IBM Host Access Transformation Services

リッチ・クライアント・プラットフォーム・プログラマーズ・ガイド

バージョン ***9.6***

SA88-5384-02

IBM Host Access Transformation Services

リッチ・クライアント・プラットフォーム・プログラマーズ・ガイド

バージョン ***9.6***

SA88-5384-02

お願い

本書および本書で紹介する製品をご使用になる前に、 137 ページの『付録 B. 特記事項』に記載されている情報をお読みください。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC27-5903-02
IBM Host Access Transformation Services
Rich Client Platform Programmer's Guide
Version 9.6
Eighth Edition (November 2017)

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

目次

第 1 章 概要

Host Access Transformation Services (HATS) Toolkit は、文字ベースの 3270 または 5250 ホスト・アプリケーションのための使いやすいグラフィカル・ユーザー・インターフェース (GUI) を備えた リッチ ・クライアント・プラットフォーム (RCP) HATS アプリケーションを作成およびカスタマイズするための多くのツールを提供します。 HATS リッチ・クライアント・アプリケーションを、Eclipse リッチ・クライアント・プラットフォーム (RCP) の実装、Lotus Notes®、または Lotus® Expeditor Client で実行されるように開発して、エンド・ユーザーのデスクトップ向けのネイティブ・クライアント・アプリケーションを実現することができます。 HATS を使用すると、文字ベースの 3270、5250、または VT アプリケーションに含まれているロジックから、サービス指向アーキテクチャー (SOA) アセットを作成することもできます。

HATS リッチ・クライアント・アプリケーションは Eclipse プラグインとしてデプロイされる Java™ プログラムであるため、独自の HATS リッチ・クライアント・アプリケーションを作成するには、以下のトピックに関する知識が必要になります。

- Java プログラミング
- Eclipse プラグインの開発
- Standard Widgets Toolkit (SWT) (Eclipse で使用するグラフィカル・ユーザー・インターフェース・ツールキット)。

これらのトピックに関する知識がない場合は、以下の技術文書を参照してください。

- Java のチュートリアル
- Eclipse の資料

HATS Toolkit および IBM® Rational® Software Delivery Platform (SDP) のウィザードおよびエディターを使用しても追加できない機能が、HATS アプリケーションで必要になることがあります。この「*Rich Client Platform Programmer's Guide*」では、追加のプログラミングを行うことにより、ご使用の HATS アプリケーションを拡張する方法をいくつか紹介します。また、次のような HATS の基本的な概念を読者が十分理解していることが前提になっています。

- HATS によるホスト画面の処理方法
- コンポーネントおよびウィジェットを使用して変換を作成する
- イベントとアクション
- グローバル変数を使用する
- マクロを記録する

これらのトピックについて十分理解されていない場合は、「*HATS* ユーザーと管理者のガイド」の説明を参照すると、本書で説明されている情報を有効に利用するための基本的な知識を得ることができます。また、Rational SDP を使用して、リッチ・クライアント・アプリケーションを作成する方法についての知識も必要です。

この「*Rich Client Platform Programmer's Guide*」では、ご使用の HATS アプリケーションをプログラミングによって拡張する方法を説明します。次のようなことが可能です。

- HATS が提供するワークベンチ・クラスを拡張して、アプリケーションが実行されるワークベンチ・ウィンドウをカスタマイズする。詳細については、 3 ページの『第 2 章 プラグインおよびアプリケーション・クラス』を参照してください。
- プラグイン・プロジェクト内で提供されるビュー・クラスを拡張して、アプリケーションの変換ビューをカスタマイズする。詳細については、 19 ページの『変換ビューのメニューの拡張』を参照してください。
- HATS セッションとプログラマチックに対話する。例えば、別の Eclipse ビューからコマンドを送信します。詳細については、 45 ページの『他の Eclipse UI ビューとの統合』を参照してください。
- 既存のホスト・コンポーネントおよびウィジェットを拡張することにより、変換で使用する新しいコンポーネントまたはウィジェットを追加する。詳しくは、71 ページの『第 8 章 カスタム・コンポーネントおよびウィジェットの作成』を参照してください。
- アプリケーションのパースペクティブをカスタマイズする。詳しくは、 15 ページの『Host Access パースペクティブ』を参照してください。

アプリケーションを拡張する際、一部の Java ソース・ファイルを編集する必要が生じる場合があります。Rational Software Delivery Platform ヘルプの「*Java* アプリケーションの開発」というセクションで提供されている情報が、このタスクには役立ちます。

コード例

本書で取り上げられているコード例は、隣接するセクションで紹介したオブジェクトや API の使用例を示しています。これらの例をご使用のアプリケーションで使用する前に、そのコード例が目的のアクションを実行するかどうかを確認してください。 これらの例を本書からご使用のアプリケーションにコピーしても、適切に機能しない場合があります。

API 資料の使用 (Javadoc)

HATS API リファレンス資料は、多くのプログラミング・タスクを実行する際に役立ちます。この資料を表示するには、HATS の IBM Knowledge Center コレクション (http://www.ibm.com/support/knowledgecenter/SSXKAY_9.6.0) を参照し、「HATS API References (Javadoc)」リンクをクリックしてください。 HATS で提供されているアプリケーション・プログラミング・インターフェースに関する情報や例が必要な場合は、この資料を参照してください。

第 2 章 プラグインおよびアプリケーション・クラス

プラグインは、OSGi マニフェスト (MANIFEST.MF) ファイルおよびプラグイン・マニフェスト (plugin.xml) ファイルを使用して *Eclipse* プラットフォーム にその構造を記述する、構造化コンポーネントです。HATS リッチ・クライアント・プロジェクトは、Lotus Notes、Lotus Expeditor Client など、Eclipse 環境で実行するプラグインを生成するために必要な成果物を含む、Eclipse プラグイン・プロジェクトです。 プラグイン・プロジェクトは、Eclipse Software Development Kit (SDK) のコンポーネントである、Eclipse プラグイン開発環境 (PDE) を使用して開発します。Rational Software Delivery Platform (SDP) は Eclipse SDK を拡張したもので、HATS は Rational SDP の拡張をさらに進めたものです。Eclipse PDE について詳しくは、http://www.eclipse.org/pde/ を参照してください。

Eclipse PDE で提供されたツールの 1 つに、プラグインのマニフェスト・エディターがあります。このエディターは、プラグインが実行中の Eclipse 環境に対するプラグインを説明したファイルを修正するために使用されます。プラグインには通常、plugin.xml および MANIFEST.MF の 2 つのマニフェスト・ファイルがあります (プラグインが別のプラグインを拡張しない場合 plugin.xml はオプション)。ワークベンチでどちらかのファイルを開いている場合、同一のエディターが開かれます。ただし、エディターのフィールドは別のファイルと結合されています。plugin.xml は、プラグインがプラットフォームを拡張する方法、プラグインがプラグイン自体を公開する拡張、およびプラグインが機能をインプリメントする方法について記述します。MANIFEST.MF ファイルは、このプラグインが必要な ID、バージョン、ベンダー、および他のプラグインを指定します。

HATS リッチ・クライアント・プロジェクトのプラグイン・マニフェストを編集するには、プロジェクトのプロジェクト設定エディターの「概要」ページにある「プラグイン・マニフェストをオープンしてください」リンクをクリックしてください。エディターには、以下のタブが付いています。

概要	プラグイン ID およびバージョンなど、プラグインの基本属性を変更するためのフィールドが含まれています。
依存関係	このプラグインが必要としているプラグインのリストです。他のプラグインで使用可能な機能を使用する場合は、このタブを使用してプラグインに依存関係を追加します。
ランタイム	このプラグインに応じて、プラグインで使用可能な Java パッケージのリストを表示します。
拡張	このプラグインによって拡張される拡張ポイントを表示します。
拡張ポイント	HATS リッチ・クライアント・プラグインによって定義されている拡張ポイントを表示します。デフォルトで、HATS リッチ・クライアント・プラグインには拡張ポイントは定義されていません。

ビルド	エクスポート操作中にこのプラグインが作成されるときに、どのファイルを組み込むかを構成できます。他のファイルをこのプラグインに追加する場合、このリストを変更する必要があります。 注: 1. 非 Java ソース・ファイルをプラグイン・プロジェクトに追加する場合、プラグインの build.properties ファイルを更新する必要があります。build.properties ファイルは、プロジェクトがプラグインとしてエクスポートされたときに、どのファイルが組み込まれるかを制御します (デプロイメントの第一ステップ)。このファイルは、プラグイン・プロジェクトのルートに配置され、プラグインのマニフェスト・エディターを使用して編集できます。 2. プラグイン・プロジェクトで新規フォルダーを作成した場合、 フォルダーを「ビルド」タブに追加しないかぎり、プラグインをエクスポートしても新規フォルダは配置されません。
MANIFEST.MF	プラグイン MANIFEST.MF ファイルのソース・ビューを表示します。ソースを直接編集するよりも、「概要」タブ、「依存関係」タブ、および「ランタイム」タブを使用して、このファイルを変更します。
plugin.xml	plugin.xml ファイルのソース・ビューを表示します。「拡張」タブおよび「拡張ポイント」タブを使用して、このファイルを変更します。
build.properties	build.properties ファイルを表示します。「ビルド」タブを使用して、このファイルを変更する必要があります。

プラグイン・プロジェクトの拡張ポイント

プラグインが、他のプラグインによる機能部分の拡張またはカスタマイズを許可する場合、拡張ポイントを宣言します。拡張ポイントは、通常は XML マークアップと Java インターフェースの組み合わせの、拡張が準拠する規約を宣言します。この拡張ポイントに接続するプラグインは、その拡張にこの規約をインプリメントする必要があります。拡張されるプラグインが、その拡張ポイント規約のスコープを超えて接続しているプラグインに関してはなにも認識しない、ということがキー属性になっています。これによって、異なる個人または企業によって作成されるプラグインは、お互いをよく認識しなくても、シームレスに対話することができます。

com.ibm.hats.rcp.runtime.rcpApplications 拡張機能は、使用する HATS リッチ・クライアント・プラグインを HATS ランタイムに登録します。こうすることにより、HATS ランタイムは、どのプラグインが HATS アプリケーション・プラグインなのかを認識します。「アプリケーション」ビューは、この拡張ポイントをインプリメントするプラグインに基づいて移植されます。この拡張ポイントも、アプリ

ケーションのインスタンスが起動されたときに、どのビュー (デフォルトでユーザーの変換ビュー) が開くのかを示しています。

HATS RCP プロジェクトは Eclipse org.eclipse.ui.views 拡張を使用して、このプラグインの変換ビュー・クラスを Eclipse プラットフォームに登録します。

アプリケーションのインスタンスの 1 つのみの許可

デフォルトでは、ユーザーはアプリケーションの変換ビューの複数インスタンスを開くことで、HATS リッチ・クライアント・アプリケーションの複数インスタンスを一度に処理できます。ユーザーがアプリケーションの複数インスタンスを始動することを制限するために、プラグイン・プロジェクトのプラグイン・マニフェスト・ファイル (plugin.xml) にフラグを設定することができます。このフラグによって、Eclipse はアプリケーションのビューのインスタンスを 1 つのみ許可するよう通知します。このフラグを設定するには、以下を行います。

1. プロジェクト設定エディターの「概要」ページにある「プラグイン・マニフェストをオープンしてください」をクリックして、プロジェクトのプラグイン・マニフェスト・ファイルを開きます。
2. 「拡張」タブをクリックします。
3. 「すべての拡張」ツリーで、org.eclipse.ui.views ノードを展開して、プロジェクトに対応するノードを選択します。デフォルトでは、1 つのビューだけがリストされています。
4. 「拡張エレメント詳細」セクションで、**allowMultiple** 設定を *true* から *false* に変更します。
5. ファイルを保管します。

注: アプリケーションが実行中にプラグイン・マニフェスト・ファイルを変更した場合、ランタイム・ワークベンチの再始動が必要です。

ランタイム時にユーザーがアプリケーションの 2 番目のインスタンスを開こうとすると、最初のアプリケーション・インスタンスに対応するビューがフォーカスされます。

HATS ランタイム拡張プラグイン

HATS リッチ・クライアント・プラグイン・プロジェクトを作成するときに、ワークスペースに HATS リッチ・クライアント・ランタイム拡張プラグイン・プロジェクトがまだ存在していない場合には、このプロジェクトも作成されます。このプラグインには、環境で実行中のすべての HATS リッチ・クライアント・プラグインに共通するファイルおよび設定が含まれています。例えば以下のものがあります (これらに制限されません)。

- トレースおよびログの設定
- 生成されたトレースおよびログ・ファイル
- キー・マッピングの登録
- 設定ダイアログの登録
- プロパティー・ページの設定

このプラグインのデフォルト名および ID は com.ibm.hats.rcp.runtime.extension で、その初期バージョンは 1.0.0 です。

注: HATS RCP ランタイム拡張プラグインの ID は、変更しないでください。
HATS ランタイムは、開始時に、この ID を持つプラグインを検索します。

このプラグインは、複数の HATS アプリケーションに共通のファイルを含んでいるという点で、HATS EAR プロジェクトと似ています。HATS EAR プロジェクトと異なる点は、Eclipse 環境で一度に実行できるのが、これらのプラグインの 1 つのみということです。つまり、トレース設定やキーボード・マッピングなどはリッチ・クライアント環境全体に適用され、プラグイン・プロジェクト・レベルでは構成できません。

表 1 では、このプラグインに含まれているファイルがリスト表示され、説明されています。

表 1. ランタイム拡張プラグインの *HATS* 固有ファイル

ファイルまたはフォルダー	説明
イメージ (フォルダー)	「アプリケーションおよび印刷ジョブ」ビューのイメージおよび Expeditor アプリケーションのイメージが含まれています。 **注:** 例えばイメージが単一の HATS アプリケーション・プラグインによってのみ使用される場合、イメージをテンプレートに含めるには、ランタイム拡張プラグインの代わりにアプリケーション・プラグインに含めることを検討してください。
ログ (フォルダー)	HATS トレースおよびメッセージ・ファイルが含まれています。
product.xml	このプラグイン・プロジェクトを生成するために使用した HATS のバージョンを示します。このファイルは、Service Pack の適用時に更新されます。
runtime.properties	HATS ランタイム設定。トレースを使用可能にする設定が含まれています。
runtime-debug.properties	runtime.properties と同じ設定です。ただし、Eclipse ランタイム・ワークベンチがデバッグ・モードで起動したときのみ使用されます。

表 2 では、このプラグインに含まれている HATS 固有ではないファイルがリスト表示され、説明されています。

表 2. ランタイム拡張プラグイン内の *HATS* 固有でないファイル

ファイルまたはフォルダー	説明
build.properties	開発環境からのエクスポート時に、どのファイルをプラグインにパッケージする必要があるかを示しています。

表 2. ランタイム拡張プラグイン内の *HATS* 固有でないファイル (続き)

ファイルまたはフォルダー	説明
MANIFEST.MF	プラグイン記述子。プラグインの ID、名前、バージョン、およびプラグイン依存関係が示されています。
plugin.xml	プラグイン記述子。このプラグインによって提供される拡張を示しています。
plugin_xx.properties	plugin.xml および MANIFEST.MF によって使用される変換済み文字列が含まれています。

表 3 は、HATS RCP ランタイム拡張プラグインによって提供されている拡張をリスト表示します。

表 3. *RCP* ランタイム拡張

拡張ポイント **ID**	説明
org.eclipse.core.runtime.applications	HATS RCP ランタイム拡張プラグインで提供されているアプリケーション・クラスを Eclipse プラットフォームに登録します。詳しくは、 9 ページの『Application クラス』を参照してください。
org.eclipse.core.runtime.products	org.eclipse.core.runtime.applications 拡張ポイントに登録されているアプリケーションを参照する製品を定義します。詳しくは、 9 ページの『Application クラス』を参照してください。
org.eclipse.ui.perspectives	Eclipse プラットフォームに、HATS RCP ランタイム拡張プラグインで提供されている ホスト・アクセス・パースペクティブ・クラスを登録します。 15 ページの『第 3 章 パースペクティブおよびビュー』を参照してください。
org.eclipse.ui.views	「アプリケーションおよび印刷ジョブ」ビューを Eclipse プラットフォームに登録します。

表 3. *RCP* ランタイム拡張 (続き)

拡張ポイント **ID**	説明
org.eclipse.ui.propertyPages	アプリケーションのプロパティー・ページを Eclipse プラットフォームに登録します。 これらのページは、ユーザーが「アプリケーション」ビューでアプリケーションを右クリックし、「プロパティー」を選択したときに表示されます。この宣言を除去またはコメント化することによって、ユーザーがこのページを使用できないようにすることが可能です。 **注:** 「接続パラメーターのオーバーライド」ページは、接続パラメーターの指定変更がプロジェクトで使用可能になっている場合のみ表示されます。「変数オーバーライド」ページは、グローバル変数の指定変更がプロジェクトで使用可能になっている場合のみ表示されます。
org.eclipse.ui.actionSets	変換ビューに表示されている項目を印刷するために使用する印刷アクションを登録します。
org.eclipse.ui.preferencePages	HATS によって提供されている設定ページを登録します。プリファレンスは、(Eclipse の環境に応じて) 通常「ファイル」>「設定」または「ウィンドウ」>「設定」を選択することによって使用可能になります。この宣言を除去またはコメント化することによって、ユーザーがこのページを使用できないようにすることが可能です。例えば、ユーザーがトレースまたはサービス設定を変更できなくするには、「トラブルシューティング」ページに宣言されている拡張を除去します。
org.eclipse.ui.contexts	Host Access キーボード・コンテキストを登録します。コンテキストは、関連するコマンドのグループを定義します。
org.eclipse.ui.commands	使用可能なアクションごとに、コマンドを登録します。コマンドは、アクションによって処理可能な、エンド・ユーザーからの要求を表しています。

表 3. *RCP* ランタイム拡張 (続き)

拡張ポイント **ID**	説明

org.eclipse.ui.bindings	特定のコマンドに対するデフォルト・キーボード・ショートカットを登録します。宣言の `sequence` 属性を変更することによって、コマンドのデフォルト・ショートカットを変更できます。ユーザーは、「キー」設定ページを使用して、コマンドのショートカットを指定変更できます。
	
	キー・マッピングは、目的のランタイム・クライアントに基づき、スキーム ID ごとに定義します。Eclipse RCP と Lotus Expeditor Client の両方に関する schemeId を以下に示します。
	`org.eclipse.ui.defaultAcceleratorConfiguration.`
	
	Lotus Notes の schemeId を以下に示します。
	`com.ibm.workplace.notes.hannoverConfiguration.`
	
	注: 例については、「*HATS* ユーザーと管理者のガイド」のセクション『HATS リッチ・クライアント・アプリケーションのキーの再マップ』を参照してください。

com.ibm.eswe.workbench.WctApplication	Host Access パースペクティブを Lotus Expeditor Client および Lotus Notes に登録します。この拡張ポイントは、「開く」メニューに表示されるアプリケーションのリストにデータを取り込むために、Lotus Expeditor Client および Lotus Notes によって使用されます。詳細は、Lotus Expeditor 資料の『アプリケーション』を参照してください。 **注:** この宣言は、Lotus Expeditor Client または Lotus Notes へのデプロイメントをターゲットとするプロジェクトがワークスペースにない場合、表示されません。

Application クラス

このセクションで説明されているクラスは、Eclipse リッチ・クライアント・プラットフォーム環境の始動および構成を行います。これらのクラスの機能およびデフォルトのインプリメンテーションの詳細については、http://www.eclipse.org/documentation/ にある Eclipse 資料を参照し、ご使用の Eclipse のバージョンを選択し、『Building a Rich Client Platform application』という名前のセクションを検索してください。

注: HostAccessApplication、HostAccessWorkbenchAdvisor、HostAccessWorkbenchWindowAdvisor、および HostAccessActionBarAdvisor クラスは、Eclipse が構成されている場合のみ、製品構成の一部として、または開始パラメーターによって使用されます。これは、HATS RCP ランタイム拡張プラグインに定義されているアプリケーションまたは製品を使用することが目的です。これらのクラスは、このプラグインが Lotus Expeditor Client または Lotus Notes で実行中の場合は使用されません。これは、これらの製品が、固有のアプリケーション・クラスを使用しているためです。

HostAccessApplication

Eclipse アプリケーション・クラスは、Eclipse 環境のメイン・エントリー・ポイントです。Eclipse プラットフォームが始動すると、指定したアプリケーションを検出してロードします。

デフォルト・アプリケーション・クラス (HostAccessApplication) は、HATS RCP ランタイム拡張プラグイン・プロジェクトに提供されています。このクラスは、提供された HostAccessWorkbenchWindowAdvisor クラスにより構成される新規ワークベンチ・ウィンドウを作成します。

注: デフォルト・アプリケーション・クラスは簡単なものであるため、通常では変更の必要はありません。

提供されているアプリケーションを Eclipse 環境インスタンスのメイン・エントリー・ポイントとして使用するには、以下のように、アプリケーション引数にアプリケーションの ID を指定します。

Windows:

```
eclipse.exe -application
com.ibm.hats.rcp.runtime.extension.application
```

Linux:

```
./eclipse -application
com.ibm.hats.rcp.runtime.extension.application
```

