERR "    for script's interface.¥n";
    do_exit("¥n");
}

open(CTFIND, "cleartool find $ctfind_paths -nxname -branch
'brtype($ccbranch)' -print |");
while(<CTFIND>) {
    chomp;
    print "$_;$ccbranch;¥n";
}
do_exit();

```

マージマネージャ起動用の新しいショートカットメニュー コマンドをサポートするために作成された **my_rc.prl** の新しいコマンドを以下に示します。このレポート プロシージャは、<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp> (ただし、英語のみのご利用となります) の **T0046** パッケージにあります。

```
# these are all set by set_record_vars in common_rightclick.prl
#
$CLEARCASE_PN = "", $CLEARCASE_XN_SFX = "", $CLEARCASE_ID_STR = "",
$CLEARCASE_XPN = "";
$CLEARCASE_BRANCH_PATH = "", $CLEARCASE_VERSION_NUMBER = "";
$ELEMENT_RESULTS = "", $BRANCH_RESULTS = "", $VERSION_RESULTS = "";
$results = "";

$debug = "no";

$start_dir = $0; $start_dir =~
s/¥¥scripts_rightclick¥¥.*/¥¥scripts_rightclick/;
$common_dir = $start_dir;
$common_dir =~ s/(.*)¥¥scripts_rightclick/$1¥¥script_tools/;

open(INCLUDE, "<$common_dir¥¥common_rightclick.prl") or
do_exit("error opening include file
'$common_dir¥¥common_rightclick.prl'");
$buf = "";
while(<INCLUDE>) {
    $buf = $buf . $_;
}
close(INCLUDE);
eval $buf || do_exit("error on eval of include file
'$common_dir¥¥common_rightclick.prl'");

if ($CLEARCASE_PN) {};
if ($CLEARCASE_XN_SFX) {};
if ($CLEARCASE_ID_STR) {};
if ($CLEARCASE_XPN) {};
if ($CLEARCASE_BRANCH_PATH) {};
if ($CLEARCASE_VERSION_NUMBER) {};
if ($ELEMENT_RESULTS) {};
if ($BRANCH_RESULTS) {};
if ($VERSION_RESULTS) {};
if ($debug) {};

$first = "yes";
```

```

while(<STDIN>) {
    chomp;
    set_record_vars($_);
#####
####
# things to be done a record at a time are done here
    if ($first eq "yes") {
        $first = "no";
        open(COMMAND, "clearmrgman |");
        while(<COMMAND>) {;}
        close(COMMAND);
    }
#####
####
}
# things to be done with the result set as a whole go here

$results =~ s/ $//;

#print "results are $results¥n";

```

トラブルシューティング

トラブルシューティングが必要な領域は、主に次の 2 つです。

- インターフェイス仕様内のエラー
- ccperl 以外の高水準言語でのコーディング

インターフェイス仕様内のエラー

レポート プロシージャのインターフェイス仕様をコーディングする際に発生する一般的なエラーには、以下のものが挙げられます。

- プログラム内で使用するインターフェイス構文がインターフェイス仕様に適合しない。
- **parameter** 仕様に無効なパラメータ名が使用されている。
- **rightclick** 仕様から呼び出したルーチンが ¥**right_click** 内に存在しない。
- **print** ステートメントによる STDOUT への出力順序が **fields** 仕様に定義されている順序と異なる。

テスト スクリプトの **ifaces.prl** を使用すると、インターフェイス仕様内のエラーを簡単に見つけることができます。このスクリプトは、**ccperl** で記述されているカスタマイズされたレポート プロシージャをチェックします。このスクリプトは、clearcase.rational.com/contrib/T0046/T0046.zip (ただし、英語のみのご利用となります) にあります。

テスト スクリプトを起動するには、次の形式のコマンドを使用します。

ccperl ifaces.prl <path-to-script-or-directory-tree>

構成済みの共有ディレクトリ ツリーにチェックインする前に、レポート プロシージャをテストすることをお勧めします。

レポートを **Report Builder** で使用する前にテスト スクリプトを実行せず、インターフェイス仕様の処理中に解析エラーが発生した場合、新しいレポートはレポート ペインのレポートのリストに表示されません。情報のフィードバック也没有ありません。このとき、レポート ペインにレポート説明が表示されるか、何も表示されないかのいずれかです。レポート説明が表示されない場合、重大な解析エラーが発生しています。レポート説明が表示される場合、全体的に見てインターフェイス仕様は適切ですが、依然としてエラーがある可能性があります。たとえば、無効なパラメータを使用している、存在しない **rightclick** ルーチンを参照している、**STDOUT** に不適切な順序で出力を送っているなどです。

Report Builder では、パラメータが有効であるかどうかのチェックを行いません。たとえば、**my_custom_report.prl** という新しいレポート プロシージャのインターフェイス仕様が以下のようであるとします。