Eclipse 製品は、Eclipse 環境をブランド化します。ブランド化には、「製品情報」ダイアログの定義、ウィンドウ・アイコンの設定、デフォルト・プリファレンスの定義、プラットフォーム始動時に表示されるスプラッシュ画面の指定などがあります。ブランド化の詳細については、http://www.eclipse.org/documentation/ にある Eclipse 資料を参照し、ご使用の Eclipse のバージョンを選択し、『Customizing a product』という名前のセクションを検索してください。 デフォルトの製品は、HATS ランタイム拡張プラグイン (com.ibm.hats.rcp.runtime.extension.product) に定義されています。この製品を使用して Eclipse プラットフォームを始動するには、以下のように製品の引数の製品 ID を指定します。

Windows:

```
eclipse.exe -product com.ibm.hats.rcp.runtime.extension.product
```

Linux:

```
./eclipse -product com.ibm.hats.rcp.runtime.extension.product
```

製品の拡張ポイントが、使用するアプリケーションを示すため、製品の指定時にアプリケーションを指定する必要はありません。HATS ランタイム拡張プラグイン製品の場合、HATS ランタイム拡張プラグイン・アプリケーションがデフォルトで使用されます。

HostAccessWorkbenchAdvisor

Eclipse WorkbenchAdvisor クラスは、デフォルトのパースペクティブを指定して、WorkbenchWindowAdvisor オブジェクトを構築します。このオブジェクトはワークベンチ・ウィンドウを構成します。このクラスのインプリメンテーションは、HATS RCP ランタイム拡張プラグインに提供されています。このクラス HostAccessWorkbenchAdvisor は、HostAccessWorkbenchWindowAdvisor オブジェクトの新規インスタンスを戻し、デフォルトのパースペクティブとして `hostaccess.perspectives.main` を指定します。これが、新規ワークベンチ・ウィンドウが開くときに、Host Access パースペクティブがデフォルトで表示される理由です。カスタム・パースペクティブを作成している場合、`getInitialWindowPerspectiveId()` メソッドを更新して、このパースペクティブの ID を戻すことができます。代わりに、`-perspective` 引数を使用して Eclipse が起動されるときに、以下のようにパースペクティブを指定することもできます。

Windows:

```
eclipse.exe -product com.ibm.hats.rcp.runtime.extension.product
-perspective myCompany.myCustomPerspective
```

Linux:

```
./eclipse -product com.ibm.hats.rcp.runtime.extension.product
-perspective myCompany.myCustomPerspective
```

注: Eclipse 3.1 より前は、WorkbenchAdvisor クラスはワークベンチの初期化、ワークベンチ・ウィンドウの構成、メニュー・バーおよびツールバーのアクションの作成を行っていました。これらの機能は、現在では次の 3 つのクラス、すなわち WindowAdvisor、WorkbenchWindowAdvisor、および ActionBarAdvisor に分割されています。

HostAccessWorkbenchWindowAdvisor

Eclipse WorkbenchWindowAdvisor クラスは、サイズ、位置、およびスタイルを含む、Eclipse ワークベンチ・ウィンドウの構成を行います。このクラスのインプリメンテーションは、HATS RCP ランタイム拡張プラグインに提供されています。このクラス (HostAccessWorkbenchWindowAdvisor) は、ワークベンチ・ウィンドウの初期サイズを指定し、クールバーおよび状況表示行域を非表示にします。これは、HATS はデフォルトではこれらの領域に寄与しないためです。

このクラスは、アクション・バー・アドバイザーを作成することもできます。アクション・バー・アドバイザーは、主に、ワークベンチ・ウィンドウのメニュー・バーを構成します。ワークベンチ・ウィンドウのメニュー・バーおよびツールバーのカスタマイズについて詳しくは、13 ページの『HostAccessActionBarAdvisor』 を参照してください。

ワークベンチ・ウィンドウのサイズの制御

ワークベンチ・ウィンドウのデフォルト・サイズを変更するには、以下の手順に従います。

1. 「ナビゲーター」ビューまたは「パッケージ・エクスプローラー」ビューから、com.ibm.hats.rcp.runtime.extension プラグイン・プロジェクトの hostaccess パッケージに配置されている HostAccessWorkbenchWindowAdvisor クラスを開きます。
2. `preWindowOpen()` メソッドを見つけ、IWorkbenchWindowConfigurer オブジェクトの `setInitializeSize` メソッドに渡されたパラメーターを変更します。例えば、初期ウィンドウ・サイズの幅を 800 ピクセル、高さを 600 ピクセルに設定するには、以下のコードを更新または追加します。

```
public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(800, 600));
    ...
}
```

ワークベンチ・ウィンドウを開くときに、デフォルトで自動的に最大化するには、HostAccessWorkbenchWindowAdvisor クラスの `postWindowCreate()` メソッドを以下のようにインプリメントします。

```
public void postWindowCreate() {
    PlatformUI.getWorkbench().getActiveWorkbenchWindow().
                    getShell().setMaximized(true);
}
```

注: 参照されている 1 つ以上のクラスが解決できない場合、このクラスのインポート・セクションを更新する必要があります。インポート・セクションを自動的に更新するには、ソース・ファイルのエディターの内部を右クリックして、「ソース」>「インポートの編成」を選択します。

パースペクティブ・バーの表示

パースペクティブ・バーはワークベンチ・ウィンドウの上部付近にあり、既に開いた状態のパースペクティブに素早くアクセスできるほか、エンド・ユーザーは新規パースペクティブを開くこともできます。パースペクティブ・バーを表示すれば、エンド・ユーザーは他のパースペクティブを素早く簡単に処理できるようになります。パースペクティブ・バーを表示するには、以下を行います。

1. 「ナビゲーター」ビューまたは「パッケージ・エクスプローラー」ビューから、com.ibm.hats.rcp.runtime.extension プラグイン・プロジェクトの hostaccess パッケージに配置されている HostAccessWorkbenchWindowAdvisor クラスを開きます。
2. `preWindowOpen()` メソッドを見つけ、IWorkbenchWindowConfigurer オブジェクトの `setShowPerspectiveBar` メソッドを `true` で呼び出すようにメソッドを更新します。

```
public void preWindowOpen() {
IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
configurer.setShowPerspectiveBar(true);
...
}
```

HostAccessActionBarAdvisor

Eclipse ActionBarAdvisor クラスは、ワークベンチ・ウィンドウのアクション・バー (ツールバー、クールバー、メニュー・バー、状況表示行) を構成します。このクラスのインプリメンテーションは、HATS RCP ランタイム拡張プラグインに提供されています。このクラス HostAccessActionBarAdvisor は、「ファイル」メニューの項目を含むメニュー・バーのアクションを作成します。

表 4 では、HostAccessActionBarAdvisor クラスによってインプリメント、または拡張可能なメソッドをリスト表示します。

表 4. *HostAccessActionBarAdvisor* クラス・メソッド

メソッド	説明
makeActions(IWorkbenchWindow window)	アクション・オブジェクトは、このメソッドに作成し、初期化する必要があります。
fillMenuBar(IMenuManager)	ワークベンチ・ウィンドウのメニュー・バーを、メニューおよびサブメニューに配置します。このメソッドの現行のインプリメンテーションでは、「ファイル」、「ウィンドウ」、および「ヘルプ」メニューを作成し、配置します。メニューに表示された状態のアクションを非表示にするために、アクションを MenuManager に追加しないでください。例えば、「印刷」アクションを非表示にするには、次の行を `fillMenuBar()` メソッドから除去します。 `fileMenu.add(printAction);` **注:** `fillMenuBar()` メソッドは、ワークベンチの IWorkbenchWindowConfigurer が、メニュー・バーを表示するよう指定している場合にのみ、適用されます。
fillCoolBar(ICoolBarManager)	ワークベンチ・ウィンドウのクールバーに項目を配置します。 **注:** `fillCoolBar()` メソッドは、ワークベンチの IWorkbenchWindowConfigurer が、クールバーを表示するよう指定している場合にのみ、適用されます。

ワークベンチ・ウィンドウ・ツールバーのカスタマイズ

HostAccessActionBarAdvisor クラスを拡張する方法を例示するために、以下の手順では、ツールバー項目をワークベンチ・ウィンドウのメイン・クールバー領域に追加する方法を示します。クリックすると、指定した HATS リッチ・クライアント・アプリケーションの新規インスタンスが開きます。

1. `HostAccessWorkbenchWindowAdvisor.preOpen()` メソッドを更新して、ワークベンチ・ウィンドウのクールバー領域を表示します。このメソッドは、既にインプリメントされています。このため、行う必要があるのは、以下のように、呼び出しを `setShowCoolBar(boolean)` メソッドに追加することだけです。

```
    public void preWindowOpen() {
        ...
        configurer.setShowCoolBar(true);
        ...
    }
```

2. 以下のように、新規プライベート・メンバーを HostAccessActionBarAdvisor に追加します。

```
private LaunchApplicationAction launchAppAction;
```

3. コードを HostAccessActionBarAdvisor.makeActions() メソッドに追加して、LaunchApplicationAction クラスの新規インスタンスを作成します。このクラスのコンストラクターは、以下のように 2 つの引数を受け入れます。
   a. HATS リッチ・クライアント・アプリケーションの ID
   b. IWorkbenchWindow オブジェクト (makeActions() メソッドに引き渡されます。)

```
protected void makeActions(final IWorkbenchWindow window) {
    ...
    launchAppAction = new LaunchApplicationAction("myPlugin", window);
}
```

4. ApplicationActionBarAdvsior.fillCoolBar() メソッドを追加または更新して、ツールバーを作成し、新規ランチャー・アクションをそのツールバーに追加して、次にそのツールバーをクールバーに追加します。

```
protected void fillCoolBar(ICoolBarManager coolBar) {
    // Create toolbar manager
    IToolBarManager mainToolbar = new ToolBarManager(SWT.FLAT | SWT.RIGHT);
    // Add the launch action to the toolbar manager
    mainToolbar.add(launchAppAction);

    // Add the main toolbar to the window coolbar
    coolBar.add(new ToolBarContributionItem(mainToolbar, "main"));

}
```

5. 新規 Eclipse ワークベンチを起動します。このプラグインに関連付けられている変換ビューの新規インスタンスを、(クリックすることによって) 開く新規ツールバー項目が必要です。

図 *1.* 新規ワークベンチの起動

注: launchAppAction オブジェクトで setText() メソッドおよび setImageDescriptor() メソッドを呼び出すことによって、起動アクション用に表示されるデフォルトのイメージおよびテキストの両方を変更できます。org.eclipse.jface.action.Action クラスについて詳しくは、Eclipse API を参照してください。

第 3 章 パースペクティブおよびビュー

パースペクティブは、Eclipse ワークベンチ・ウィンドウにおけるビューの初期レイアウトを定義します。それぞれのパースペクティブは、特定タイプのタスクを実行する一連の機能を提供し、その特定タイプのリソースを処理します。例えば、Java パースペクティブは、Java ソース・ファイルを編集中に一般的に使用されるビューを結合し、デバッグ・パースペクティブは、Java プログラムをデバッグするためのビューを含んでいます。

パースペクティブは、特定のメニューおよびツールバーの表示項目を制御します。パースペクティブは、可視のアクション・セットも定義します。このアクション・セットを使用して変更することにより、パースペクティブをカスタマイズすることができます。この方法で作成したパースペクティブを保管することによって、後に再びカスタム・パースペクティブを開くことができるようにします。

Host Access パースペクティブ

デフォルトのパースペクティブ・クラス (hostaccess.perspectives.MainPerspective) は、com.ibm.hats.rcp.runtime.extension プラグイン・プロジェクトで提供されています。このクラスは、Eclipse パースペクティブ・インターフェース (org.eclipse.ui.IPerspectiveFactory) をインプリメントする HATS com.ibm.hats.rcp.ui.Perspective クラスから継承されます。このパースペクティブのデフォルト・タイトルは、**Host Access** です。ただし、com.ibm.hats.rcp.runtime.extension プラグイン・プロジェクト内の plugin_en.properties ファイルを変更することによって、このタイトルを変更できます。

注: ユーザーがサポートする言語ごとに、plugin_xx.properties ファイルをそれぞれ変更する必要があります。

表 5 に、このパースペクティブ・クラスでオーバーライド可能なメソッドの一覧を示します。

表 5. パースペクティブ・クラスに対してオーバーライド可能なメソッド

メソッド	説明
addApplicationsView(IPageLayout)	HATS アプリケーション・ビューをパースペクティブに追加します。
createTransformationViewsPlaceholder(IPageLayout)	HATS 変形ビューのプレースホルダーを作成します。
createPrintJobViewsPlaceholder(IPageLayout)</code>	3270 および 5250 用の印刷ジョブ・ビューのプレースホルダー領域を作成します。
addPrintActionsSet(IPageLayout)</code>	com.ibm.hats.rcp.rntime.extension プラグインの plugin.xml で定義された印刷アクションのセットをパースペクティブに追加します。

Host Access ランタイム・アプリケーションによって最初に開かれたデフォルト・パースペクティブを変更するには、com.ibm.hats.rcp.runtime.extension プラグイン・プロジェクト内の `HostAccessWorkbenchAdvisor` ソース・ファイルを編集し、PERSPECTIVE_ID をユーザーのカスタム・パースペクティブ ID に変更します。

ホスト・アクセス・パースペクティブについて詳しくは、 11 ページの『HostAccessWorkbenchAdvisor』 を参照してください。

以下の例は、com.ibm.hats.rcp.runtime.extension 用に Host Access パースペクティブがプラグイン記述子にどのように登録されているかを示しています。

```
<extension
     point="org.eclipse.ui.perspectives">
     <perspective
            name="%PERSPECTIVE_TITLE"
            icon="images/applications_view.gif"
            class="hostaccess.perspectives.MainPerspective"
            id="hostaccess.perspectives.main">
      </perspective>
</extension>
```

Host Access パースペクティブの使用は必須ではありません。このパースペクティブは、提供した HATS アプリケーションへのエンド・ユーザーによるアクセスを容易にするためのものです。Host Access パースペクティブを取り掛かりとして使用するか、あるいは最初から開始してユーザーの要件にさらに適合するパースペクティブを開発します。

デフォルトで、Host Access パースペクティブには、「アプリケーション」ビューが含まれています。このビューについて詳しくは、『「アプリケーション」ビュー』を参照してください。

com.ibm.hats.rcp.ui.Perspective クラスについて詳しくは、Javadoc API を参照してください。

注**: Lotus Expeditor Client** および **Lotus Notes** の開発者へ: パースペクティブは Lotus Expeditor Client または Lotus Notes ウィンドウの「オープン」メニューでは表示されません。このメニューでは、com.ibm.eswe.workbench.WctApplication 拡張ポイントを使用して登録したアプリケーションを表示します。この拡張ポイントについて詳しくは、 5 ページの『HATS ランタイム拡張プラグイン』を参照してください。

「アプリケーション」ビュー

「アプリケーション」ビュー (com.ibm.hats.rcp.ui.views.ApplicationsView) は、com.ibm.hats.rcp.runtime.extension プラグインに登録されています。このビューは、クライアント環境にインストール済みの HATS リッチ・クライアント・アプリケーションへのアクセスをユーザーに提供します。ユーザーはこのビューから「プロパティー」ダイアログを起動して、アプリケーションのプロパティーを構成することもできます。

このビューの使用方法について詳しくは、「ユーザーと管理者のガイド」の『「アプリケーション」ビュー』を参照してください。

アプリケーション・インスタンスのプログラマチックな開始

「アプリケーション」ビューは、Host Access パースペクティブにデフォルトで提供されています。これにより、エンド・ユーザーは、HATS リッチ・クライアント・アプリケーションの新規インスタンスを開始できます。提供されている API によって、ユーザーは、HATS リッチ・クライアント・アプリケーションの新規イン

スタンスをプログラマチックに起動できます。これはパースペクティブで「アプリケーション」ビューを表示しないよう選択した場合に役立ちます。しかし、アプリケーションをパースペクティブから起動する方法を用意したり、アプリケーション・インスタンスを別のビューからプログラマチックに起動することが必要な場合もあります。例えば、HATS アプリケーションが 1 つのホスト・システムからデータを収集して、別の HATS アプリケーションの新規インスタンスを起動し、データを新規インスタンスに渡すことがあります。

新規のアプリケーション・インスタンスにパラメーターを渡すために、ユーザーは API によって接続パラメーターを指定変更し、グローバル変数値を初期化できます。ボタンをクリックして HATS リッチ・クライアント・アプリケーション・インスタンスを起動するには、以下を行います。

1. Visual Editor がインストールされている場合は、SWT ボタンをパレットから親コンポジットまたは変換に追加します。Visual Editor がインストールされていない場合は、標準の SWT プログラミングを使用してボタンと widgetSelected() コードを追加します。詳しくは、 21 ページの『変換の編集』を参照してください。
2. ボタンを右クリックして、「イベント」>「**widgetSelected**」を選択します。
3. `widgetSelected(SelectionEvent)` メソッドのサンプル・プレースホルダー・コードを以下のコードと置き換えます。

```
String applicationId = "myApplication";

Properties initParams = new Properties();
initParams.setProperty(com.ibm.eNetwork.beans.HOD.Session.HOST, "129.12.11.2");
initParams.setProperty(com.ibm.eNetwork.beans.HOD.Session.PORT, "623");
initParams.setProperty("hatsgv_userName", "some user");
initParams.setProperty("hatsgv_password", "xxxxx");
initParams.setProperty("hatssharedgv_someVariable", "zzzzz");

ViewInitInfo viewInitInfo = new ViewInitInfo(true, initParams);
RcpUiUtils.launchView(PlatformUI.getWorkbench().getActiveWorkbenchWindow().
                                  getActivePage(), applicationId, viewInitInfo);
```

注: 接続パラメーターを指定変更するには、プロジェクトの「プロジェクト設定」にある接続パラメーターの指定変更を使用可能にする必要があります。(また、グローバル変数の指定変更を許可するため、グローバル変数の指定変更を使用可能にする必要があります。) この機能を使用可能にすると、「接続パラメーター」ページが、アプリケーションのプロパティー・ダイアログに表示されます。プロパティー・ダイアログを使用して「接続パラメーター」ページを非表示にすることについては、 7 ページの表 3 の org.eclipse.ui.propertyPages 拡張ポイントの説明を参照してください。

変換ビュー

ビューは、情報をユーザーに対して表示する Eclipse パースペクティブの可視コンポーネントです。パースペクティブについて詳しくは、 15 ページの『Host Access パースペクティブ』を参照してください。それぞれの HATS リッチ・クライアント・アプリケーションは、実行中の Eclipse 環境にビュー (基本の com.ibm.hats.rcp.ui.views.TransformationView クラスから拡張したもの) を提供します。変換ビューは、ユーザーが HATS リッチ・クライアント・アプリケーションと対話するために使用します。

表 6 は、指定変更可能なアプリケーションの変換ビュー・クラスのメソッドをリスト表示します。

表 *6.* 変換ビュー・メソッド

メソッド	説明
`createViewActions()`	ビューのメニューのアクションを作成します。 **注:** アクションは、作成後に登録する必要があります。詳細については、 19 ページの『変換ビューのメニューの拡張』を参照してください。
`createKeyboardActions()`	com.ibm.hats.rcp.runtime.extension プラグイン・プロジェクトの plugin.xml に登録された、すべての既知の HATS キーボード機能のためのキーボード・アクション・クラスを作成します。キーボード・アクションが作成されると、それぞれが registerKeyboardActions() メソッドによってワークベンチ・ウィンドウの IKeyBindingService に登録されます。 **注:** `createKeyboardActions()` メソッドと、他のキーにキーボード・サポートを追加する方法については、「ユーザーと管理者のガイド」を参照してください。
`getApplicationKeypadDisplayInfo()`	ビューのツールバーのアプリケーション・キーパッド領域に表示するボタンのコントロール。デフォルトでは、アプリケーションの設定値を読み取って、表示するキーを識別します。表示対象を計算する必要がある場合、変換ビュー・クラス内でこのメソッドを指定変更できます。ビューのツールバーが非表示の場合、このメソッドに影響はありません。
`getHostKeypadDisplayInfo()`	ビューのホスト・キーパッド領域で表示するキーを制御し、キーの表示方法を示します。デフォルトでは、アプリケーションの設定値を読み取って、表示するキーを識別します。表示対象を計算する必要がある場合、変換ビュー・クラス内でこのメソッドを指定変更できます。
`getOiaDisplayInfo()`	変換ビューの OIA に表示する情報を示します。デフォルトでは、アプリケーションの設定値を読み取って、表示対象を識別します。表示対象を計算する必要がある場合、変換ビュー・クラス内でこのメソッドを指定変更できます。

表 6. 変換ビュー・メソッド (続き)

メソッド	説明
getToolbarDisplayInfo()	ビューのツールバーを表示するかどうかを示します。表示する場合、ツールバー上でのボタンのレンダリング方法も示します。デフォルトでは、アプリケーションの設定値を読み取って、ツールバーの表示方法を識別します。変換ビュー・クラス内でこのメソッドを指定変更できます。
shouldAutoStart()	このビューのオープン時に、アプリケーションが自動的に接続するかどうかを示しています。