```
description : "This test report asks for a three known parameters and  
two unknown parameters"
```

```
id : 2500
```

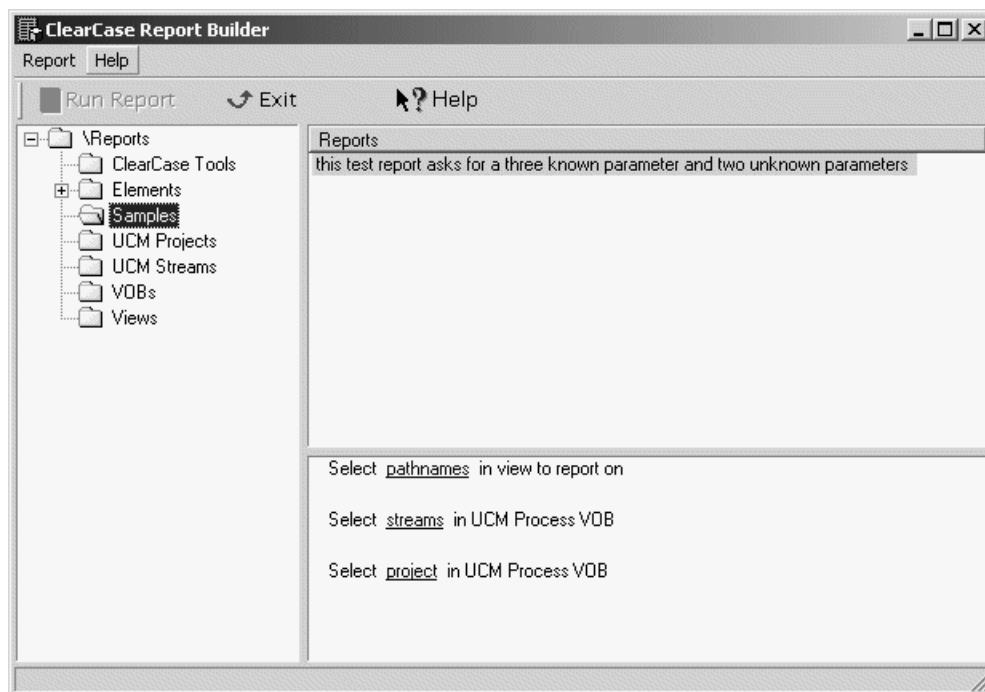
```
parameters : LOOKIN UNKNOWN_1 STREAMS FOO PROJECT
```

```
rightclick :
```

```
fields : "field 1"(string)
```

このインターフェイス仕様の 2 番目と 4 番目のパラメータは無効です。実行時に、**Report Builder** のレポート ペインにこのレポートの説明が表示されますが、2 番目と 4 番目のパラメータはパラメータ ペインに空白行として表示されます (図 64)。

図 64 無効なパラメータのある Report Builder ウィンドウ



ただし、これらのパラメータ名は ClearCase レポートのものではないため、テスト スクリプトによってエラーが検出されます (表 7 を参照)。

my_custom_report.prl:

```
desc: this test report asks for a three known parameter and two
unknown parameters
```

```
id: 2500
```

```
parm: LOOKIN
```

```
*****
```

```
ERROR: illegal parameter: UNKNOWN_1
```

```
*****
```

```
continue? (y/n) > y
```

```
UNKNOWN_1 STREAMS
```

```
*****
```

```
ERROR: illegal parameter: FOOBAR
```

```
*****
```

```
continue? (y/n) > y
```

ccperl 以外の高水準言語でのコーディング

Visual C++、Java、JavaScript、Visual Basic などの ccperl 以外の高水準言語でコーディングする場合は、<http://www.rational.com/support/downloadcenter/addins/clearcase/contrib/index.jsp> (ただし、英語のみのご利用となります) にある T0046 パッケージのプログラミング例を参照してください。

索引

C

ccase-home-dir ディレクトリ xxviii
ClearCase レポート
 カスタマイズ可能な機能 266
 カスタマイズの例 284
 共有ディレクトリの設定 270
 仕組み 265
 実行時の処理 268
 パラメータ セレクタ 281
 レポート プロシージャの
 インターフェイス仕様 273
ClearQuest 内の親/子コントロール 74
ClearQuest データベースのクエリー 28, 124
ClearQuest の統合
 UCM スキーマの設定 (手順) 67
 アクティビティの分解 74
 概要 17, 25
 環境変数 75
 計画上の課題 49
 推奨する使用法 261
 スキーマのカスタマイズ (手順) 68
 設定 17
 データベース、設定 67
 データベースのクエリー 124
 プロジェクトの使用可能にする (手順) 98
 プロジェクトへのリンクの無効化 101
 ポリシーのカスタマイズ 74
 利用可能なポリシー 61
ClearQuest 用の環境変数 75
cquest-home-dir ディレクトリ xxviii

M

main ブランチ 158
makefile と構成仕様 181
MultiSite
 PVOB 内の ClearQuest リンク 102
 UCM での使用法 20

ブランチ 159
マスターシップ転送モデル 197
リモート デリバー 113

P

PVOB
 ClearQuest データベースへのマッピング 49
 ClearQuest リンクと MultiSite 102
 概要 12
 管理 VOB としての PVOB 45
 既存の構成からの作成 93
 新規作成 (手順) 78, 79
 必要な数 44

R

Rational Unified Process 29, 30

U

UCMPolicyScripts パッケージ 51
UCM でのエレメント タイプ 48
UCM でのポリシー
 ClearQuest のカスタマイズ 74
 ClearQuest の設定 (手順) 99
 概要 18
 推奨ベースライン 55
 チェックアウト前のアクティビティの
 所有者の検証 61
 デフォルトのビュー タイプ 55
 デリバー 遷移状態 62
 デリバー前の承認 61
 デリバー前のリベース 57
 プロモーション レベル 23
 変更可能なコンポーネント 55
 保留中のチェックアウトを含むデリバー 57

UCM でのマージ

「デリバー操作」、「リベース操作」を参照

UCM とベース ClearCase、比較 1

UCM のプロジェクト

ClearQuest アクティビティ リンクの修正 101

ClearQuest データベースへの
リンクの無効化 101

概要 9

既存の構成からの作成 93

既存のプロジェクトからの作成 95

クリーンアップ タスク 127

計画上の課題 29

コンポーネントのインポート 92

コンポーネントのマッピング 30

作成 12

新規作成 (手順) 85

新規設定 78

進捗を監視するツール 121

同時、管理 147

パッチ リリースの組み込み 150

範囲を決定する要因 30

ベース ClearCase ブランチへのマージ 153

メンテナンス タスク 109

UCM のベースライン

インポートされたファイル用の作成 (手順) 94

概要 15

基本 86

削除できる場合 128

作成 21

新規作成 (手順) 115

推奨、プロモーション ポリシー 55

テスト計画 43

テスト用のストリームの作成 (手順) 105