注: 特定の変換について、アプリケーション・キーパッド、ホスト・キーパッド、OIA、またはツールバーの表示方法を指定変更する必要がある場合、変換ビュー・クラスではなく変換クラスで適切なメソッドを変更します。領域の表示方法をプログラマチックに変更する必要がある場合のみ、これらのメソッドを変換ビュー・クラスで指定変更します。特定の変換に対してホスト・キーパッドを表示する方法を指定変更する例については、 29 ページの『ホスト・キーパッドのカスタマイズ』を参照してください。

これらのメソッドについて詳しくは、「HATS Rich Client API Reference」を参照してください。

変換ビューのメニューの拡張

以下のコード・サンプルでは、変換ビューのメニューを拡張する方法を示しています。サンプルでは切断メニュー項目が追加され、これをクリックするとホスト接続が切断されて、アプリケーションの切断および停止イベントが実行されます。必要なのは、ビューのライフ・サイクル全体の間で適用されるビューのメニューにアクションを追加することだけです。

変換ビュー・クラスのメニューを拡張するには、以下の手順に従います (このクラスの Java ソース・ファイルは、プロジェクトの <プロジェクト名>.views パッケージにあります):

1. アクションは少なくとも変換ビュー・クラスの 2 つのメソッドでアクセスする必要があるため、変換ビュー・クラスの private メンバーとして宣言します。

   ```
   private com.ibm.hats.rcp.ui.actions.DisconnectAction disconnectAction;
   ```

2. 変換ビュー・クラスの createViewActions() メソッドを指定変更して、HATS で提供されたデフォルト・アクションをビューのメニューで使用可能にしたい場合は、super.createViewActions() を呼び出すようにします。

   ```
   protected void createViewActions() {
         super.createViewActions();

         //create an action and append the action to the view's menu.
         disconnectAction = new DisconnectAction("Disconnect",getSessionService());
         getViewSite().getActionBars().getMenuManager().add(new Separator());
         getViewSite().getActionBars().getMenuManager().add(disconnectAction);
      }
   ```

3. `updateViewActions()` メソッドを指定変更して、メニューを更新したときに新規アクションが使用可能または使用不可になるようにします。

```
protected void updateViewActions() {
      super.updateViewActions();
      //enable the action based on the session state.
      disconnectAction.setEnabled(getSessionService().getSessionServiceState().
isOperational());
   }
```

「切断」アクションの追加後、変換ビュー内のメニュー・ボタンをクリックすると、メニューに「切断」が表示されます。

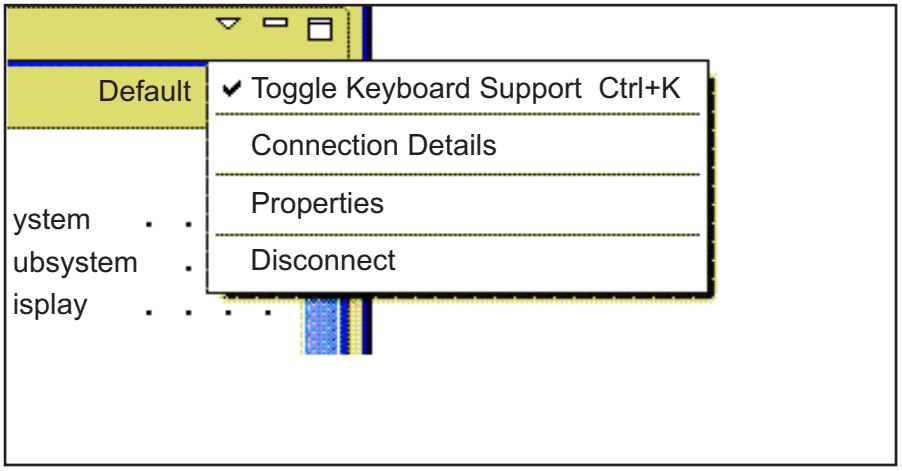


図 2. 変換ビュー・メニュー上の「切断」

第 4 章 変換

変換は、ホスト画面の変換方法を制御する HATS リソースです。リッチ・クライアント・アプリケーションでは、変換は com.ibm.hats.rcp.ui.transformations.RcpTransformation クラスから拡張する Java クラスです。このクラスは、Standard Widget Toolkit (SWT) コンポジット・クラスから拡張します。SWT コンポジット・クラスは、他のコントロールを含む UI コントロールです。変換は、SWT ウィジェット、ホスト・コンポーネント (ComponentRendering コンポジット)、DefaultRendering コンポジット、グローバル変数コントロール、マクロ・コントロール、ホスト・キー、およびアプリケーション・キーを含むことができます。変換は SWT コンポジットであるため、標準の SWT ウィジェットも追加できます。

変換は一般的に、以下のような場合に使用されます。

- ホスト画面情報の表示の再アレンジ
- ユーザーに表示したくないホスト画面情報のフィルタリング
- リッチ・クライアント・アプリケーションでの SWT ウィジェットとしてのホスト・コンポーネントの表示。

変換の編集

変換は、「リッチ・クライアント・コンテンツ」>「変換」の下の「HATS プロジェクト」ビュー内にあります。

ComponentRendering コンポジットなどの HATS 固有のコントロールを追加できます。パレットからコントロールを追加すると、変換に Java コードが生成されます。生成されたコードは、選択されたパレット項目に基づいて特定のクラスを参照します (表 7 を参照)。パレット項目は、パレットの HATS ドロワーで使用可能です。

表 7. *HATS* 固有のコントロールおよび対応するクラス

HATS 固有のパレット・ツール	コードに生成されるクラス
コンポーネント	com.ibm.hats.rcp.transform.ComponentRendering
デフォルト・レンダリング	com.ibm.hats.rcp.transform.DefaultRendering
マクロ・キー	com.ibm.hats.rcp.transform.MacroKey
グローバル変数	com.ibm.hats.rcp.transform.GlobalVariableControl
ホスト・キー	com.ibm.hats.rcp.transform.HostKey
アプリケーション・キー	com.ibm.hats.rcp.transform.ApplicationKey

この表に示したクラスについて詳しくは、以降のセクションで説明します。

HATS 固有のコントロール

標準の SWT コントロールを追加するだけでなく、ComponentRendering コンポジットなどの HATS 固有のコントロールを変換に追加することもできます。以下のセクションでは、これらの HATS 固有のコントロールについて説明します。

ComponentRendering クラス

ComponentRendering コンポジットは、指定されたコンポーネント、ウィジェット、および設定を使用して指定されたホスト画面領域を変換する専門化された UI コントロールです。選択された HATS ウィジェットに基づいて、SWT ウィジェットが生成され、ComponentRendering コンポジットに置かれます。以下のコード例で、この概念を示します。

```
ComponentRendering componentRendering = new ComponentRendering(this, SWT.NONE);
   componentRendering.setScreenCapture(<fully qualified transformation
                                                    class name>.screenCapture);
   componentRendering.setComponent("<fully qualified component class name>");
   componentRendering.setComponentSettings
                           (new com.ibm.hats.common.StringableProperties(""));
   componentRendering.setWidget("<fully qualified widget class name>");
   componentRendering.setWidgetSettings
           (new com.ibm.hats.common.StringableProperties("<widget settings>"));
   componentRendering.setRegion(new com.ibm.hats.transform.regions.
                   BlockScreenRegion(start_row, start_col, end_row, end_col));
   componentRendering.setApplyTextReplacement(true);
   componentRendering.setApplyGlobalRules(true);
```

注: `setWidgetSettings` メソッドおよび `setComponentSettings` メソッドは、パラメーターとして標準の java.util.Properties オブジェクトを取ります。HATS StringableProperties クラスの使用とは対照的に、`setProperty(String,String)` メソッドを使用して Properties オブジェクトを構成し、データを取り込むことができます。

DefaultRendering クラス

DefaultRendering コンポジットは、レンダリング・セットを使用して指定されたホスト画面領域を変換する、専門化された UI コントロールです。レンダリング・セットは、プロジェクト設定エディターの「レンダリング」タブで定義します。レンダリング・セットの定義について詳しくは、「*HATS* ユーザーと管理者のガイド」の『デフォルト・レンダリング』を参照してください。選択されたレンダリング・セットに基づいて、SWT ウィジェットが生成され、DefaultRendering コンポジットに置かれます。以下のコード例で、この概念を示します。

```
DefaultRendering defaultRendering = new DefaultRendering(this, SWT.NONE);
defaultRendering.setScreenCapture(this.screenCapture);
defaultRendering.setRenderingSetName("main");
defaultRendering.setApplyTextReplacement(true);
defaultRendering.setApplyGlobalRules(true);
defaultRendering.setRegion
         (new com.ibm.hats.transform.regions.BlockScreenRegion(1,1,24,80));
```

MacroKey クラス

com.ibm.hats.rcp.transform.MacroKey クラスは、個々のマクロをボタン (org.eclipse.swt.widgets.Button クラス) またはリンク (org.eclipse.swt.widgets.Link クラス) として表します。以下のコード例で、ボタンとして表示される MacroKey を示します。

```
macroKey = new MacroKey(this, SwtTransformationConstants.BUTTON);
macroKey.setText("macro_1");
macroKey.setMacroName("macro_1");
macroKey.addSelectionListener(new org.eclipse.swt.events.SelectionListener() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        com.ibm.hats.runtime.services.ISessionService
                            sessionService = getSessionService();
        if (sessionService != null) {
            sessionService.playMacro(macroKey.getMacroName());
        }
    }
    public void widgetDefaultSelected(org.eclipse.swt.events.SelectionEvent e) {
    }
});
```

GlobalVariableControl クラス

com.ibm.hats.rcp.transform.GlobalVariableControl クラスは、グローバル変数を静的テキスト (org.eclipse.swt.widgets.Label クラス) または入力フィールド (org.eclipse.swt.widgets.Text クラス) として表します。以下のコード例で、グローバル変数を入力フィールドとして表示する GlobalVariableControl を示します。

```
gvControl = new GlobalVariableControl(this, SwtTransformationConstants.TEXT);
gvControl.setInitialValueFromGlobalVariable(true);
gvControl.setUseAllIndices(true);
gvControl.setGlobalVariableName("userName");
gvControl.setShared(false);
gvControl.setSeparator("");
gvControl.setPasswordField(false);
gvControl.setIndex(0);
gvControl.setGlobalVariableIndexed(false);
```

注: GlobalVariableControl クラスを使用せずに変換からグローバル変数を読み取ったり変更したりすることができます。詳しくは、 27 ページの『グローバル変数値の設定および検索』を参照してください。

HostKey クラス

com.ibm.hats.rcp.transform.HostKey クラスは、個々のホスト・キーをボタン (org.eclipse.swt.widgets.Button クラス) またはリンク (org.eclipse.swt.widgets.Link クラス) として表します。複数のキーを 1 つの HostKey インスタンスに組み込むことはできません。複数のキーが必要な場合は、複数の HostKey インスタンスを定義してください。以下のコード例で、押されたときにホストに [PF1] を送信するボタンとして表示される HostKey を示します。

```
hostKey = new HostKey(this, SwtTransformationConstants.BUTTON);
hostKey.setText("F1");
hostKey.setCommand("[pf1]");
hostKey.addSelectionListener(new org.eclipse.swt.events.SelectionListener() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        com.ibm.hats.runtime.services.ISessionService
                                sessionService = getSessionService();
        if (sessionService != null) {
            sessionService.sendCommand(hostKey.getCommand());
```

```
        }
    }
    public void widgetDefaultSelected(org.eclipse.swt.events.SelectionEvent e) {
    }
});
```

ApplicationKey クラス

これは、個々のアプリケーション・キーをボタンまたはリンクとして表すクラスです。複数のキーを 1 つの ApplicationKey クラスに組み込むことはできません。複数のキーが必要な場合は、複数の ApplicationKey インスタンスを定義してください。以下のコード例で、クリックされると HATS ランタイムに切断コマンドを送信するリンクとして表示される ApplicationKey を示します。

```
applicationKey = new ApplicationKey(this, SwtTransformationConstants.LINK);
applicationKey.setText("<a>Disconnect</a>");
applicationKey.setCommand("disconnect");
applicationKey.addSelectionListener(new org.eclipse.swt.events.SelectionListener() {
    public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
        com.ibm.hats.runtime.services.ISessionService
                                    sessionService = getSessionService();
        if (sessionService != null) {
            sessionService.sendCommand(applicationKey.getCommand());
        }
    }
    public void widgetDefaultSelected(org.eclipse.swt.events.SelectionEvent e) {
    }
});
```

変換クラス

表 8 は、変換クラスでオーバーライド可能なメソッドの一覧を示します。

表 *8. RcpTransformation* メソッド

メソッド	説明
`getApplicationKeypadDisplayInfo()`	アプリケーション・キーパッドの表示方法と表示するかどうかを制御する IApplicationKeypadDisplayInfo オブジェクトを戻します。このメソッドにより、この変換に合わせてアプリケーション・キーパッドをカスタマイズできます。このメソッドからヌルを戻すと、アプリケーションのデフォルト設定に基づいてデフォルト・アプリケーション・キーパッドを表示することが示されます。
`getAutoAdvanceOrderer()`	自動拡張の実行方法と実行するかどうかを制御する IAutoAdvanceOrderer オブジェクトを戻します。デフォルトのインプリメンテーションではヌルが戻され、システムがプロジェクト設定から使用する自動拡張ハンドラーを決定することが示されます。

表 8. *RcpTransformation* メソッド (続き)

メソッド	説明
`getDefaultMonospacedFont()`	変換レベルでデフォルトのモノスペース・フォントをオーバーライドできるようにします。このメソッドの使用例については、 30 ページの『デフォルトのモノスペース・フォントのオーバーライド』を参照してください。
`getHostKeypadDisplayInfo()`	ホスト・キーパッドの表示方法と表示するかどうかを制御する IHostKeypadDisplayInfo オブジェクトを戻します。このメソッドにより、この変換に合わせてホスト・キーパッドをカスタマイズできます。このメソッドからヌルを戻すと、アプリケーションのデフォルト設定に基づいてデフォルト・ホスト・キーパッドを表示することが示されます。このメソッドの使用例については、 29 ページの『ホスト・キーパッドのカスタマイズ』を参照してください。
`getOiaDisplayInfo()`	OIA の表示方法と表示するかどうかを制御する IOiaDisplayInfo オブジェクトを戻します。このメソッドにより、この変換に合わせて OIA をカスタマイズできます。このメソッドからヌルを戻すと、アプリケーションのデフォルト設定に基づいてデフォルト OIA を表示することが示されます。
`getToolbarDisplayInfo()`	ツールバーの表示方法と表示するかどうかを制御する IToolbarDisplayInfo オブジェクトを戻します。このメソッドにより、この変換に合わせてツールバーをカスタマイズできます。このメソッドからヌルを戻すと、アプリケーションのデフォルト設定に基づいてデフォルトのツールバーを表示することが示されます。
`handlePreSubmit()`	このメソッドは、コマンドが実行依頼される直前に呼び出されます。このメソッドの使用例については、 28 ページの『変換の入力の検証』を参照してください。
`handlePostSubmit()`	このメソッドは、コマンドが実行依頼された直後に呼び出されます。
`isInitialFocusAtCursorPosition()`	初期のフォーカスをカーソル位置に置くかどうかを示します。詳しくは、「*HATS* ユーザーと管理者のガイド」の『クライアント設定』にある初期カーソル位置を参照してください。

表 8. *RcpTransformation* メソッド (続き)

メソッド	説明
`needsAutoAdvanceSupport()`	この変換に対して自動拡張機能を使用可能にするかどうかを示します。詳しくは、「*HATS* ユーザーと管理者のガイド」の『クライアント設定』にある自動フィールド進行を使用可能にするを参照してください。
`preRender()`	このメソッドは、render() メソッドの呼び出し前に呼び出されます。
`postRender()`	このメソッドは、render() メソッドが呼び出されて変換のレンダリングが完了した後に呼び出されます。
`setScreenCapture()`	HATS Toolkit での ComponentRendering および DefaultRendering の編集中に、画面取りを使用するために設定します。`componentRendering.setScreenCapture( <fully qualified transformation name>.screenCapture);`
`needsArrowKeyNavigationSupport()`	この変換に対して矢印キーを使用したナビゲーションのサポートが有効かどうかを示します。詳しくは、「*HATS* ユーザーと管理者のガイド」の『クライアント設定』にある矢印キー・ナビゲーションのサポートに関する個所を参照してください。

サンプル

ボタンからのキーの送信

パレットを使用して、SWT ボタン・ウィジェットを変換に追加します。RcpTransformation クラスの `postRender()` メソッドを指定変更します。

注: `postRender()` メソッドをオーバーライドするには、以下のステップを実行します。

1. ソース・エディター・ペインで右クリックします。
2. 「ソース」>「メソッドのオーバーライド/実装」 を選択します。
3. RcpTransformation の下にある postRender() にチェック・マークを付けます。
4. 「**OK**」をクリックします。

`postRender()` メソッドに、以下のコードを追加してください。

```
final RcpContextAttributes attrs = (RcpContextAttributes) getContextAttributes();
  button.addSelectionListener(new SelectionListener()
  {
   public void widgetDefaultSelected(SelectionEvent event) {
   }
```

```
   public void widgetSelected(SelectionEvent event) {
    attrs.getSessionService().sendCommand("[enter]");
   }
  });
```

ユーザーによる SWT リスト・ウィジェット項目選択後の入力フィールドの更新

パレットを使用して、SWT リスト・ウィジェットを変換に追加します。
RcpTransformation クラスの postRender() メソッドを指定変更します。
postRender() メソッドに、以下のコードを追加してください。

```
//retrieve a model adapter that is bound to a host field at position 1527.
//assuming that the list widget will update the field at this position on the host.
  final IModelAdapter modelAdapter = factory.getFieldAdapter(1527);
list.addSelectionListener(new SelectionListener()
  {
   public void widgetDefaultSelected(SelectionEvent event) {
   }

      //when an item in the list is selected, update the field at position 1527
      //with the value of selected item.
      public void widgetSelected(SelectionEvent event) {
    modelAdapter.setValue(list.getData(list.
getItem(list.getSelectionIndex())).toString());
   }
  });
```

注: ファクトリーを解決できない場合は、その宣言を変換に追加してください。例:

```
    private HostScreenModelAdapterFactory factory;
```

変換からのグローバル変数の値の設定

以下のコード・サンプルは、カスタマー番号をプロンプトで問い合わせる場合に、ユーザーに値の入力依頼を行うプロンプトの出し方、そして次に *customerNumber* グローバル変数をこの値に設定する方法を示しています。

```
Button button = new Button(this, SWT.NONE);
button.setText("Prompt for command");
button.addSelectionListener(new org.eclipse.swt.events.SelectionAdapter() {

public void widgetSelected(org.eclipse.swt.events.SelectionEvent e) {
    InputDialog dialog = new InputDialog(getShell(), "Prompt",
                                         "Enter a customer number:", "", null);
    if (dialog.open() == InputDialog.OK) {
        String value = dialog.getValue();
        getSessionService().getParameterDataAccessService().
                setParameterValue("hatsgv_customerNumber",value);
    }
}
 });
```

注: HATS ランタイムは、フォームが送信されるまで、setParameterValue() メソッド呼び出しによって設定された新規グローバル変数の値を認識しません。

グローバル変数値の設定および検索

以下の例では、「myGlobalVariable」という名前の非共用グローバル変数に値を設定します。

```
getSessionService().getParameterDataAccessService().setParameterValue
                              ("hatsgv_myGlobalVariable", "some new value");
```

以下の例では、同じ名前の非共用グローバル変数から値を取得します。

```
// This will return some string or null
String str = getSessionService().getParameterDataAccessService().
                                 getParameterValue("hatsgv_myGlobalVariable");
```

注: グローバル変数を共用する場合は、前記の例の hatsgv_myGlobalVariable を hatssharedgv_myGlobalVariable で置き換えてください。

変換の入力の検証

すべての HATS リッチ・クライアント変換では、com.ibm.hats.rcp.transform.IPrePostSubmitHandler インターフェースをインプリメントします。これにより、特定の変換が表示されるとき、HATS ランタイムに対する要求の直前や直後に発生する状況を制御できるようになります。このインターフェースの 1 次機能によって、HATS ランタイムへの要求が発行される前に、ユーザーから変換に提供された値を検証することができます。1 つ以上のユーザーの応答に問題が見つかった場合、ユーザーは、要求のキャンセル、パラメーターの変更、コマンドの変更、およびユーザーへのメッセージの表示を行うことができます。

注: 検証は、変換クラスの handlePreSubmit メソッドで実行する必要があります。これは、HATS ランタイムへの要求が発行される直前に、このメソッドが呼び出されるためです。要求が一度発行されると、その変更もキャンセルも行えません。