比較 (手順) 121, 123

頻繁に作成する利点 42

プロモーション レベル 23

プロモートとデモート (手順) 119

方針 39

命名規則 43

問題の解決 (手順) 118

UCM プロジェクトのパッチ リリース 150

UnifiedChangeManagement パッケージ 51, 52

V

VOB

UCM コンポーネントへの変換 (手順) 93

ベース ClearCase の作成とデータ入力 158

VOB 作成ウィザード 78

あ

アクティビティ

ClearQuest での作成と割り当て (手順) 100

ClearQuest での分解 74

ClearQuest 統合への移行 98

ClearQuest リンクの修正 101

概要 9

所有者の検証 61

新規プロジェクトでの作成と設定 (手順) 90

デリバー後の状態遷移 62

い

イベント レコード 162

インクルード ファイル機能 167

インテグレーション ストリーム

アンロック (手順) 117

開発ビューのロード規則の更新 111

概要 14

コンポーネントの追加 (手順) 109

削除できる場合 128

プロジェクト間でのリベース (手順) 149

ベース ClearCase ブランチへのマージ 153

ロック 105

ロックについての検討内容 44

インテグレーション ビュー

UCM プロジェクト用の作成 (手順) 88

推奨ビュー タイプ 56

え

エレメント タイプ

- 事前定義とユーザー定義 223
- 割り当て方法 220

か

開発ストリーム 14

- 削除できる場合 128
- テスト用の作成 (手順) 105
- リベース (手順) 118

開発ポリシー

- 「ベース ClearCase のポリシー」、
- 「UCM でのポリシー」を参照

カスタマ サポート xxx

管理 VOB と PVOB 45

き

規則、表記 xxviii

基本ベースライン 86

く

グローバル タイプ 45, 162

け

検証テスト 43

減法マージ 212

こ

構成仕様

- あるプロジェクトの例 239
- インクルード ファイル機能 167

エレメント タイプの使用法 222

開発タスクの例 170

概要 160, 165

サンプルのプロジェクト環境 168

時間規則の例 170, 172, 177, 178

デフォルト、標準規則 166

ビルドの例 178

プラットフォーム間での共有 182

プロジェクトを監視する例 174

変更を単一ディレクトリに制限 173

ライブラリのバージョンの選択 179

構成仕様の時間規則 170, 172, 177, 178

コンポーネント

VOB の変換 (手順) 93

インテグレーション ストリームへの追加 (手順) 109

概要 13

削除できる場合 128

新規作成 (手順) 83

推奨するディレクトリ構造 34

設計上の検討内容 29

ファイルのインポート (手順) 92

プロジェクトのための編成 32

プロジェクトへのマッピング 30

補助 33

読み取り専用の候補 36

コンポーネント ツリー ブラウザ 121

さ

作業空間 14

し

システム アーキテクチャ 29

状態タイプ

概要 27

カスタム スキーマ用の設定 71

デフォルトの遷移要件 72

す

- 推奨ベースライン 55
- スキーマ (ClearQuest)
 - UCM に適用可能な要件 51
 - UCM 向けのカスタマイズ 53
 - UCM 向けのスキーマのカスタマイズ (手順) 68
 - 概要 27
 - 格納上の課題 50
 - クエリー 28
 - 事前定義、使用法 67
- スキーマのレコードタイプ、カスタム 70
- ストリーム 14

せ

- 選択マージ 211

そ

- 属性
 - 概要 161
 - 構成仕様での使用法 174
 - プロジェクト状態を監視する際の使用法 186
 - 変更依頼ポリシー 193
- ソフトウェアのビルド、ビュー構成 178

た

- タイプ マネージャ
 - compare メソッドの実装 229
 - 概要 221
 - 仕組み 225
 - 事前定義 223
 - ディレクトリの作成 225
 - テスト 231
 - メソッドの継承 226
 - ユーザー定義 223

て

- ディレクトリ構造
 - 新規作成 (手順) 91
 - 推奨、UCM コンポーネント 34
- ディレクトリ、マージ 216
- デリバリー操作
 - MultiSite 20, 113
 - エレメントタイプとマージ 48
 - 状態遷移ポリシー 62
 - 配布待ちの作業の検索 (手順) 113