次のサンプルでは、ユーザーが行った入力に対する検証の実行方法を示しています。無効値が指定された場合、実行依頼はキャンセルされ、ユーザーに対してメッセージが表示されます。

handlePreSubmit メソッドが変換クラスで指定変更されていない場合、以下のステップを実行してください。

1. (Java エディターで) 変換のソースを右クリックして、「ソース」>「メソッドのオーバーライド/実装」を選択します。
2. RcpTransformation ノードを拡張して、handlePreSubmit(CommandEvent) の横にチェック・マークを付け、「**OK**」をクリックします。
3. 変換ソースにスタブ・メソッドを挿入します (Java コード・スタイルの設定によっては、コード例が下記のようにならない場合があります)。

```
public void handlePreSubmit(CommandEvent event) {
    // TODO Auto-generated method stub
    super.handlePreSubmit(event);
}
```

次のコード・サンプルは、仮想部分数量 (hypothetical part quantity) フィールドの範囲検査を実行する方法を示しています。ユーザーが指定した数量が 500 以下の場合、メッセージが表示され、実行依頼はキャンセルされます。このフィールドは、この変換によって変換されるホスト画面の 4 行目、20 カラム目にあると想定します。既存の handlePreSubmit(CommandEvent) メソッドを以下のコードで置き換えます。

```
public void handlePreSubmit(CommandEvent event) {
    // Avoid validation if the user is attempting to disconnect the session
    if (event.getCommand().equals(RuntimeConstants.CMD_DISCONNECT)) {
        return;
    }
```

```
        // Retrieve the value of the "quantity" field using the
        // IHostScreenDataAccessService interface
        // (which is accessed by calling the getHostScreenDataAccessService()
        // method on the session service)
        // The getFieldValue() method takes three parameters
        //
        //   Returns the value of host screen field at the specified position.
        //
        //   @param startPos The start position of the field.
        //   @param offset The offset position of the field.
        //   @param length The length of the field.
        //
        //   @return The value of the host screen field.
        //
        // The Component.convertRowColToPos() method converts a row, column value
        // into a linear screen position.
        // "5" indicates the length of the field.
        String quantityString = getSessionService().
                            getHostScreenDataAccessService().
                            getFieldValue(Component.convertRowColToPos(4, 20,
                            getHostScreen().getSizeCols()),0, 5);

        // Convert the numeric string into an int
        int quantity = 0;
        try {
            quantity = Integer.parseInt(quantityString);
        } catch (Exception ex) {
        }

        // Check to ensure the quantity entered is greater than 500
        if (quantity < 500) {
            // Display a message to the user indicating what the problem is
            MessageDialog.openError(getShell(),
                            getSessionService().getApplication().getName(),
                    "You must order at least 500 units.");

            // Cancel the event (this prevents the request from being made)
            event.setCanceled(true);
        }
    }
```

ホスト・キーパッドのカスタマイズ

- 以下のコード・サンプルでは、ホスト・キーパッドを非表示にする方法を示しています。

```
public IHostKeypadDisplayInfo getHostKeypadDisplayInfo() {
       HostKeypadDisplayInfo displayInfo = new HostKeypadDisplayInfo();
       displayInfo.setKeypadVisible(false);
       return displayInfo;
}
```

  このメソッドのデフォルト・インプリメンテーションは、NULL を戻します。NULL は、ホスト・キーパッドを表示 (または非表示に) するために (また、どのキーを表示するかを決定するために)、プロジェクト設定に定義されている設定を使用するということを HATS ランタイムに指示します。このメソッドを指定変更して、NULL 以外の値を戻すことによって、HATS ランタイムは、変換がランタイム中に適用されるときに、この HostKeypadDisplayInfo オブジェクトを使用します。

- 以下のコード例では、ホスト・キーパッドに Enter キーと F1/Help キーのみを表示する方法を示します。

```
public IHostKeypadDisplayInfo getHostKeypadDisplayInfo() {
// Construct an array of keys to include on the keypad
  KeypadKey[] keysToDisplay = new KeypadKey[] {
                              new KeypadKey("[enter]", "Enter"),
                              new KeypadKey("[pf1]", "Help") };

// Construct and return the keypad display info object
  return new HostKeypadDisplayInfo(keysToDisplay, true,
                                   IHostKeypadDisplayInfo.DISPLAY_BUTTON);
}
```

アプリケーション・キーパッドのカスタマイズ

変換の getApplicationKeypadDisplayInfo() メソッドをオーバーライドし、IApplicationKeypadDisplayInfo タイプのオブジェクトを戻すことで、アプリケーション・キーパッドを変換レベルでカスタマイズすることが可能です。

以下に示すコード例は、getApplicationKeypadDisplayInfo() メソッドをオーバーライドすることで、アプリケーション・キーパッド上に「Page down」および「Page up」を変換の適用時表示します。

```
public IApplicationKeypadDisplayInfo getApplicationKeypadDisplayInfo() {
// Define the extra keys
  KeypadKey[] extraKeys = new KeypadKey[] {
                          new KeypadKey(ECLConstants.PAGEDWN_STR, "Page Down"),
                          new KeypadKey(ECLConstants.PAGEUP_STR, "Page Up") };

// Construct a new ApplicationKeypadDisplayInfo object using the project-level
// settings and the new keys
   Application app = getSessionService().getApplication();
   Properties appKeypadSettings = app.getDefaultSettings
      (ApplicationKeypadConstants.SETTINGS_ID);
   ApplicationKeypadDisplayInfo info = new ApplicationKeypadDisplayInfo
      (appKeypadSettings, extraKeys);

  return info;
}
```

変換ビューのクラスである MainView の getApplicationKeypadDisplayInfo() クラスを上書きすることで、アプリケーション・レベルでのアプリケーション・キーパッドのカスタマイズが可能です。詳細については、17 ページの『変換ビュー』を参照してください。

デフォルトのモノスペース・フォントのオーバーライド

このコード例を使用すると、ホスト画面が標準の 80 列ではなく 132 列である場合は、小さいフォント・サイズが使用されます。

```
public Font getDefaultMonospacedFont() {
    if (getHostScreen() != null) {
        if (getHostScreen().getSizeCols() == 132) {
return FontManager.getInstance(getDisplay()).getFont("Courier New", 8, 0);
        }
    }

    return null;
}
```

他のユーザー・インターフェース・ウィジェットの統合

本セクションのサンプルでは、ユーザーは、org.eclipse.emf.common プラグインをリッチ・クライアント・プロジェクトの依存関係リストに追加する必要があります。これを行うには、以下を実行します。

1. プロジェクト用のプラグイン・マニフェスト・ファイルを開きます。

2. 「依存関係」タブで、「追加」をクリックし、「org.eclipse.emf.common」を選択します。
3. ファイルを保管します。

SWT スライダー・ウィジェットのホスト入力フィールドへの結合

スライダーは、ユーザーが値の範囲から数値を選択できるようにするコントロールです。例えば、OS/400® のメインメニューでは、スライダーを使用して選択を行えます。

パレットを使用して、SWT スライダー・ウィジェットを変換に追加します。RcpTransformation クラスの postRender() メソッドを指定変更します。postRender() メソッドに、以下のコードを追加してください。

```
// The following code creates 2 adapters :  a FieldAdapter that adapts to a
// Host Screen field located at a specific position, an IControlAdapter that
// adapts to a Slider widget, and bind these 2 adapters so that when slider
// selection is changed, its selection value is updated in the model, and
// when the data in the model is changed, the slider selection is updated.
// In this example, we bind to the a host screen field located at
// position (20,007).
    final RcpContextAttributes attrs = (RcpContextAttributes)
                                                    getContextAttributes();
  HostScreenDataModelManager dataModelManager = attrs.
                                            getHostScreenDataModelManager();
  HostScreenModelAdapterFactory factory = (HostScreenModelAdapterFactory)
                                  dataModelManager.getModelAdapterFactory();
  final IModelAdapter modelAdapter = factory.getFieldAdapter(1527);
  IControlAdapter sliderAdapter = new SliderWidgetAdapter(slider,dataModelManager);
  dataModelManager.bindControlToModel(sliderAdapter, modelAdapter);
```

SliderWidgetAdapter クラスの例

```
//This class allows the slider widget to update a host field with its values and
//also update its value from a host field.
public class SliderWidgetAdapter implements IControlAdapter {

 private Slider slider;  //the target slider widget.
 private IDataModelManager dataModelManager; //handles updating values in the model

 public SliderWidgetAdapter(Slider slider,IDataModelManager dataModelManager) {
  this.slider = slider;
  this.dataModelManager = dataModelManager;

  //create a SelectionListener such that when the value of the slider
  //is changed, it updates the host field.
  this.slider.addSelectionListener(new SelectionListener()
  {
   public void widgetDefaultSelected(SelectionEvent event) {
   }

   public void widgetSelected(SelectionEvent event) {
    updateModel();
   }
  });
 }

 //this method calls the updateModel from the IDataModelManager to
 //update the host field.
 private void updateModel() {
  if ( dataModelManager != null )
   dataModelManager.updateModel(this);
 }

 //return the value of slider selection.
```

```
 public String getValue() {
  return slider.getSelection() + "";
 }

 //this method updates the slider selection from a value. This method
 // is called when the value of a host field is changed.
 public void setValue(String value) {
  try {
   slider.setSelection(Integer.parseInt(value));
  }
  catch ( NumberFormatException e ) {
   slider.setSelection(0);
  }
 }
}
```

第 5 章 テンプレート

リッチ・クライアント変換と同様に、リッチ・クライアント・テンプレートは Java クラスです。これは、org.eclipse.swt.widgets.Composite から拡張する com.ibm.hats.rcp.ui.templates.RcpTemplate クラスから拡張します。テンプレートは、SWT ウィジェット、マクロ・コントロール、ホスト・キー、およびアプリケーション・キーを含むことができます。テンプレートは SWT コンポジットであるため、テンプレートには非 HATS ウィジェットも追加できます。

テンプレートにホスト・コンポーネント (ComponentRendering コンポジット)、デフォルト・レンダリング・コンポジット、グローバル変数コントロールを含めることはできませんが、変換にはこれらを含めることができます。

テンプレートは、HATS が提供するいずれかの定義済みテンプレートを拡張することで作成できます。テンプレートは、以下のようなアプリケーションの基本的レイアウトおよびスタイルを定義します。

- 変換に使用する前景色および背景色
- 変換に使用するフォント
- 静的ラベル (ロゴ・イメージなど)
- リンク (企業のホーム・ページなど)

表 9 に、RcpTemplate クラスによってオーバーライド可能なメソッドをリストします。

表 *9. RcpTemplate* メソッド

メソッド	説明
`getDefaultFont()`	ウィジェットが使用するデフォルトのフォントを戻します。
`getDefaultMonospacedFont()`	テンプレート・レベルでデフォルトのモノスペース・フォントをオーバーライドできるようにします。
`getDefaultBackgroundColor()`	テンプレートのデフォルト背景色を戻します。
`getDefaultForegroundColor()`	テンプレートのデフォルト前景色を戻します。
`getColorMapper()`	ホスト色がビュー上の色にマップされる方法を制御する IColorMapper オブジェクトを戻します。
`getTableColorProvider()`	テーブル制御でどの色を使用するかを制御する ITableColorProvider オブジェクトを戻します。
`getContentContainer()`	テンプレートに表示される変換やコンポジットの親になるコンポジットを戻します。

これらのメソッドについて詳しくは、「HATS Rich Client Platform API reference」を参照してください。

HATS ランタイムは、テンプレートの構成と廃棄を実行します。HATS ランタイムは、以下のステップを実行します。

1. 指定したテンプレート・クラスのインスタンスを新規構成します。このクラスは、RcpTemplate から派生させる必要があります。これは、RcpTemplate から拡張するか、RcpTemplate から拡張するクラスから拡張する必要があるということを意味します。
2. 変換クラスのインスタンスを新規構成します。
3. テンプレート・インスタンスの `setContent()` メソッドに変換インスタンスを引き渡して、このメソッドを呼び出します。このメソッドは、変換インスタンスの `applyStyleToComposite()` メソッドを呼び出します。
4. 変換ビュー・インスタンスの `setContent()` メソッドにテンプレート・インスタンスを引き渡して、このメソッドを呼び出します。

テンプレートの編集

テンプレートは、「リッチ・クライアント・コンテンツ」>「テンプレート」の下の「HATS プロジェクト」ビュー内にあります。

Java ビジュアル・エディター フィーチャー (別名 Visual Editor) が Rational SDP とともにインストールされている場合は、これを使用して変換を編集できます。このセクションの情報に加えて、「*HATS* ユーザーと管理者のガイド」の『リッチ・クライアント・プロジェクトのテンプレートの編集』を参照してください。

デフォルトで、HATS で出荷されているテンプレートは、RcpTemplate のメソッドに対するテンプレート・クラスの背景色、前景色、およびフォントを戻します。IRcpTemplate インターフェースをインプリメントする場合は、Visual Editor の「プロパティー」ビューを使用してテンプレートの背景色、前景色、およびフォントを更新することをお勧めします。

注: Eclipse API の解説書に示されているように、作成した Image、Color、および Font オブジェクトは必ず廃棄しなければなりません。`Display.getSystemColor()` メソッドを使用して検索した色は、ユーザーが作成したものではないため、廃棄しないでください。非標準色の場合、HATS によって提供されている ColorManager クラスを使用することを推奨します。このクラスは、色の作成、キャッシング、および廃棄を管理します。ColorManager によって作成された色は破棄しないでください。FontManager クラスも提供されます。このクラスは、フォントを管理する点以外は ColorManager クラスと同様に機能します。

Visual Editor には Image、Color、および Font オブジェクトをクリーンアップするときの問題があることが分かっています。Visual Editor によって提供される Properties Editor を使用してこれらをテンプレートまたは変換に追加する場合は、Java エディターを使用して手動でクリーンアップ・ロジックを追加する必要があります。例えば、Visual Editor を使用して新しい Color オブジェクトを追加すると、以下のコードが生成されます。

```
sideBarArea.setBackground(new Color(Display.getCurrent(), 221, 228, 255));
```

次に、ウィジェットを破棄するときに Color オブジェクトをクリーンアップするための以下のコード行を追加してください。

```
sideBarArea.addDisposeListener(new DisposeListener(){
    public void widgetDisposed(DisposeEvent e){
       Color background = sideBarArea.getBackground();
       if (background != null && !background.isDisposed()) {
           background.dispose();
       }
    }
});
```

Image オブジェクトおよび Font オブジェクトの場合にも、クリーンアップのための同様のロジックが必要です。以下のコード行は、Image オブジェクトをクリーンアップする場合の例です。

```
label = new Label(bannerArea, SWT.RIGHT);
gridData = new GridData(SWT.END, SWT.CENTER, true, true);
label.setImage(new Image(Display.getCurrent(), getClass().
                                getResourceAsStream("/images/sportsmast.gif")));
label.setLayoutData(gridData);
label.addDisposeListener(new DisposeListener() {
    public void widgetDisposed(DisposeEvent e){
        Image image = label.getImage();
              if (image != null && !image.isDisposed()) {
                  image.dispose();
              }
        }
});
```

前記のように、HATS が提供する ColorManager クラスまたは FontManager クラスを使用する場合は、Color オブジェクトおよび Font オブジェクトをクリーンアップする必要はありません。クリーンアップ・コードを記述するときには、Visual Editor が提供する Properties Editor ではなく、必ず Java エディターを使用してください。以下のコードは、ColorManager の使用方法の例です。

```
sideBarArea.setBackground(com.ibm.hats.rcp.ui.misc.ColorManager.
                    getInstance(Display.getCurrent()).getColor(221, 228, 255));
```

サンプル

ホスト色マッピングのカスタマイズ

ホスト色のビュー上の色へのマッピングは、テンプレート・クラスの `getColorMapper()` メソッドによって戻される IColorMapper オブジェクトによって制御されます。このメソッドのデフォルト・インプリメンテーションでは、DefaultColorMapper タイプのオブジェクトを戻します。以下の色マッパー・クラスは、HATS によって提供されています。

- com.ibm.hats.rcp.ui.templates.DefaultColorMapper
- com.ibm.hats.rcp.ui.templates.WhiteBackgroundColorMapper

DefaultColorMapper は、非ホワイト背景色を持つテンプレートで使用する必要があります。これは、ホワイト・ホスト・フィールドの色をホワイトにマッピングしているためです。WhiteBackgroundColorMapper は、ホワイトまたは明るい背景色を持つテンプレートで使用する必要があります。これは、このマッパーが、ホワイト・ホスト・フィールドの色を黒にマッピングし、より暗い色 (デフォルト色マッパーと比較して) を使用しているためです。

IColorMapper オブジェクトの mapColor() メソッドは、ホスト色の SWT RGB 値へのマッピングを行います。このメソッドは、「前景色を使用可能にする」設定が有効になっている場合、通常 HATS フィールド・ウィジェットから呼び出されます。表 10 は、このメソッドに提供可能なホスト色値をリストします。

表 *10. mapColor* メソッドに提供される色

ホストの色	説明
0	ブランク
1	青
2	緑
3	シアン
4	赤
5	マジェンタ
6	茶 (3270)、黄色 (5250)
7	白 (通常輝度)
8	グレー
9	ライト・ブルー
10	ライト・グリーン
11	ライト・シアン
12	ライト・レッド
13	ライト・マジェンタ
14	黄色
15	白 (高輝度)

色がテンプレートによってマッピングされる方法を変更するには、2 つの方法があります。

1. IColorMapper インターフェースをインプリメントするクラスを作成し、mapColor() メソッドをインプリメントし、このクラスの新規インスタンスを戻すようにテンプレートを更新します。
2. 指定された色マッパー・クラスの 1 つから拡張するクラスを作成し、mapColor() メソッドを指定変更して、このクラスの新規インスタンスを戻すようにテンプレートを更新します。

HATS によって提供されているクラスの 1 つを拡張することにした場合は、mapColor() メソッドを指定変更します。以下のコード・サンプルでは、青 (ホスト色 1) およびマジェンタ (ホスト色 5) ホスト・フィールドの色を変更する方法を示しています。

```
public class MyCustomColorMapper extends DefaultColorMapper {

 public RGB mapColor(int hostColor) {
  RGB rgb = null;

  if (hostColor == 1) {
   rgb = new RGB(100, 100, 100);
  } else if (hostColor == 5) {
   rgb = new RGB(255, 0, 0);
  } else {
   rgb = super.mapColor(hostColor);
  }
```

```
  return rgb;
 }

}
```

以下のコード・サンプルでは、カスタム色マッパーを戻すように、テンプレート・クラスの `getColorMapper()` メソッドをインプリメントする方法を示しています。

注:

1. このメソッドは、既にテンプレートに存在している場合があります。存在している場合には、現在のメソッドを以下のコードに置き換えます。

   ```
   public IColorMapper getColorMapper() {
           return new MyCustomColorMapper();
       }
   ```

2. 色マッピングは、テンプレート・レベルで制御されています。色が変換レベルでマッピングされる方法を変更する場合、新規テンプレートを作成する必要があります。このテンプレートにより、既存のテンプレートを拡張し、このテンプレートを使用するように画面のカスタマイズを構成することができます。

入力フィールドからの枠の削除

HATS ウィジェットによってレンダリングされる入力フィールドの枠を削除するには、使用しているテンプレートの `getControlStyleClass(Class)` メソッドをオーバーライドし、ある特定のクラスが SWT テキスト・ウィジェット・クラス (org.eclipse.swt.widgets.Text) である場合のみに「0」を戻すように設定することが可能です。

`getControlClassStyle()` メソッドのオーバーライドを実行するには、以下の手順に従ってください。

1. ソース・エディター・ペインで右クリックします。
2. 「ソース」>「メソッドのオーバーライド/実装」 をクリックします。
3. 「**RcpTemplate**」で、「**getControlClassStyle**」のチェック・ボックスを選択します。
4. 「**OK**」をクリックします。

「`getControlClassStyle()`」メソッドに、以下のコードを追加してください。

```
if (controlClass.equals(org.eclipse.swt.widgets.Text.class)) {
  return 0;
 } else {
  return super.getControlClassStyle(controlClass);
 }
```

第 6 章 ランタイム・サービス

HATS ランタイム、HATS リッチ・クライアント・アプリケーション、および実行中アプリケーション・インスタンスとのインターフェースとして、一連のサービスが用意されています (HATS ランタイム・サービスと総称します)。HATS ランタイムとの対話には、提供されているこれらの API を使用することをお勧めします。それにより、下位の詳細を考慮する必要がなくなります。

表 11 は、提供されているサービス・タイプに関する説明です。ランタイム・サービスおよびクライアント・サービスのインスタンスは Eclipse 環境ごとに 1 つしか存在しませんが、アプリケーション・サービスのインスタンスは複数存在することができ (単一の Eclipse 環境に複数の HATS リッチ・クライアント・アプリケーションをインストールできるためです)、セッション・サービスのインスタンスも複数存在することができます (ユーザーが複数のセッション、つまりアプリケーションのインスタンスを開始できるためです)。

表 *11.* 提供済みサービス

サービス	説明	有効範囲
ランタイム	ランタイム初期化などのランタイム関連サービスを提供します。これにより、新規クライアント・セッションが作成され、接続マネージャーおよびアプリケーション・マネージャー・クラスへのアクセスが提供されます。	Eclipse 環境ごとに 1 つ
アプリケーション	HATS リッチ・クライアント・アプリケーションに関する情報を検索するためのインターフェースを提供します。	HATS リッチ・クライアント・プラグイン・アプリケーションごとに 1 つ
クライアント	クライアントまたはユーザーに関する情報を検索するためのインターフェースを提供します。	Eclipse 環境ごとに 1 つ
セッション	特定のアプリケーション・インスタンス/ホスト接続に代わって、HATS ランタイムと対話するためのインターフェースを提供します。	実行アプリケーション・インスタンスごとに 1 つ

注: サービス・クラスは拡張できません。

注: 以下のセクションのコード例では、下記のインポートが必要です。

```
import java.util.Collection; import java.util.Iterator;
import com.ibm.hats.rcp.runtime.RcpRuntimePlugin;
import com.ibm.hats.runtime.services.IApplicationService;
import com.ibm.hats.runtime.services.IClientService;
```

```
import com.ibm.hats.runtime.services.IRuntimeService;
import com.ibm.hats.runtime.services.IServiceManager;
import com.ibm.hats.runtime.services.ISessionService;
import com.ibm.hats.runtime.services.ServiceType;
```

サービス・マネージャーへのアクセス

すべてのサービスは、単一のサービス・マネージャーによって作成され、保守されます (サービス・マネージャーは、com.ibm.hats.runtime.services.IServiceManager インターフェースをインプリメントします)。サービス・オブジェクトは、サービス・マネージャーの外側に構成しないでください。以下のコード・サンプルは、サービス・マネージャーのアクセス方法を示しています。

```
IServiceManager serviceManager = RcpRuntimePlugin.getDefault().getServiceManager();
```

表 12 に、IServiceManager オブジェクトから呼び出し可能なメソッドを示します。

表 *12. IServiceManager* メソッド

メソッド	説明
addServiceManagerListener(ServiceManagerListener)	リスナーをこのサービス・マネージャーに追加します。
removeServiceManagerListener(ServiceManagerListener)	リスナーをこのサービス・マネージャーから除去します。
getApplicationService(String)	指定したアプリケーション・プラグイン ID に応じて、IApplicationService を戻します。
getClientService(String)	指定したクライアント ID に対応する IClientService を戻します (リッチ・クライアントでは、クライアント ID は、必ず RcpRuntimeService.rcpClientId の値になっています)。
getRuntimeService()	環境に適合する IRuntimeService を戻します。
getSessionService(String, String, String)	指定したクライアント ID、アプリケーション・プラグイン ID、およびビュー ID に対応する ISessionService を戻します。
getServiceIDs(ServiceType)	このサービス・マネージャーによって管理されているサービスに対応する、指定したサービス・タイプ getServiceEntryCount(ServiceType) を持つ一連の ID を戻します。
getServiceEntryCount(ServiceType)	このサービス・マネージャーによって管理されている、指定したタイプを持つサービスの数を戻します。

詳しくは、com.ibm.hats.runtime.services.IServiceManager API を参照してください。

ランタイム・サービスの使用

ランタイム・サービス (com.ibm.hats.runtime.services.IRuntimeService インターフェースをインプリメントする) は、HATS ランタイムと対話するメソッドを提供します。このサービスに提供されているほとんどのメソッドは、直接呼び出すことはできません。ただし、HATS ランタイムの他のコンポーネントが通信できるように公開されています。アプリケーションのランタイム・サービス・オブジェクトは、サービス・マネージャーから検索する必要があります。ランタイム・サービスを取得するためのコード例を以下に示します。

```
IRuntimeService runtimeService =
            serviceManager.getRuntimeService();
```

以下のメソッドは、ランタイム・サービスによって呼び出すことができます。

表 13. IRuntimeService メソッド

メソッド	説明
`getConnectionManager()`	デフォルトの ConnMgr インスタンスを戻します。
`getServiceManager()`	このサービスをインスタンス化したサービス・マネージャーを戻します。

アプリケーション・サービスの使用

アプリケーション・サービス (com.ibm.hats.runtime.services.IApplicationService インターフェースをインプリメントする) は、HATS リッチ・クライアント・アプリケーションに関連する情報を検索するメソッドを提供します。Eclipse 環境にインストールされ、使用可能になっている、各 HATS リッチ・クライアント・アプリケーション・プラグインには、IApplicationService オブジェクトが関連付けられています。アプリケーションのアプリケーション・サービス・オブジェクトは、サービス・マネージャーから検索する必要があります。アプリケーション・サービスを取得するためのコード例を以下に示します。

```
IApplicationService applicationService =
                serviceManager.getApplicationService("myPluginId");
```

`getApplicationService(String)` メソッドは、指定されているアプリケーション ID が無効な場合、またはプラグインが開始できなかった場合、NULL を戻します。

表 14 に、IApplicationService オブジェクトから呼び出し可能なメソッドを示します。

表 14. IApplicationService メソッド

メソッド	説明
`getApplication()`	アプリケーションに関する情報が含まれている Application オブジェクトを戻します。このオブジェクトは application.hap ファイルとの同期が保証されないため、このメソッドによって戻されたオブジェクトは情報の取得のみに使用してください。このオブジェクトを使用して設定を変更しないでください。

表 14. *IApplicationService* メソッド (続き)

メソッド	説明
getApplicationId()	アプリケーション ID を戻します。
getConfig()	構成を戻します。
getRuntimeService()	ランタイム・サービスを戻します。
getServiceManager()	このサービスをインスタンス化したサービス・マネージャーを戻します。

クライアント・サービスの使用

クライアント・サービス (com.ibm.hats.runtime.services.IClientService インターフェースをインプリメントする) は、クライアント・サービスに関連する情報を取得するメソッドを提供します。クライアント・サービスを取得するためのコード例を以下に示します。

```
IServiceManager serviceManager = RcpRuntimePlugin.getDefault().getServiceManager();
IClientService clientService = serviceManager.getClientService("theClientId");
```

表 15 に、IClientService オブジェクトから呼び出し可能なメソッドを示します。

表 15. *IClientService* メソッド

メソッド	説明
getClientId()	クライアント ID を戻します。リッチ・クライアントでは、常に *RcpRuntimeService.rcpClientId* の値になります。
getRuntimeService()	ランタイム・サービスを戻します。
getServiceManager()	このサービスをインスタンス化したサービス・マネージャーを戻します。
getSession(String)	指定されたセッションを戻します。 **注:** これは、型 ISession のホスト・セッションです (セッション・サービス・インスタンスではありません)。

セッション・サービスの使用

セッション・サービス (com.ibm.hats.runtime.services.SessionService インターフェースをインプリメントする) は、HATS リッチ・クライアント・アプリケーションのインスタンスと対話するためのメソッドを提供します。実行中の各アプリケーション・インスタンス (変換ビュー・インスタンスなど) にはセッション・サービスが関連付けられています。変換ビュー・インスタンスおよびセッション・サービスとの間には、1 対 1 のマッピングが存在します。

セッション・サービスの完全なコレクションを取得するためのコード例を以下に示します。

```
IServiceManager serviceManager =
        RcpRuntimePlugin.getDefault().getServiceManager();
Collection sessionServices =
        serviceManager.retrieveServiceEntries( ServiceType.SESSION );
```

```
Iterator itSessionServices =
         sessionServices.iterator();
while ( itSessionServices.hasNext() )
{
ISessionService aSessionService =
      (ISessionService) itSessionServices.next();
// Your custom code goes here
}
```

表 16 に、アプリケーションのインスタンスである ISessionService オブジェクトによって呼び出すことができるメソッドを示します。

表 *16. ISessionService* メソッド

メソッド	説明
addPresentationListener (IPresentationListener)	指定されたプレゼンテーション・リスナーをリストに追加します。プレゼンテーションが更新されると、リスナーに通知されます。
addSessionServiceListener (ISessionServiceListener)	指定されたセッション・サービス・リスナーをリストに追加します。セッションの状態が変化したとき、またはコマンドが送信されるときには、リスナーに通知されます。
canSendCommand(String)	現在のセッション・サービスの状態で指定のコマンドを送信できる場合は true のブール値を戻します。それ以外の場合は false のブール値を戻します。
disconnect()	ホスト・セッションを切断します。このメソッドは、アプリケーション・キーパッド上の「切断」ボタンによって呼び出されます。関連付けられている変換ビューは、このメソッドではクローズされません。このメソッドは、セッション・サービスが既にコマンドを処理している場合には、何もアクションを実行しません。
getApplication()	このセッション・サービス・インスタンスに関連付けられたアプリケーションを戻します。このメソッドによって戻されたオブジェクトは情報の取得のみに使用してください。このオブジェクトを使用して設定を変更しないでください。
getApplicationId()	このセッション・サービスに関連付けられたアプリケーションのアプリケーション ID (プラグイン ID など) を戻します。
getApplicationService()	このセッション・サービス・インスタンスに関連付けられたアプリケーション・サービスを戻します。
getClientId()	このセッション・サービス・インスタンスに関連付けられたクライアント ID を戻します。
getCurrentState()	このセッション・サービス・インスタンスの現在の状態を戻します。

表 *16. ISessionService* メソッド (続き)

メソッド	説明
`getHostScreen()`	このセッション・サービス・インスタンスに関連付けられたホスト画面を戻します。これは、HATS ランタイムによって処理された最後のホスト画面を戻します。戻り値は必ずしも現在のホスト画面とは限りません。
`getHostScreenDataAccessService()`	フォーム上のホスト・フィールド関連制御の値を含むモデルを戻します。このメソッドは、変換がビュー上で現在表示されている場合のみ有効です。
`getLocale()`	メッセージの表示に使用するロケールを戻します。プロジェクト設定エディターでアプリケーションのクライアント・ロケール設定を変更した場合、このロケールは Eclipse 環境のロケールと異なる場合があります。
`getMacroPromptDataAccessService()`	モデル・データへのアクセスに使用するサービスを戻します。
`getParameterDataAccessService()`	フォーム上の非ホスト・フィールド関連制御の値を含むモデルを戻します。このサービスは、グローバル変数値など、その他のフォーム値の設定および検索を行うために使用されます。これらの値は、現行フォームが送信されると HATS ランタイムに引き渡されます。
`getRuntimeService()`	ランタイム・サービスを戻します。
`getServiceManager()`	このサービスをインスタンス化したサービス・マネージャーを戻します。
`getSession()`	このセッション・サービス・インスタンスに関連付けられたホスト・セッションを戻します。
`getSessionServiceState()`	セッション・サービスの状態を状態オブジェクトとして戻します。
`getViewId()`	変換ビューのインスタンス ID を戻します。
`isAsyncUpdateEnabled()`	非同期更新が使用可能な場合は true のブール値を戻します。それ以外の場合は false のブール値を戻します。
`isCommandSupported(String)`	指定されたコマンドがサポートされる場合は true のブール値を戻します。それ以外の場合は false のブール値を戻します。
`playMacro(String)`	指定したマクロを実行します。
`playMacro(String, Properties)`	指定したマクロを実行し、マクロで定義されているすべてのプロンプトに値を引き渡します。
`removePresentationListener (IPresentationListener)`	指定されたプレゼンテーション・リスナーをリストから除去します。
`removeSessionServiceListener (ISessionServiceListener)`	指定されたセッション・サービス・リスナーをリストから除去します。

表 16. *ISessionService* メソッド (続き)

メソッド	説明
`sendCommand(String)`	指定したコマンドを送信します。一般的なコマンドは、[pf1]、[pf2]、[enter]、[fldext] です。**注:** コマンドの前後の大括弧は必須です。
`sendCommand(String, Properties)`	String によって指定されたコマンドを関連プロパティーとともに送信します。コマンドは、[enter] などの Host On-Demand ニーモニックか、default などの HATS コマンドのいずれかです。追加のパラメーターを指定できます。これらのパラメーターは、実行依頼される要求に組み込まれます。これらのパラメーターにより、ホスト入力フィールドなどの SDO モデルまたはグローバル変数から収集される同じ名前のパラメーターがオーバーライドされます。オーバーライド・パラメーターの値は型 String[] でなければなりません。
`sendContinue()`	**continue** コマンドを送信します。このコマンドを送信して、「コンポジットを表示」アクションまたはマクロ・プロンプト・アクションから続行する必要があります。
`sendEnter()`	Enter キー・コマンドを送信します。
`sendF1() - sendF24()`	各 PF キー・コマンドを送信します。
`sendRefresh()`	更新コマンドを送信します。
`sendShowDefault()`	デフォルト表示コマンドを送信します。
`start(Properties)`	指定された接続およびグローバル変数オーバーライドを使用してセッション・サービスを開始します。
`stop()`	セッション・サービスを停止します。
`updatePresentation(IPresentable)`	指定されたプレゼンテーション可能オブジェクトでプレゼンテーション (一般には変換ビュー) を更新します。

変換ビューに関連する ISessionService オブジェクトは、ビュー上で `getSessionService()` メソッドを呼び出すことによって、検索できます。このメソッドは、ITransformationView インターフェースによって提供されます。このインターフェースは、基本 TransformationView クラスによってインプリメントされます。

他の Eclipse UI ビューとの統合

ランタイム・サービス API を使用することによって、HATS 変換ビュー (HATS リッチ・クライアント・アプリケーション) は他の Eclipse UI ビュー (非 HATS リッチ・クライアント・アプリケーション) と通信できます。この通信は、HATS アプリケーションに対して着信または発信のいずれかにすることができます。着信通信は、HATS リッチ・クライアント・セッション受信データまたは HATS 外部のエンティティーからのコマンドとして定義されます。発信通信は、HATS リッチ・クライアント・セッション送信データまたは外部エンティティーに対するコマ

ンドとして定義されます。この機能により、適切なデータを複数のソースから取得して単一のビューで表示できます。したがって、HATS リッチ・クライアント・アプリケーションは、他の Eclipse リッチ・クライアント・アプリケーションからデータを受信して表示するビューか、1 つ以上の Eclipse リッチ・クライアント・アプリケーションにデータを提供して表示させるアプリケーションのいずれかにすることができます。

着信通信のシナリオ

以下のシナリオでは、フライト番号を入力として受け入れ、そのフライトに関する情報を表示するホスト・アプリケーションを想定します。このアプリケーションは HATS RCP アプリケーションに変換されています。別の Eclipse RCP ベースのアプリケーションは、フライト番号を含む予約情報を受け入れます。フライト番号は、このアプリケーションに入力されると HATS RCP アプリケーションに送信され、そのフライト番号に対する情報で HATS ビューが更新されます。このシナリオのコード例を以下に示します。

```
IServiceManager serviceManager = RcpRuntimePlugin.getDefault().getServiceManager();
IClientService clientService = serviceManager.getClientService("theClientId");

/*-------------------------------------------------------------------------------
//
// Provides an example of a HATS application receiving information from another
// application.  This example cannot be executed directly but is provided to
// highlight the approach required to achieve this functionality.
//
// @param  Flight number from GUI as a Java String.  This value cannot be
// <code>null</code>.
//
*///-------------------------------------------------------------------------------
    public void inboundToHATSApplication( String flightNumber )
    {
        // Save flight number from GUI
        Properties params = new Properties();

                params.setProperty( "fltNum",  flightNumber );
        // Get the service manager from singleton plug-in instance.
IServiceManager serviceManager = RcpRuntimePlugin.getDefault().getServiceManager();
// This will be the plug-in session we use to play our macro
        ISessionService aSessionService = null;

        // Find the correct HATS plug-in session
        Collection sessionServices =
                    serviceManager.retrieveServiceEntries( ServiceType.SESSION );
        Iterator itSessionServices = sessionServices.iterator();

        while ( itSessionServices.hasNext() )
        {
                        aSessionService = (ISessionService) itSessionServices.next();
            // Assumes the first instance of the HATS plug-in called "myApp"
            // is the one we want.
if ( "myApp".getApplicationId().equals( aSessionService ) )
            {
                break;
            }
        }

        // Refer to documentation to see how to launch an RCP plug-in instance if
        // one is not already launched
        if ( aSessionService != null )
        {
            // Really should check the state in a limited-time loop,
            // but let's assume we can go ahead
            if ( aSessionService.getSessionServiceState().isOperational() )
            {
                // Ask HATS session service to play the macro with the fltNum
                // from our GUI
                // Assumes that the macro can start from current screen or from
                // the results screen.
```

```
                aSessionService.playMacro( "displayFlightInfo", params );
            }
        }
    }
```

サンプル

複数のランタイム・サービスにアクセスする方法を示すクラスおよびメソッドの例

```
/*-------------------------------------------------------------------------------
//
// Host Access Transformation Services technology
//
// Module Name: RuntimeServicesExamples.java
//
// Description: Provides java code samples for accessing HATS Rich Client Program
// Runtime Services.
//
*///-------------------------------------------------------------------------------

package com.ibm.hats;

import java.util.Collection;
import java.util.Iterator;

import com.ibm.hats.rcp.runtime.RcpRuntimePlugin;

import com.ibm.hats.runtime.services.IApplicationService;
import com.ibm.hats.runtime.services.IClientService;
import com.ibm.hats.runtime.services.IRuntimeService;
import com.ibm.hats.runtime.services.IServiceManager;
import com.ibm.hats.runtime.services.ISessionService;
import com.ibm.hats.runtime.services.ServiceType;

//-------------------------------------------------------------------------------


/** <code><B>
*  Class Name:  </B> RuntimeServicesExamples <B><BR>
*  Class Type:  </B> Normal <B><BR>
*  Base Class:  </B> None   <B><BR>
*  Intf Class:  </B> None   <B><BR>
*  Description: </code></B> Rich Client Program Runtime Services java code samples.
*///-------------------------------------------------------------------------------
public class RuntimeServicesExamples
{
    //-----------------------------| Constants |----------------------------------

    //-----------------------------| Variables |----------------------------------

    //-------------------------------------------------------------------------------
    /**
    * The default constructor.
    *///-------------------------------------------------------------------------------
    public RuntimeServicesExamples()
    {
        super();
    }

    //-------------------------------------------------------------------------------
    /**
    * Provides an example of accessing each application service.
    *///-------------------------------------------------------------------------------
```

```
public void accessApplicationService()
{
    Collection applicationServices = this.retrieveApplicationServices();

    Iterator itApplicationServices = applicationServices.iterator();

    while ( itApplicationServices.hasNext() )
    {
        IApplicationService aApplicationService = (IApplicationService)
                                        itApplicationServices.next();

        // Custom code goes here
    }
}

//-----------------------------------------------------------------------------
/**
* Provides an example of accessing each client service.  Currently, there is
// only one.
*///-----------------------------------------------------------------------------
public void accessClientService()
{
    Collection clientServices = this.retrieveClientServices();

    Iterator itClientServices = clientServices.iterator();

    while ( itClientServices.hasNext() )
    {
        IClientService aClientService = (IClientService)
                                                itClientServices.next();

        // Custom code goes here
    }
}

//-----------------------------------------------------------------------------
/**
* Provides an example of accessing each runtime service.  Currently, there is
// only one.
*///-----------------------------------------------------------------------------
public void accessRuntimeService()
{
    Collection runtimeServices = this.retrieveRuntimeServices();

    Iterator itRuntimeServices = runtimeServices.iterator();

    while ( itRuntimeServices.hasNext() )
    {
        IRuntimeService aRuntimeService = (IRuntimeService)
                                                itRuntimeServices.next();

        // Custom code goes here
    }
}

//-----------------------------------------------------------------------------
/**
* Provides an example of accessing each session service.
*///-----------------------------------------------------------------------------
public void accessSessionService()
{
    Collection sessionServices = this.retrieveSessionServices();

    Iterator itSessionServices = sessionServices.iterator();

    while ( itSessionServices.hasNext() )
    {
```

```
        ISessionService aSessionService = (ISessionService)
                                                        itSessionServices.next();

        // Custom code goes here
    }
}

//---------------------------------------------------------------------------
/**
* Retrieve the client service.
*
* @return  The client service.
*///-------------------------------------------------------------------------
public IClientService getClientService( final String clientId )
{
    return this.getServiceManager().getClientService( clientId );
}

//---------------------------------------------------------------------------
/**
* Retrieve the runtime service.
*
* @return  The runtime service.
*///-------------------------------------------------------------------------
public IRuntimeService getRuntimeService()
{
    return this.getServiceManager().getRuntimeService();
}

//---------------------------------------------------------------------------
/**
* Retrieve the service manager.
*
* @return  The service manager.
*///-------------------------------------------------------------------------
public IServiceManager getServiceManager()
{
    return RcpRuntimePlugin.getDefault().getServiceManager();
}

//---------------------------------------------------------------------------
/**
* Provides an example of retrieving a Collection of current application
* service objects.
*
* @return  A Collection of current application service objects.
*///-------------------------------------------------------------------------
public Collection retrieveApplicationServices()
{
    IServiceManager manager = this.getServiceManager();

    Collection applicationServices = manager.retrieveServiceEntries(
                                                    ServiceType.APPLICATION );

    return applicationServices;
}

//---------------------------------------------------------------------------
/**
* Provides an example of retrieving a Collection of current client service
* objects.  Currently, there is only one.
*
* @return  A Collection of current client service objects.
*///-------------------------------------------------------------------------
public Collection retrieveClientServices()
{
    IServiceManager manager = this.getServiceManager();
```

```
        Collection clientServices = manager.retrieveServiceEntries(
                                                              ServiceType.CLIENT );

        return clientServices;
    }

    //-------------------------------------------------------------------------
    /**
    * Provides an example of retrieving a Collection of current runtime service
    * objects.  Currently, there is only one.
    *
    * @return  A Collection of current runtime service objects.
    *///-------------------------------------------------------------------------
    public Collection retrieveRuntimeServices()
    {
        IServiceManager manager = this.getServiceManager();

        Collection runtimeServices = manager.retrieveServiceEntries(
                                                              ServiceType.RUNTIME );

        return runtimeServices;
    }

    //-------------------------------------------------------------------------
    /**
    * Provides an example of retrieving a Collection of current session service
    * objects.
    *
    * @return  A Collection of current session service objects.
    *///-------------------------------------------------------------------------
    public Collection retrieveSessionServices()
    {
        IServiceManager manager = this.getServiceManager();

        Collection sessionServices = manager.retrieveServiceEntries(
                                                              ServiceType.SESSION );

        return sessionServices;
    }

    //-------------------------------------------------------------------------
    /**
    * Provides an example of turning off border rendering for all input fields.
    *
    *
    *///-------------------------------------------------------------------------
    public int getControlClassStyle(class controlClass)
    {
      if (controlClass.equals(Text.class))
      {
        return 0;// super.getControlClassStyle(controlClass);
       }
      else
      {
        return super.getControlClassStyle(controlClass);
    }
}
```