 - 保留中のチェックアウトのポリシー 57
 - リベース ポリシー 57
 - リモート、完了 (手順) 114
 - リモート デリバリー 113

と

- ドキュメント
 - ヘルプの説明 xxix
- トリガ
 - checkin コマンドの例 185
 - UNIX と Windows 間での共有 134, 198
 - 概要 161
 - コマンド使用の制限 196
 - スクリプト例 190
 - チェックインの拒否 192
 - チームへの新しい作業の通知 190

は

- ハイパーリンク
 - 概要 161
 - 要求追跡メカニズム 194
- バージョン管理、候補 30

ひ

ビュー

UCM でのデフォルト タイプのポリシー 55

開発タスクの構成 170

構成仕様 165

ビルドの構成 178

プロジェクトを監視するための構成 174

ベース ClearCase の命名規則 160

変更の表示の制限 189

マージのための共有 213

履歴の構成 177, 178

ビュー プロファイル

UCM への移行 257

概要 160

表記規則 xxviii

ふ

ファイルとディレクトリのインポート 92

ファイルのマージ

仕組み 207

ブランチ

main へのマージ 213

MultiSite 159

UCM プロジェクトからの
エレメントのマージ 153

開発の停止 252

概要 158

構成仕様規則 170, 172

作成の管理 160

バグ修正ポリシー 188

複数レベル、構成仕様 172

プロジェクト方針の例 237

マージポリシー 163

マスタースhip転送モデル 197

命名規則 159

ブランチ タイプ、例 240

プロジェクト エクスプローラ 85

プロジェクトに参加ウィザード 55

プロモーション レベル

概要 23

新規プロジェクトでの定義 (手順) 87

推奨ベースラインに関連するポリシー 55

デフォルト 43

変更 (手順) 119

へ

並行開発

UCM のシナリオ 147

ベース ClearCase の拡張例 235

ベース ClearCase のメカニズム 158

ベース ClearCase でのマージ

ClearCase 以外のツール 217

GUI ツール 209

main ブランチ 213

概要 163

拡張例 244, 249

コマンド 210

仕組み 207

選択マージ 211

ソース ツリー全体 213

ディレクトリ バージョン 216

マージされた変更の削除 212

ベース ClearCase と UCM、比較 1

ベース ClearCase のプロジェクト

開発ポリシー 161

計画と設定 158

構成仕様 160

進捗を監視するためのビュー 174

ブランチ作成ポリシー 158

マージポリシー 163

ライフサイクルの拡張例 235

レポートの作成 162

ベース ClearCase のベースライン

作成、拡張例 242, 248

ラベル関連付けポリシー 188

ベース ClearCase のポリシー

コーディング標準 192

コマンド使用の制限 196

実施メカニズム 161, 185

新規作業の通知 190

ソースの状態の監視 186

ブランチ上でのバグ修正 188

ブランチ マスタースhipの転送 197

プロジェクト ファイルへのアクセス 190

ベースラインへのラベル関連付け 188

変更依頼 193

変更の説明 185

変更の表示の制限 189

マージ 163

要求追跡 194

ヘルプ、アクセス xxix

変更依頼

状態の追跡 27

ベース ClearCase 内での追跡 193

変更セット 9

ま

マスターシップ

概要 20

転送のモデル 197

め

命名規則

ClearQuest のスキーマ 49

UCM のベースライン 43

ブランチ 159

ベース ClearCase のビュー 160

ゆ

ユーザー アカウント

ClearQuest プロファイルの作成 (手順) 75

ら

ラベル

概要 161

構成仕様での使用法 177, 178

ベース ClearCase のベースライン 188

り

リベース操作

エレメント タイプとマージ 48

開発ビューのロード規則の更新 112

デリバリー操作のポリシー 57

プロジェクト間 (手順) 149

リモート デリバリー操作 113, 114

れ

レポート

ClearQuest のクエリー 124

ベース ClearCase のプロジェクト 162

ろ

ロック

概要 162

例 190

ロード規則、インテグレーション

ストリームの更新 111

わ

割り当て、確認 28