3270 印刷ジョブの listen

サービス・マネージャーを使用してプラグインが 3270 印刷ジョブ・イベントを listen できるようにする方法を以下の例に示します。以下の変更は、RCP アプリケーション・プラグイン・クラスに対して行います。

1. ServiceManagerListener、ISessionServiceListener、および IPrintJobManagerChangeListener をインプリメントします。
2. ServiceManagerListener メソッド public void serviceChanged (ServiceManagerEvent event) をインプリメントします。このメソッドは、サービス・マネージャーが変化したときに呼び出されます。この例では、いつ新規セッションが作成または破棄されたかを判別するために使用します。新規セッションが作成されるときには、自分自身をリスナーとして追加します。セッションが破棄されるときには、自分自身をリスナーから除去します。
3. ISessionServiceListener メソッドをインプリメントします。
   - public void sessionStateChanged(StateChangeEvent event) - このメソッドは、セッションの状態が変化したときに呼び出されます。このメソッドは、セッションの印刷ジョブ・マネージャーを検出し、自分自身をリスナーとして追加するために使用します。
   - public void aboutToProcessCommand(CommandEvent event) - この例では、このメソッドは空のままにします。
   - public void afterProcessCommand(CommandEvent event) - この例では、このメソッドは空のままにします。
4. IPrintJobManagerChangeListeners をインプリメントします。
   - public void addPrintJob(PrintJobManager printJobManager, PrintJob printJob) このメソッドは、印刷ジョブが印刷ジョブ・マネージャーに追加されるときに呼び出されます。
   - public void removePrintJob(PrintJobManager printJobManager, PrintJob printJob) このメソッドは、印刷ジョブが印刷ジョブ・マネージャーから除去 (削除) されるときに呼び出されます。
   - public void updatePrintJob(PrintJobManager printJobManager, PrintJob printJob) このメソッドは、印刷ジョブが更新されるときに呼び出されます。
5. アプリケーション・プラグインの start() メソッドで、自分自身をサービス・マネージャー・リスナーとして追加します。
6. アプリケーション・プラグインの stop() メソッドで、自分自身をサービス・マネージャー・リスナーから除去します。

```
package printExample;

import java.util.HashSet;

import org.eclipse.jface.resource.ImageDescriptor;
import org.osgi.framework.BundleContext;

import com.ibm.hats.rcp.runtime.RcpRuntimePlugin;
import com.ibm.hats.rcp.ui.AbstractRcpApplicationPlugin;
import com.ibm.hats.runtime.ApplicationSpecificInfo;
import com.ibm.hats.runtime.ClientSpecificInfo;
import com.ibm.hats.runtime.IPrintJobManagerChangeListener;
import com.ibm.hats.runtime.PrintJob;
import com.ibm.hats.runtime.PrintJobManager;
import com.ibm.hats.runtime.PrintResourceHandler;
import com.ibm.hats.runtime.PrintSpecificInfo;
import com.ibm.hats.runtime.events.CommandEvent;
import com.ibm.hats.runtime.events.ISessionServiceListener;
import com.ibm.hats.runtime.events.ServiceManagerEvent;
import com.ibm.hats.runtime.events.ServiceManagerListener;
```

```
import com.ibm.hats.runtime.events.StateChangeEvent;
import com.ibm.hats.runtime.services.IService;
import com.ibm.hats.runtime.services.IServiceManager;
import com.ibm.hats.runtime.services.ISessionService;
import com.ibm.hats.util.StateType;

/**
 * The activator class controls the plug-in life cycle
 */
public class PrintExamplePlugin extends AbstractRcpApplicationPlugin implements
                                ServiceManagerListener, ISessionServiceListener,
                                IPrintJobManagerChangeListener
{
    // The plug-in ID
    public static final String PLUGIN_ID = "PrintExample";

    // The shared instance
    private static PrintExamplePlugin plugin;

    // The service manager
    private IServiceManager serviceManager = null;

    // The print managers we are listeners for
    private HashSet printJobManagers = new HashSet();

    /**
     * The constructor
     */
    public PrintExamplePlugin()
    {
        plugin = this;

        System.out.println("PrintExamplePlugin: ctor");
    }

    /*
     * (non-Javadoc)
     *
     * @see org.eclipse.ui.plugin.AbstractRcpApplicationPlugin#
     *                  start(org.osgi.framework.BundleContext)
     */
    public void start(BundleContext context) throws Exception
    {
        super.start(context);

        // Add the service manager listener
        serviceManager = RcpRuntimePlugin.getDefault().getServiceManager();
        serviceManager.addServiceManagerListener(this);
    }

    /*
     * (non-Javadoc)
     *
     * @see org.eclipse.ui.plugin.AbstractRcpApplicationPlugin#stop(org.osgi.
     *                                                  framework.BundleContext)
     */
    public void stop(BundleContext context) throws Exception
    {
        plugin = null;

        // Remove the service manager listener
        if (serviceManager != null)
            serviceManager.removeServiceManagerListener(this);

        super.stop(context);

    }
```

```
    /**
     * Returns the shared instance
     *
     * @return the shared instance
     */
    public static PrintExamplePlugin getDefault()
    {
        return plugin;
    }

    /**
     * Returns an image descriptor for the image file at the given plug-in
     * relative path
     *
     * @param path
     *            the path
     * @return the image descriptor
     */
    public static ImageDescriptor getImageDescriptor(String path)
    {
        return imageDescriptorFromPlugin(PLUGIN_ID, path);
    }

    /*
     * (non-Javadoc)
     *
     * @see com.ibm.hats.runtime.IPrintJobManagerChangeListener#
     *         addPrintJob(com.ibm.hats.runtime.PrintJobManager,
     *         com.ibm.hats.runtime.PrintJob)
     */
    public void addPrintJob(PrintJobManager printJobManager,
                                            PrintJob printJob)
    {
        System.out.println("PrintExamplePlugin: addPrintJob:
                                      " + printJob.toString());
    }

    /*
     * (non-Javadoc)
     *
     * @see com.ibm.hats.runtime.IPrintJobManagerChangeListener#
     *      removePrintJob(com.ibm.hats.runtime.PrintJobManager,
     *      com.ibm.hats.runtime.PrintJob)
     */
    public void removePrintJob(PrintJobManager printJobManager,
                                               PrintJob printJob)
    {
        System.out.println("PrintExamplePlugin: removePrintJob: "
+ printJob.toString());
    }

    /*
     * (non-Javadoc)
     *
     * @see com.ibm.hats.runtime.IPrintJobManagerChangeListener#
     *      updatePrintJob(com.ibm.hats.runtime.PrintJobManager,
     *      com.ibm.hats.runtime.PrintJob)
     */
    public void updatePrintJob(PrintJobManager printJobManager,
                                                 PrintJob printJob)
    {
        System.out.println("PrintExamplePlugin: updatePrintJob: "
                                            + printJob.toString());
    }

    /*
```

```
 * (non-Javadoc)
 *
 * @see com.ibm.hats.runtime.events.ServiceManagerListener#
 *      serviceChanged(com.ibm.hats.runtime.events.ServiceManagerEvent)
 */
public void serviceChanged(ServiceManagerEvent event)
{
    IService theService = event.getService();

    if (theService instanceof ISessionService)
    {
        ISessionService sessionService = (ISessionService) theService;

        int type = event.getType();
        if (type == ServiceManagerEvent.TYPE_SERVICE_CREATED)
        {
            sessionService.addSessionServiceListener(this);
        }
        else if (type == ServiceManagerEvent.TYPE_SERVICE_DESTROYED)
        {
            sessionService.removeSessionServiceListener(this);
        }
    }
}

/*
 * (non-Javadoc)
 *
 * @see com.ibm.hats.runtime.events.SessionStateListener#
 *      sessionStateChanged(com.ibm.hats.runtime.events.SessionStateEvent)
 */
public void sessionStateChanged(StateChangeEvent event)
{
    // If the session has change to operational then add a listener to
    // the PrintJobManager (if there is one and we haven't already done so).
    StateType state = event.getNewState();
    if (state == StateType.OPERATIONAL)
    {
        ISessionService sessionService = event.getSessionService();
        ClientSpecificInfo csi = sessionService.getRuntimeService()
             .getClientContainer().accessClient(sessionService.getClientId());

        if (csi != null)
        {
            String asiId = ApplicationSpecificInfo.
                                    createCompositeAsiId(sessionService.
                                    getApplicationId(),
                                              sessionService.getViewId());
            ApplicationSpecificInfo asi = csi.peekAll(asiId);
            if (asi != null)
            {
                PrintSpecificInfo psi = asi.getPrint();
                if (psi != null)
                {
                    PrintResourceHandler prh = psi.getResourceHandler();
                    if (prh != null)
                    {
                        PrintJobManager pjm = prh.getPrintJobManager();

                        // Add this PrintJobManager to our list of
                        // PrintJobManagers.
                        // This will return true if we haven't encountered
                        // this PrintJobManager before.
                        // In this case, add ourselves as a
                        // listener to this PrintJobManager.
                        if (printJobManagers.add(pjm))
                        {
```

```
                                pjm.addChangeListener(this);
                            }
                        }
                    }
                }
            }
        }

        /*
         * (non-Javadoc)
         *
         * @see com.ibm.hats.runtime.events.ISessionServiceListener#
         *      aboutToProcessCommand(com.ibm.hats.runtime.events.CommandEvent)
         */
        public void aboutToProcessCommand(CommandEvent event)
        {
        }

        /*
         * (non-Javadoc)
         *
         * @see com.ibm.hats.runtime.events.ISessionServiceListener#
         *      afterProcessCommand(com.ibm.hats.runtime.events.CommandEvent)
         */
        public void afterProcessCommand(CommandEvent event)
        {
        }
}
```

アクションの表示で使用するカスタム・コンポジットの作成

「コンポジットを表示」アクションで参照される SWT コンポジットは、com.ibm.hats.rcp.transform.IRenderable インターフェースをインプリメントする必要があります。SWT コンポジットが com.ibm.hats.rcp.transform.IRenderable をインプリメントしている場合、このコンポジットは特に、コンポジットが表示されるセッションの IsessionService を含む RcpContextAttributes にアクセスできます。ISessionService の `sendContinue()` メソッドは、HATS アプリケーションの実行を再開するために呼び出すことができます。

Visual Editor を使用してコンポジットを作成するには、以下を実行します (21 ページの『変換の編集』を参照)。

1. 「ファイル」>「新規」>「その他」>「**Java**」>「ビジュアル・クラス」を選択して、「新規 **Java** ビジュアル・クラス」ウィザードを開始します。
2. プラグイン・プロジェクトの `src` フォルダーを選択し、新規コンポジット・クラスの名前 (例えば、`MyComposite`) を入力します。
3. 「スタイル」ツリーで、「**SWT**」>「コンポジット」を選択します。
4. 「追加」をクリックして、com.ibm.hats.rcp.transform.IRenderable インターフェースを選択します。
5. クラスを作成するには、「終了」をクリックします。

Visual Editor を使用して続行ボタンをコンポジットに追加するには、以下を実行します (21 ページの『変換の編集』を参照)。

1. 「パレット」ビューから、「**SWT** コントロール」ドロワーを開き、「ボタン」を選択します。

2. ボタンをコンポジットにドラッグ・アンド・ドロップします。プロンプトが出たら、変数のデフォルト名を受諾します。
3. 選択されたボタンを右クリックして「テキストの設定」を選択します。ボタンの表題を入力します。例えば、「OK」または「**サブミット**」と入力します。「**OK**」をクリックします。
4. ボタンを再度右クリックして「イベント」>「イベントの追加」を選択します。「選択」>「**widgetSelected**」を選択し、「終了」をクリックします。
5. `widgetSelected(SelectionEvent)` メソッドの一時コンテンツを、以下と置き換えます。

```
RcpContextAttributes contextAttributes =
                          (RcpContextAttributes)getContextAttributes();
ISessionService sessionService = contextAttributes.getSessionService();
                                 sessionService.sendContinue();
```

注: クラスのインポート・セクションを更新する必要がある場合があります。これを行うには、CTRL+SHIFT+O、またはクラスのソースを右クリックし、「ソース」>「インポートの編成」を選択します。

第 7 章 ビジネス・ロジックの統合

ビジネス・ロジックは、ホスト画面の認識や HATS アプリケーションの起動などのイベントの発生時にアクションとして呼び出される、任意の Java コードです。ビジネス・ロジックはアプリケーションに固有のものであり、HATS の一部として提供されるものではありません。ユーザーはビジネス・ロジックを使用して、HATS アプリケーションを拡張し、データベースなどの他のデータ・ソースと統合することができます。例えば、ファイルやデータベースのコンテンツを HATS グローバル変数へ読み取ることや、グローバル変数を使用して、アプリケーションの GUI ページで使用するドロップダウン・リストやポップアップに入力することができます。

ビジネス・ロジックを作成してプロジェクトに追加するには、「ビジネス・ロジックの作成」ウィザードを使用します。このウィザードを起動するには、HATS Toolkit の「**HATS** プロジェクト」タブを右クリックして、「新規 **HATS**」>「ビジネス・ロジック」とクリックします。

「ビジネス・ロジックの作成」ウィザードで、ビジネス・ロジックを追加するプロジェクトを指定し、Java クラス名を入力します。デフォルトのパッケージ名は、*projectName*.businessLogic ですが、これは Studio の「設定」で変更することができます。任意で、パッケージ名を指定するか、「参照」をクリックして既存の Java パッケージを選択することができます。プロジェクトのグローバル変数に簡単にアクセスするためのメソッドをビジネス・ロジックに組み込むか、またはプロジェクトのグローバル変数を除去する場合は、「グローバル変数ヘルパー・メソッドの作成」チェック・ボックスを選択します。必要な情報を入力したら、「完了」をクリックします。

ビジネス・ロジック・クラスの作成後、それを 1 つ以上の画面イベントやアプリケーション・イベントにリンクして、そのイベントの発生時に実行させることができます。リンクの追加先の各イベント (アプリケーション・イベントまたは画面カスタマイズ) を編集します。「アクション」タブの「追加」をクリックして、「ビジネス・ロジックを実行」を選択し、ビジネス・ロジック・クラスの詳細を入力します。画面カスタマイズの編集とイベントの編集のいずれについても、「*HATS* ユーザーと管理者のガイド」を参照してください。

プロジェクト内のビジネス・ロジック・ファイルを見るには、HATS Toolkit の「**HATS** プロジェクト表示」タブ上のソース・フォルダーを展開します。ソース・フォルダー内の個々のパッケージ名またはクラス名が表示されます。Java クラス名を参照するには、パッケージ名フォルダーを展開します。クラスを編集するには、クラス名をクリックします。ソース・フォルダーには、HATS プロジェクトにインポートされたその他の Java ファイルも組み込むことができます。

「ビジネス・ロジックの作成」ウィザードを使用してビジネス・ロジックを作成する場合、デフォルトでは、実行アクションによって起動されるメソッドに **execute** という名前が付けられます。独自のクラスを作成する場合は、メソッドに以下の属性がなければなりません。

- public および static としてマーク付けされている。
- 戻りの型が void である。
- com.ibm.hats.common.IBusinessLogicInformation オブジェクトを唯一のパラメーターとして受け入れる。

メソッドは、次のように上の形式を使用して、この後にユーザー独自のビジネス・ロジック・コードを続ける必要があります。

```
public static void myMethod (IBusinessLogicInformation businesslogic)
```

カスタム Java コードに渡される IBusinessLogicInformation オブジェクトにより、HATS プロジェクトの各種オブジェクトおよび設定にアクセスし、それらを使用または変更することができます。この種のオブジェクトや設定には以下のものがあります。

- com.ibm.hats.runtime.IRequest クラス。このクラスは、HATS ランタイムに対してなされた要求を表すオブジェクトを返して、パラメーターを要求するためのアクセスを提供します。
- com.ibm.hats.runtime.IResponse クラス。このクラスは、HATS ランタイムからの応答を表すオブジェクトを返します。
- `getConnectionMap()` メソッド。アプリケーションに関して指定した接続情報の設定を含む java.util.Map を返します。
- `getGlobalVariables()` メソッド。このアプリケーション・インスタンス用のグローバル変数の java.util.Hashtable を返します。このテーブルには、共用グローバル変数は含まれていません。
- `getSharedGlobalVariables()` メソッド。このアプリケーション・インスタンス用の共用グローバル変数の java.util.Hashtable を返します。
- クラス・プロパティー。これは、コンポーネントやウィジェットなどのオブジェクトに関するデフォルト設定を提供します。
- com.ibm.hats.common.HostScreen オブジェクト。これにはホスト画面情報が含まれています。
- クライアントの java.util.Locale クラス
- com.ibm.hats.common.TextReplacementList の値と設定
- クライアント・セッション ID ストリング (「ビジネス・ロジックの作成」ウィザードが提供したビジネス・ロジックのテンプレート内の getter メソッドによって戻されたもの)
- 双方向セッションの場合の現在の画面方向
- 双方向セッションの場合のブラウザー内の「画面を反転」ボタンの有無

使用可能なクラスの詳細については、HATS Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSXKAY_9.6.0) にある HATS API 資料に記載されている IBusinessLogicInformation クラスを参照してください。 IBusinessLogicInformation は、IBaseInfo クラスを拡張するため、これらの API のいくつかは IBaseInfo クラスで定義されます。

他のアプリケーションからの Java コードの組み込み

他の既存のアプリケーションから HATS プロジェクトに Java コードを組み込むには、いくつか方法があります。

既存のビジネス・ロジックからソース・コード (.java ファイル) を組み込んで、そのコードを変更できるようにする場合は、既存プロジェクトのソース・フォルダーに .java ファイルをインポートすることができます。「ファイル」>「インポート」>「一般」>「ファイル・システム」をクリックして、「インポート」ウィザードを開きます。「インポート」ウィザードでは、「ソース・ディレクトリー」フィールドでソース・ファイルの場所を選択します。RCP プロジェクトの場合は、「宛先フォルダー」入力フィールドで、プロジェクトの **src** フォルダーを選択します。インポートされたソース .java ファイルは自動的にコンパイルされて、HATS プロジェクトに組み込まれます。Rational SDP ワークベンチで、ソース・ファイルの編集、ブレークポイント設定、デバッグが可能です。

コンパイル済みの Java ビジネス・ロジックが含まれる Java アーカイブ・ファイル (.jar) を組み込むこともできます。 Java アーカイブ全体が追加されます。個々のクラスを選択して追加することはできません。

.jar ファイルを RCP プロジェクトにインポートするには、以下の手順を使用します。

1. ディレクトリーを作成します (通常「lib」または「runtime」) (オプション)。
2. 「ファイル」>「インポート」>「一般」>「ファイル・システム」とクリックして「インポート」ウィザードを開き、.jar ファイルをインポートします。インポート先ディレクトリーには、ステップ 1 で作成したものを選択します。ステップ 1 で説明したとおりにディレクトリーを作成していない場合には、プロジェクトのルートにインポートします。
3. ナビゲーター・ビューに移動して、META-INF ディレクトリー内の MANIFEST.MF ファイルを見つけます。
4. このファイルをダブルクリックして、マニフェスト・エディターで開きます。
5. 「ランタイム」タブへ移動し、「クラスパス」セクションの下の「追加**...**」を選択します。
6. インポートした .jar ファイルを参照して、「ビルド・パスの更新」にチェック・マークを付け、「**OK**」をクリックします。
7. MANIFEST.MF ファイルを保存します。

ビジネス・ロジックでのグローバル変数の使用

HATS アプリケーションでグローバル変数を使用して情報を保管している場合は、それらのグローバル変数をビジネス・ロジックで使用できます。

グローバル変数にはローカルと共用の 2 つのタイプがあります。ローカル・グローバル変数は、特定の RCP プロジェクト内で作成され、そのプロジェクトでのみ使用可能な変数です。共用グローバル変数は、1 つ以上の RCP プロジェクト内で作成され、同じユーザー環境内で実行中のすべてのアプリケーションで使用可能な変数です。また、HATS グローバル変数には 2 つのリストがあります。1 つはローカ

ル・グローバル変数用で、もう 1 つは共用グローバル変数用です。同じ名前を持つ 2 つのグローバル変数がある場合、1 つがローカルでもう 1 つが共用であれば共存が可能です。

ビジネス・ロジック・クラスを作成する際は、「ビジネス・ロジックの作成」ウィザードを使用して、「グローバル変数ヘルパー・メソッドの作成」チェック・ボックスを選択します。これによって、ビジネス・ロジックに、ローカル変数および共用グローバル変数を取得、設定、および除去するためのメソッドが作成されます。

以下のメソッドが作成されます。

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
/**
 * Example method that sets a named global variable
 * from the current session to a value
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the global variable
 * @param value - Value of the global variable
 */
public static void setGlobalVariable(IBusinessLogicInformation blInfo,
     String name,Object value)
{
  IGlobalVariable gv = blInfo.getGlobalVariable(name);
  if ( gv == null )
   {
    gv = new GlobalVariable(name,value);
   }
  else
   {
    gv.set(value);
   }
  blInfo.getGlobalVariables().put(name,gv);
}

/**
 * Example method that sets a named shared
 * global variable from the current session to a value
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the shared global variable
 * @param value - Value of the shared global variable
 */
public static void setSharedGlobalVariable(IBusinessLogicInformation
  blInfo,String name,Object value)
{
  IGlobalVariable gv = blInfo.getSharedGlobalVariable(name);
  if ( gv == null )
   {
    gv = new GlobalVariable(name,value);
   }
  else
   {
    gv.set(value);
   }
  blInfo.getSharedGlobalVariables().put(name,gv);
}

/**
 * Example method that removes a named global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the global variable
*/
public static void removeGlobalVariable(IBusinessLogicInformation blInfo, String  name)
```

```
{
  IGlobalVariable gv = blInfo.getGlobalVariable(name);
  if ( gv != null )
   {
    blInfo.getGlobalVariables().remove(name);
    gv.clear();
    gv = null;
   }
}
/**
 * Example method that removes a named shared global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the shared global variable
*/
public static void removeSharedGlobalVariable(IBusinessLogicInformation blInfo, String name)
{
  IGlobalVariable gv = blInfo.getSharedGlobalVariable(name);
   if ( gv != null )
    {
     blInfo.getSharedGlobalVariables().remove(name);
     gv.clear();
     gv = null;
    }
}
/**
 * Example method that retrieves a named global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the global variable
 * @return - an instance of the global variable, or null if not found.
 */
public static IGlobalVariable getGlobalVariable(IBusinessLogicInformation
  blInfo,String name)
{
  IGlobalVariable gv = blInfo.getGlobalVariable(name);
  return gv;
}

/**
 * Example method that retrieves a named shared
 * global variable from the current session
 * @param blInfo - IBusinessLogicInformation from current session
 * @param name - Name of the shared global variable
 * @return - an instance of the global variable, or null if not found.
 */
public static IGlobalVariable getSharedGlobalVariable(IBusinessLogicInformation
  blInfo,String name)
{
  IGlobalVariable gv = blInfo.getSharedGlobalVariable(name);
  return gv;
}
```

コード内の別の部分でローカル・グローバル変数の値が必要になった場合は、このメソッドを呼び出すことができます。

```
GlobalVariable gv1 = getGlobalVariable(blInfo,"varname");
```

共用グローバル変数の値を取得するには、次のメソッドを使用します。

```
GlobalVariable gv1 = getSharedGlobalVariable(blInfo,"varname");
```

ビジネス・ロジックの例

ここでは、ビジネス・ロジックの使用例を示します。各例では、グローバル変数を扱います。それぞれの例で前述のグローバル変数ヘルパー・メソッドを 1 つ以上使用するため、クラスにそれらのメソッドを組み込む必要があります。ここでは、コード例を見やすくするため、スタブが省略されています。

例: 日付変換

この例では、日付を mm/dd/yy の形式から月、日、年の形式に変換します。例えば、6/12/2004 は June 12, 2004 に変換されます。この例では、ビジネス・ロジッ

クが呼び出される前にグローバル変数 *theDate* が設定されていることを前提としています。この例では、以下のメソッドをどのように使用して入力変数の値を取得しているかに注意してください。

```
IGlobalVariable inputDate = getGlobalVariable(blInfo, "theDate");
```

この例では、標準の Java 関数を使用して日付を希望の形式で表示するようにストリングを操作した後で、以下のメソッドを使用して同じグローバル変数に新規ストリングを組み込んでいます。

```
setGlobalVariable(blInfo, "theDate", formattedDate);
```

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.GlobalVariable;
import com.ibm.hats.common.IGlobalVariable;

public class CustomDateFormatter
{
   public static void execute(IBusinessLogicInformation blInfo)
   {
    IGlobalVariable inputDate = getGlobalVariable(blInfo, "theDate");
    SimpleDateFormat inputFormat = new SimpleDateFormat("MM/dd/yyyy");
    SimpleDateFormat outputFormat = new SimpleDateFormat("MMMM dd, yyyy");

    try
     {
      Date tempDate = inputFormat.parse(inputDate.getString().trim());
      String formattedDate = outputFormat.format(tempDate);
      setGlobalVariable(blInfo, "theDate", formattedDate);
     }
      catch (ParseException ex)
       {
        ex.printStackTrace();
       }
   }
}
```

例: 索引付きグローバル変数に含まれている値の追加

この例では、索引付きグローバル変数に含まれている値を追加し、別の索引なしグローバル変数に合計を保管します。これは、数値を表すストリングが索引付きグローバル変数 *subtotals* に保管されていることを前提としています。

前の例では、入力グローバル変数と出力グローバル変数 (*theDate*) の名前が set 呼び出しに含まれていました。この例では、入力変数と出力変数の名前をローカル・ストリング変数内に設定し、それらのストリングを呼び出しで使用してグローバル変数値を取得および設定します。グローバル変数の名前は変数として渡されているため、引用符で囲まれていません。

```
        setGlobalVariable(blInfo,gvOutputName, new Float(myTotal));
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.GlobalVariable;
import com.ibm.hats.common.IGlobalVariable;

  public static void execute(IBusinessLogicInformation blInfo)
    {
     // Name of indexed global variable to be read in
        String gvInputName = "subtotals";
        // Name of global variable to be calculated and saved
        String gvOutputName = "total";

        // The indexed global variable where each index is a subtotal to be summed
        GlobalVariable gvSubtotals =
     ((GlobalVariable)getGlobalVariable(blInfo, gvInputName));

        float myTotal = 0;

        // Calculate the total by adding all subtotals
        for (int i = 0; i < gvSubtotals.size(); i++)
        {
         myTotal = myTotal + Float.valueOf(gvSubtotals.getString(i)).floatValue();
        }

        // Save the total as a non-indexed local variable
        setGlobalVariable(blInfo,gvOutputName, new Float(myTotal));
    }
```

例: ファイルから索引付きグローバル変数へのストリング・リストの読み取り

この例では、ファイル・システムからファイルを読み取り、ファイル内のストリングを索引付きグローバル変数に保管します。こうした技法を使用して、例えば会社の場所のリストを含むファイルを読み取ることができます。グローバル変数にストリングを保管した後、そのグローバル変数を使用してドロップダウン・リストやその他のウィジェットに値を入力し、ユーザーが値のリストから選択できるようにすることができます。適切な入力フィールドがあるところではどこでも、このウィジェットを使用するためのグローバル規則を作成できます。アプリケーションが起動するとすぐにグローバル変数が使用可能になるようにするには、このビジネス・ロジック・クラスの実行アクションを Start イベントに追加します。

注: テキスト・ファイルが行間に復帰および改行を使用している場合、次の例の StringTokenizer コンストラクターの呼び出しで 2 番目の引数として「¥r¥n」を使用する必要があります。

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
```

```
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
import com.ibm.ejs.container.util.ByteArray;
import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.GlobalVariable;
import com.ibm.hats.common.IGlobalVariable;

public class ReadNamesFromFile
{

  public static void execute(IBusinessLogicInformation blInfo)
  {
  // Name of indexed global variable to be saved
  String gvOutputName = "namesFromFile";

  // The file containing a list of information (in this case, it contains names)
  java.io.File myFileOfNames = new java.io.File("C:" + java.io.File.separator
       + "temp" + java.io.File.separator + "names.txt");

     try
     {
   // First, read the contents of the file

   java.io.FileInputStream fis = new java.io.FileInputStream(myFileOfNames);

   int buffersize = (int)myFileOfNames.length();
   byte[] contents = new byte[buffersize];;

   long n = fis.read(contents, 0, buffersize);

   fis.close();

   String namesFromFile = new String(contents);

   // Next, create an indexed global variable from the file contents

   java.util.StringTokenizer stok =
      new java.util.StringTokenizer(namesFromFile, "¥n", false);

   int count = stok.countTokens();
   String[] names = new String[count];
   for (int i = 0; i < count; i++)
   {
    names[i] = stok.nextToken();
   }

   IGlobalVariable gv = new GlobalVariable(gvOutputName, names);
   blInfo.getGlobalVariables().put(gvOutputName,gv);
     }
     catch (java.io.FileNotFoundException fnfe)
     {
      fnfe.printStackTrace();
     }
     catch (java.io.IOException ioe)
     {
      ioe.printStackTrace();
     }
  }
}
```

カスタム画面認識の使用

ビジネス・ロジックを使用して、カスタム画面認識を実行できます。 HATS Toolkit では、画面カスタマイズに画面認識のオプションが多数用意されています。これには、画面上のフィールド数、ストリングの有無、グローバル変数の使用などがあります。これらのオプションについては、「*HATS* ユーザーと管理者のガイド」を参照してください。しかし、画面カスタマイズ・エディターのオプションでは設定できないような方法で画面を認識したい場合もあるでしょう。そのような場合は、独自のカスタム画面認識論理を追加できます。

注: ここで説明する情報は、画面カスタマイズだけでなくマクロでの画面認識にも適用できます。

HATS グローバル変数を使用してカスタム画面認識論理を作成する方法については、 67 ページの『グローバル変数を使用したカスタム画面認識』を参照してください。

既に ECLCustomRecoListener クラスを拡張してカスタム画面認識論理を作成してある場合は、その論理を HATS 内で使用できます。新規カスタム論理を作成する場合は、次の手順に従ってください。

1. Java パースペクティブを開きます。
2. 「ファイル」>「新規」>「クラス」とクリックします。
3. HATS プロジェクトのソース・ディレクトリーを参照します。
4. パッケージとクラスの名前を入力します。
5. スーパークラスの場合は、「参照」をクリックして、com.ibm.hats.common.customlogic.AbstractCustomScreenRecoListener を見つけます。
6. 「継承された抽象メソッド」チェック・ボックスを選択します。「完了」をクリックします。これにより、指定したプロジェクトにコード・スケルトンがインポートされます。
7. `isRecognized` メソッドに論理を追加します。ブール値が戻されることを確認します。

   ```
   public boolean isRecognized(String arg0, IBusinessLogicInformation
   arg1, ECLPS arg2, ECLScreenDesc arg3)
   ```

   このメソッドの詳細については、HATS Knowledge Center で HATS API の資料を参照してください。
8. メソッドの作成後、画面認識を更新してメソッドを呼び出す必要があります。「HATS プロジェクト」ビューから、プロジェクトと「画面カスタマイズ」フォルダーを展開します。カスタム論理を追加する画面カスタマイズの名前をダブルクリックします。「ソース」タブをクリックして画面カスタマイズのソース・ビューを開きます。
9. ソース・ビュー内に、<description> タグで始まり </description> タグで終わるブロックが表示されます。このブロックには、画面の認識に使用される情報が含まれています。カスタム論理を起動するには、このブロック内に行を追加します。

   ```
   <customreco id="customer.class.package.MyReco::settings"
   invertmatch="false" optional="false"/>
   ```

   ここで、customer.class.package.MyReco はご使用のパッケージとクラス名です。クラスに設定値を渡す場合は、クラス名の後に 2 つのコロンで区切って設定値を追加します。設定値はオプションです。クラスは、渡された値をすべて解析する必要があります。設定値が必要ない場合は、2 つのコロンを省略します。

説明ブロック内のどこに <customreco> タグを置くかを検討します。他のすべての基準が一致した場合だけカスタム論理を起動するには、ブロックの終わりの </description> タグの直前に <customreco> タグを置きます。画面カスタマイズで画面領域と値を比較する場合、説明ブロックには <block> タグと </block> タグで囲まれた、より小さいブロックが含まれ、画面領域と比較する値が定義されます。このブロック内には customreco タグを置かないでください。

説明ブロックの例を次に示します。<customreco> タグは </description> タグの直前にあり、<block タグと </block> タグの間にはないことに注意してください。

```
<description>
<oia invertmatch="false" optional="false" status="NOTINHIBITED"/>
<numfields invertmatch="false" number="61" optional="false"/>
<numinputfields invertmatch="false" number="16" optional="false"/>
<block casesense="false" col="2" ecol="14" erow="21"
   invertmatch="false" optional="false" row="20">
<string value="USERID ==="/>
<string value="PASSWORD ==="/>
</block>
<cursor col="16" invertmatch="false" optional="false" row="20"/>
<customreco id="customer.class.package.MyReco::settings"
   invertmatch="false" optional="false"/>
</description>
```

10. HATS プロジェクトをビルドし直すには、ツールバーの「プロジェクト」> 「クリーン」をクリックします。
11. RCP の場合は、ご使用のローカル・テスト環境でアプリケーションを実行して、プロジェクトをテストします。詳細については、「*HATS* スタートアップ・ガイド」を参照してください。

カスタム画面認識の例

以下に、カスタム画面認識を実行するビジネス・ロジックの例を示します。このビジネス・ロジック・クラスは、ブランクで区切られたコード・ページ番号のリストを設定値としてとり、設定値にリストされた値のいずれかとコード・ページが一致した場合に画面を認識します。タグ構文は次のとおりです。

```
<customreco id="company.project.customlogic.CodePageValidate::[settings]"
optional="false" invertmatch="false" />
```

例えば、次のタグを説明ブロックに挿入することができます。

```
<customreco id="company.project.customlogic.CodePageValidate::037 434 1138"
optional="false" invertmatch="false" />
```

この場合は、コード・ページが 037、434、または 1138 であるときに画面が認識されます。

```
////////////////////////////////////////////////////////////////////////////
//  This sample is provided AS IS.
//  Permission to use, copy and modify this software for any purpose and
//  without fee is hereby granted. provided that the name of IBM not be used in
//  advertising or publicity pertaining to distribution of the software without
//  specific written permission.
//
//  IBM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SAMPLE, INCLUDING ALL
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IBM
//  BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
//  DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER
//  IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING
```

```
//  OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SAMPLE.
////////////////////////////////////////////////////////////////////////////
package company.project.customlogic;

import java.util.StringTokenizer;
import java.lang.Integer;
import com.ibm.eNetwork.ECL.ECLPS;
import com.ibm.eNetwork.ECL.ECLScreenDesc;
import com.ibm.hats.common.IBusinessLogicInformation;
import com.ibm.hats.common.HostScreen;
import com.ibm.hats.common.customlogic.AbstractCustomScreenRecoListener;

public class CodePageValidate extends AbstractCustomScreenRecoListener {

/**
 * @see com.ibm.hats.common.customlogic.AbstractCustomScreenRecoListener
 * #isRecognized(java.lang.String, com.ibm.hats.common.IBusinessLogicInformation,
 *       com.ibm.eNetwork.ECL.ECLPS,  com.ibm.eNetwork.ECL.ECLScreenDesc)
  */
 public boolean isRecognized(
  String settings,
  IBusinessLogicInformation bli,
  ECLPS ps,
  ECLScreenDesc screenDescription) {
  HostScreen hs=bli.getHostScreen();
  int int_codepage=hs.GetCodePage();
  if(settings!=null)
  {
   StringTokenizer tokenizer = new StringTokenizer(settings);
   while( tokenizer.hasMoreTokens() )
   {
    int int_token= Integer.valueOf(tokenizer.nextToken()).intValue();
    if ( int_token==int_codepage )
    {
      return true;
    }
   }
  }
  return false;
 }

}
```

グローバル変数を使用したカスタム画面認識

HATS Toolkit には、次に示すように、グローバル変数を使用した画面認識オプションがいくつか用意されています。

- グローバル変数が存在することを検証
- グローバル変数が存在しないことを検証
- グローバル変数の整数またはストリング値を検証

これらのオプションについては、「*HATS* ユーザーと管理者のガイド」を参照してください。HATS グローバル変数に基づいて画面認識を実行する場合、「グローバル変数論理 (Global Variable Logic)」パネル内のオプションが要件に合わなければ、1 つ以上のグローバル変数の値または存在に基づいて独自の論理を追加できます。このアプローチでは、Java クラスを作成する必要はありません。代わりに、HATS が提供する GlobalVariableScreenReco クラスを使用します。<customreco> タグで比較を設定値として指定できます。フォーマットは次のいずれかになります。

- ```
  <customreco
  id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name,option,resource, index)}COMPARE{type
  (name,option,resource,
                                              index)}"
    invertmatch="false" optional="false"/>
  ```
  ```

- <customreco
  id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name,option,resource, index)}COMPARE{type(value)}"
  invertmatch="false" optional="false"/>

比較される 2 つの項目それぞれを含む中括弧 {} が使用されます。最初の項目は、*name* で名前が指定された HATS グローバル変数です。option を使用して、比較で変数の値、長さ、存在を使用することを指定できます。resource と index の設定はオプションです。resource を使用すると、グローバル変数がローカル (これがデフォルトです) か共用かを示すことができます。index を使用すると、索引付きグローバル変数から使用する値を示すことができます。

2 番目の項目は次のいずれかです。

- 同様のオプションを持つ別の HATS グローバル変数。この場合は、最初のフォーマットが使用されます。
- 固定値。この場合は、2 番目のフォーマットが使用されます。

設定で有効な値が表 17 に表示されます。 COMPARE 設定の場合、ストリングの比較に有効な値は、EQUAL と NOTEQUAL のみです。

表 *17.* 設定の有効値

設定	有効値
type	• variable • integer • boolean • string
COMPARE	• EQUAL • NOTEQUAL • GREATERTHAN • GREATERTHANOREQUAL • LESS THAN • LESSTHANOREQUAL
オプション	• exists (ブール値) • value (ストリング/整数/ブール値) • length (整数) • object (オブジェクト)
resource	• local • shared
index	任意の正の整数または 0

次の例では、2 つのローカル・グローバル変数の値を比較します。

```
<customreco id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name=gv1,option=value,resource=local)}EQUAL
  {variable(name=gv2,option=value,resource=local)}"
  invertmatch="false" optional="false"/>
```

gv1 と *gv2* の値が同じである場合、この式は true と評価されます。

次に、長さのオプションについて考えます。索引のないグローバル変数の場合、長さは変数の値の長さになります。索引付きグローバル変数の場合は、索引を指定すると、長さはグローバル変数のその索引の長さになります。索引を指定しないと、長さはグローバル変数内の索引付きエントリーの数になります。

```
<customreco id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name=gv1,option=length,resource=shared)}LESSTHANOREQUAL
  {variable(name=gv2,option=length,index=4)}" invertmatch="false" optional="false"/>
```

この式は、*gv1* の長さと *gv2* の 4 番目の索引の長さを比較します。*gv1* の長さが *gv2* の 4 番目の索引の長さと等しいかそれより小さい場合、この式は true と評価されます。長さは整数値を戻すので、LESSTHANOREQUAL を使用できます。

この例の *gv1* に対する resource=shared の使用は、*gv1* が共用グローバル変数であることを示しています。もう 1 つのオプションは、デフォルトの resource=local です。これは、グローバル変数がその他のアプリケーションと共有されないことを意味します。

あるグローバル変数を別のグローバル変数と比較する必要はありません。グローバル変数の長さは固定整数と比較できます。グローバル変数の値は別のストリングと比較できます。例:

```
<customreco id="com.ibm.hats.common.customlogic.GlobalVariableScreenReco::
  {variable(name=gv1,option=value)}EQUAL{string(value=mystring)}"
  invertmatch="false" optional="false"/>
```

この式は、*gv1* の値を mystring に含まれているストリングと比較します。このストリングは、固定されたストリング、変数の値、またはメソッド呼び出しから戻された値です。通常は、グローバル変数の長さまたは値を固定値と比較するためにカスタム論理を使用する必要はありません。これらの比較は、「グローバル変数論理 (Global Variable Logic)」パネルを使用して追加できます。

第 8 章 カスタム・コンポーネントおよびウィジェットの作成

HATS には、ホスト画面のエレメントを認識するホスト・コンポーネントと、この認識されたエレメントをレンダリングするウィジェットのセットが用意されています。デフォルト設定が要求通りにコンポーネントを認識しない場合やウィジェットをレンダリングしない場合、コンポーネントおよびウィジェットの設定を変更できます。HATS が提供するコンポーネント、ウィジェット、および設定がニーズに合わない場合は、独自のカスタム・コンポーネントまたはウィジェットを作成するか、既存のホスト・コンポーネントまたはウィジェットを変更することができます。HATS コンポーネントが認識しないホスト画面のエレメントを認識するために独自のホスト・コンポーネントを作成することもできます。また、ページでのエレメントの表示方法を変更するために独自のウィジェットを作成することもできます。以下のセクションでは、カスタム・ホスト・コンポーネントおよびウィジェットの作成方法について説明します。詳しくは、 87 ページの『付録 A. HATS Toolkit ファイル』を参照してください。

注: 双方向コード・ページを使用している場合は、ウィジェットおよびその他の表示エレメントを制御できます。 81 ページの『第 9 章 HATS 双方向 API の使用』を参照してください。

RCP アプリケーションのコンポーネントおよびウィジェットのプロパティー

RCP プロジェクトの場合は、変換は SWT (Standard Widget Toolkit) コンポジットです。 RCP の変換には、.java ファイル拡張子が付きます。変換は、「リッチ・クライアント・コンテンツ」>「変換」の下の「HATS プロジェクト」ビュー内にあります。変換は、ComponentRendering コンポジット (特殊なコンポジット、または指定されたコンポーネント、ウィジェット、および設定を使用して指定された領域を変換するパネル) から構成されます。以下のコード例で、この概念を示します。

```
ComponentRendering rendering1 = new ComponentRendering(this, 0);
    rendering1.setComponent("<fully qualified component class name>");
    rendering1.setWidget("<fully qualified widget class name>");
    rendering1.setRegion(new BlockScreenRegion(start_row, start_col, end_row, end_col));
    rendering1.setComponentSettings(new StringableProperties("<component settings>"));
    rendering1.setWidgetSettings(new StringableProperties("<widget settings>"));
    rendering1.setHostScreen(getHostScreen());
    rendering1.setTransformInfo(getTransformInfo());
    rendering1.setAutoRender(false);
    rendering1.setTextReplacement("<text replacement string>");
    rendering1.render();
```

上の例で、*component settings* はコンポーネント設定のストリングを意味し、*widget settings* はウィジェット設定のストリングを意味します。

これらのコンポジットは、通常、変換の `render()` メソッド内で構成およびレンダリングされます。

SWT ウィジェットは、RCP プロジェクトのための com.ibm.hats.rcp.transform.renderers.SwtRenderer インターフェースをインプリメントする必要があり、drawSwt メソッドを備えています。

以下の SWT ウィジェット・コードの例は、コンストラクターおよび drawSwt メソッドを示したものです。

```
public MyCustomWidget extends Widget implements SwtRenderer {
    public MyCustomWidget (ComponentElement[] componentElements, Properties settings) {
super(componentElements, settings);
    }

    public Control[] drawSwt(Composite parent) {
        SwtElementFactory elementFactory = SwtElementFactory.newInstance(getContextAttributes(),
        getSettings(), parent.getDisplay());

                // Add code here to generate SWT Control objects here
return controls;  // returns controls added to the specified parent
    }
}
```

カスタム・ホスト・コンポーネントの作成

HATS は「コンポーネントの作成 (Create Component)」ウィザードを提供しており、カスタム・コンポーネントの作成を支援します。いくつかの方法でウィザードを開始できます。

- HATS Toolkit の「ファイル」>「新規」>「**HATS** コンポーネント」メニューから
- HATS Toolkit の「**HATS**」>「新規」>「コンポーネント」メニューから
- HATS プロジェクトのコンテキスト (右クリック) メニューから「新規 **HATS**」>「コンポーネント」を選択

「コンポーネントの作成 (Create Component)」ウィザードには 2 つのパネルがあります。最初のパネルでは、プロジェクト名、新規コンポーネント名、コンポーネントの Java パッケージ名を入力します。オプションで、新規コンポーネントによって使用される設定の構成用の GUI パネルを定義できるスタブ・メソッドを組み込むためのチェック・ボックスを選択することができます (詳しくは、 79 ページの『カスタム・コンポーネントおよびウィジェットの設定に対する HATS Toolkit のサポート』を参照)。2 番目のパネルでは、HATS Toolkit で新規コンポーネント名として表示する名前を入力し、そのコンポーネントに関連付けるウィジェットを選択します。ウィジェットの関連付けはウィザードを完了するうえで必須ではありません。コンポーネントとウィジェットの関連は後で定義することができます。コンポーネントとウィジェットの関連付けについての詳細は、 77 ページの『コンポーネントまたはウィジェットの登録』を参照してください。

次のセクションでは、ウィザードが提供するカスタム・コンポーネントの必須エレメントについて説明します。

- ホスト・コンポーネントの抽象クラス、com.ibm.hats.transform.components.Componentを拡張。

  HATS ホスト・コンポーネントの 1 つがユーザーが必要なものとよく似ている場合、そのコンポーネントの拡張はより容易です。詳しくは、 74 ページの『コンポーネント・クラスの拡張』を参照してください。

- コンストラクター・メソッドを追加。コンポーネントに指定されたこのメソッドは、com.ibm.hats.common.HostScreen オブジェクトを受け入れる必要があります。例:

  ```
  public MyComponent(HostScreen hostScreen) {
    super(hostScreen);
  }
  ```

  コンストラクターは、ホスト画面オブジェクトに基づいて、recognize() メソッドが必要とするパラメーターを初期化します。

- recognize() メソッドの追加。

  ```
  public ComponentElement[] recognize(BlockScreenRegion region,
                                      Properties settings)
  ```

  recognize() メソッドは、ホスト・コンポーネント・クラスごとに別々のインプリメンテーションを用います。これは、渡された領域および設定値を受け入れ、コンポーネント・エレメント・オブジェクトの配列を戻します。ご使用のパターン認識論理をインプリメントするには、このメソッドをインプリメントする必要があります。

  recognize() メソッドは、com.ibm.hats.transform.elements.ComponentElement で定義されているように、ComponentElement オブジェクトの配列を戻す必要があります。HATS コンポーネントは、ComponentElement を拡張する一連のエレメントとして、それぞれやや異なる値を戻します。例えば、SelectionListComponent は、SelectionComponentElement オブジェクトの配列を戻します。このコンポーネント・エレメントの配列は、指定されたウィジェットに渡されるので、使用するウィジェットが受け入れることのできるエレメントの配列が戻されるようにしてください。

  このメソッドの引数については、「HATS API References (Javadoc)」で Component クラスの recognize() メソッドの説明を参照してください。 2 ページの『API 資料の使用 (Javadoc)』を参照してください。

- プロジェクトのソース・フォルダーにコンポーネントのソース・コードを追加。
- Rational SDP ワークベンチのプリファレンス (「ウィンドウ」>「設定」>「一般」>「ワークスペース」または「プロジェクト」>「自動的にビルド」) で、「自動的にビルド」にチェック・マークを付けている場合は、新規コンポーネント .java ファイルをコンパイル。コンポーネントが .class ファイルにコンパイルされていない場合、HATS Toolkit では使用できません。
- ComponentWidget.xml ファイルに新規コンポーネントを登録。コンポーネントの登録の詳細については、 77 ページの『コンポーネントまたはウィジェットの登録』を参照してください。

チェック・ボックスを選択し HATS Toolkit グラフィカル・ユーザー・インターフェースのサポート・メソッドを組み込んだ場合は、コンポーネントの設定変更が可能になるとともに、「コンポーネントの作成」ウィザードで以下のメソッドが追加されます。

- プロパティー設定でページ番号を戻すメソッド。

  ```
  public int getPropertyPageCount() {
    return (1);
  }
  ```

- カスタマイズ可能な設定を戻すメソッド。

```
public Vector getCustomProperties(int iPageNumber, Properties properties,
   ResourceBundle bundle) {
  return (null);
}
```

- カスタマイズ可能な設定のデフォルト値を戻すメソッド。

```
public Properties getDefaultValues(int iPageNumber) {
  return (super.getDefaultValues(iPageNumber));
 }
```

カスタム・コンポーネントのサポートに必要なメソッドの詳細については、 79 ページの『カスタム・コンポーネントおよびウィジェットの設定に対する HATS Toolkit のサポート』を参照してください。

注: デフォルト・レンダリング内でコンポーネントを正しく機能させるには、コンポーネント・エレメントを戻す前に、コンポーネントが戻す各コンポーネント・エレメントで使用された領域 (処理されたホスト画面の領域) を設定する必要があります。これによって、デフォルト・レンダリングは、ホスト・コンポーネントによってこの画面領域が使用 (処理) されたため再び処理すべきではないということを認識します。使用された領域を設定するには、次のメソッドを使用します。

```
public void setConsumedRegion(BlockScreenRegion region)
```

詳しくは、「HATS API References (Javadoc)」で ComponentElement クラスの説明を参照してください。 2 ページの『API 資料の使用 (Javadoc)』を参照してください。

コンポーネント・クラスの拡張

HATS はホスト・コンポーネント・クラスの番号を提供します。`ComponentWidget.xml` ファイル内にあるすべてのホスト・コンポーネントのクラスは、新規コンポーネント用に作成された .java ファイル内のステートメント `public class` *`MyCustomComponent`* `extends Component` を既存コンポーネントのクラス名で置換することによって拡張できます。例:

```
public class MyCustomComponent
             extends com.ibm.hats.transform.components.CommandLineComponent
```

注: 双方向コンポーネントは com.ibm.hats.transform.components.BIDI パッケージに保管されています。コンポーネントの双方向クラスの名前は、通常のコンポーネントの名前と同じですが、例えば com.ibm.hats.transform.components.BIDI.CommandLineComponentBIDI のように最後に「BIDI」が付きます。

各 HATS コンポーネントは、`recognize()` メソッドでホスト画面のエレメントの認識を実行します。ホスト・コンポーネントを拡張し、必要とする特定の認識タスクを実施するには、次のアプローチのいずれかを使用できます。

- HATS が提供するコンポーネント・クラスの 1 つを拡張し、コンポーネントの `recognize()` メソッドをオーバーライドします。`recognize()` メソッドのどこかで、`super.recognize(region, settings);` のような呼び出しを追加し、拡張し

たクラスの recognize() メソッドを呼び出します。設定を変更してからスーパークラスを呼び出すか、スーパークラスによって戻された出力を処理することによって、プロセスを変更します。

- HATS が提供するコンポーネント・クラスの 1 つを拡張し、コンポーネントの recognize() メソッドをオーバーライドします。スーパークラスの recognize() メソッドを使用する代わりに、その他のいずれかのコンポーネント・クラスの recognize() メソッドを呼び出します。このアプローチは、複数の HATS コンポーネントの特徴を結合した複合ホスト・コンポーネントを認識する場合に便利です。

「コンポーネントの作成 (Create Component)」ウィザードでは、null を返す recognize() メソッドが生成されます。これは、新規コンポーネントでホスト画面領域が認識されないことを示します。拡張元の HATS コンポーネント (エレメントには正しい ComponentElements がすべて含まれる) と同じように機能するようにカスタム・コンポーネントを変更するには、.java ファイルから「return null」を削除し、コンポーネント・コード内のコードを変更します。例:

```
public ComponentElement[] recognize(LinearScreenRegion region, Properties settings) {
ComponentElement [] elements = super.recognize(region, settings);
return elements;
}
```

ComponentWidget.xml ファイルを編集するには、HATS Toolkit の「ナビゲーター」タブをクリックします。 ComponentWidget.xml ファイルは、プロジェクトの「ナビゲーター」ビューの下部に表示されます。ComponentWidget.xml ファイルの詳細については、 77 ページの『コンポーネントまたはウィジェットの登録』を参照してください。

カスタム ウィジェットの作成

HATS は「ウィジェットの作成 (Create Widget)」ウィザードを提供しており、カスタム・ウィジェットの作成を支援します。いくつかの方法でウィザードを開始できます。

- HATS Toolkit の「ファイル」>「新規」>「**HATS** ウィジェット」メニューから
- HATS Toolkit の「**HATS**」>「新規」>「ウィジェット」メニューから
- HATS プロジェクトのコンテキスト・メニュー (右マウス・ボタンをクリック) から「新規 **HATS**」>「ウィジェット」を選択

「ウィジェットの作成」ウィザードには 2 つのパネルがあります。最初のパネルでは、プロジェクト名、新規ウィジェット名、ウィジェットの Java パッケージ名を入力します。オプションで、新規ウィジェットによって使用される設定の構成用の GUI パネルを定義できるスタブ・メソッドを組み込むためのチェック・ボックスを選択することができます (詳しくは、 79 ページの『カスタム・コンポーネントおよびウィジェットの設定に対する HATS Toolkit のサポート』を参照)。 2 番目のパネルでは、HATS Toolkit で新規ウィジェット名として表示する名前を入力し、そのウィジェットに関連付けるコンポーネントを選択します。

このウィザードにより、以下のカスタム・ウィジェットの必須エレメントが提供されます。

- ウィジェットの抽象クラスを拡張し、SwtRenderer インターフェースをインプリメント。`public class MyCustomWidget extends Widget implements SwtRenderer`

  詳しくは、 77 ページの『ウィジェット・クラスの拡張』を参照してください。
- コンストラクター・メソッドを追加。

```
public MyCustomWidget(ComponentElement[] arg0, Properties arg1) {
  super(arg0, arg1);
}
```

- 以下のメソッドを追加して、RCP プロジェクト用の SWT 出力を生成。

```
public Control[] drawSwt(Composite parent) {
        SwtElementFactory elementFactory =
                 SwtElementFactory.newInstance(contextAttributes,
                                  settings, parent.getDisplay());

        // Construct controls using SwtElementFactory instance

        return new Control[0];
    }
```

  SwtElementFactory は、新規コントロールの基本的なデータ・モデルへの関連付け、および関連付けられたテンプレートのスタイルの適用を扱うため、SWT コントロールの構成に使用する必要があります。
- プロジェクトのソース・フォルダーにウィジェットのソース・コードを追加。
- Rational SDP ワークベンチのプリファレンス (「ウィンドウ」>「設定」>「一般」>「ワークスペース」) またはプロジェクト・メニューで、「自動的にビルド」を選択している場合は、新規ウィジェット .java ファイルをコンパイル。ウィジェットが .class ファイルにコンパイルされていない場合、HATS Toolkit では使用できません。
- `ComponentWidget.xml` ファイルに新規ウィジェットを登録。ウィジェットの登録の詳細については、 77 ページの『コンポーネントまたはウィジェットの登録』を参照してください。

チェック・ボックスを選択し HATS Toolkit グラフィカル・ユーザー・インターフェースのサポート・メソッドを組み込んだ場合は、ウィジェットの設定変更が可能になるとともに、「ウィジェットの作成」ウィザードで以下のメソッドが追加されます。

- プロパティー設定でページ番号を戻すメソッド。

```
public int getPropertyPageCount() {
  return (1);
}
```

- カスタマイズ可能な設定を戻すメソッド。

```
public Vector getCustomProperties(int iPageNumber, Properties properties,
   ResourceBundle bundle) {
  return (null);
}
```

- カスタマイズ可能な設定のデフォルト値を戻すメソッド。

```
public Properties getDefaultValues(int iPageNumber) {
  return (super.getDefaultValues(iPageNumber));
 }
```

カスタム・ウィジェットのサポートに必要なメソッドの詳細については、 79 ページの『カスタム・コンポーネントおよびウィジェットの設定に対する HATS Toolkit のサポート』を参照してください。

ウィジェット・クラスの拡張

HATS はウィジェット・クラスの番号を提供します。次のコンポーネントを置換して ComponentWidget.xml ファイル内にあるウィジェットのクラスを拡張できます。

```
public class MyCustomWidget extends Widget implements SwtRenderer
```

作成された .java ファイル内のこのウィジェットを、次のような既存のウィジェットのクラス名を持つ新規ウィジェットと置換します。

```
public class MyCustomWidget extends
  com.ibm.hats.rcp.transform.widgets.SwtFieldWidget
```

注: 双方向ウィジェットは com.ibm.hats.transform.widgets.BIDI パッケージに保管されています。ウィジェットの双方向クラスの名前は、通常のウィジェットの名前と同じですが、末尾に「BIDI」が付きます。例えば、次のようになります。

```
public class newBIDIField extends
  com.ibm.hats.rcp.transform.widgets.BIDI.SwtFieldWidgetBIDI implements SwtRenderer
```

既存ウィジェットを変更する場合は、既存ウィジェット・クラスの 1 つを拡張し、その drawSwt メソッドをオーバーライドする必要があります。ウィジェットのインターフェースおよびメソッドについて詳しくは、「HATS API References (Javadoc)」を参照してください。 2 ページの『API 資料の使用 (Javadoc)』を参照してください。

ウィジェットとグローバル規則

入力フィールドを表すウィジェットは、入力フィールドが HATS グローバル規則によって既に処理されているかどうかを確認します。ホスト画面が受信されると、HATS はそこから、その HATS アプリケーションに対して定義されたグローバル規則と一致するホスト・コンポーネントを検索します。入力フィールドが HATS グローバル規則によって既に処理されているかどうかをウィジェットが確認する際、入力フィールドが処理されていない場合、呼び出しはヌルを戻します。入力フィールドがグローバル規則に従って既に処理されている場合、この呼び出しは、グローバル規則によって入力フィールドから変換された変換フラグメントを戻します。ウィジェットは、コンポーネント・エレメントを処理するのではなく、フラグメントを出力しなければなりません。

```
Control control = RcpRenderingRulesEngine.processMatchingElement(parent, fce,
                  contextAttributes);
                     if (control == null) {
                          ...
                      }
```

ウィジェットの drawSwt() メソッドに上記の例を追加してください。

コンポーネントまたはウィジェットの登録

カスタム・コンポーネントおよびウィジェットを ComponentWidget.xml ファイルに登録すると、HATS Toolkit で使用可能になります。例えば、「ホスト・コンポーネントを挿入」ウィザードで使用できます。

ホスト・コンポーネントを、特定のウィジェットにマップする必要があります。カスタム・ホスト・コンポーネントは、既存のウィジェットまたはカスタム・ウィジェットのいずれにもマップすることができます。カスタム・コンポーネントやウィジェットの「作成」ウィザードは、ユーザーのカスタム・コンポーネントとウィジェットを ComponentWidget.xml ファイルに登録し、コンポーネントとウィジェットを関連付けます。カスタム・コンポーネントとウィジェットを関連付けずにウィザードを使用した場合、ComponentWidget.xml ファイルを編集し、関連を追加する必要があります。ComponentWidget.xml ファイルを編集するには、HATS Toolkit の「ナビゲーター」タブをクリックします。 ComponentWidget.xml ファイルは、プロジェクトの「ナビゲーター」ビューの下部に表示されます。

注: 登録後にカスタム・コンポーネントまたはウィジェットを削除する場合、コンポーネントやウィジェットのソース・コードをプロジェクトのソース・フォルダーから削除するだけでは完全に除去することはできません。レジストリー内ではなお参照されており、プログラマチックに除去する方法はありません。
ComponentWidget.xml ファイルを編集してレジストリーから除去し、そのコンポーネントまたはウィジェットへの参照を削除する必要があります。

次に、HATS が提供するフィールド・テーブル・コンポーネントおよび関連ウィジェットの 1 つ、垂直バー・グラフ・ウィジェットを表示する ComponentWidget.xml ファイルの例を示します。

```
<ComponentWidgetList>
  <components>
    <component className="com.ibm.hats.transform.components.FieldTableComponent"
       displayName="Field table" image="table.gif">
      <associatedWidgets>
        <widget className="com.ibm.hats.rcp.transform.widgets.SwtVerticalBarGraphWidget"/>
      </associatedWidgets>
    </component>
  </components>


  <widgets>
    <widget className="com.ibm.hats.rcp.transform.widgets.SwtVerticalBarGraphWidget"
       displayName="Vertical graph" image="verticalBarGraph.gif" />
  </widgets>
</ComponentWidgetList>
```

ご覧のように、このファイルには、コンポーネントとウィジェットの 2 つのセクションがあります。

コンポーネント・セクションには、登録されたコンポーネントの全リストが含まれています。カスタム・コンポーネントを登録し、HATS Toolkit で使用可能にするには、<component> タグと関連の <widget> タグを ComponentWidget.xml ファイルに追加します。className、displayName、および関連ウィジェットを指定する必要があります。

className
ホスト画面のエレメントを認識するコードを含む Java クラスを示します。通常、クラス名の形式は *myCompany.myOrg.ClassName* です。

displayName
カスタム・コンポーネントの既知名と、HATS Toolkit でそれがどのようにコンポーネントのリストに表示されるかを示します。この名前は、登録済み

のコンポーネント間で固有である必要があります。カスタム・コンポーネントの displayName の形式は、単純なストリングです。スペースは、displayName に使用できます。

image image 属性は、HATS Toolkit での表示用に使用するコンポーネントのイメージを示します。

widget
このコンポーネントに関連付けられたウィジェットを識別します。関連付けられた各ウィジェットには、個別の <widget> タグが必要です。コンポーネントの <widget> タグはすべて、<associatedWidgets> タグおよび、その </associatedWidgets> 終了タグの中で定義される必要があります。<associatedWidgets> タグ内の <widget> タグには、className 属性のみがあり、これはウィジェットをコンポーネントにリンクするコードを含む Java クラスを識別します。通常、クラス名の形式は *myCompany.myOrg.ClassName* です。

ウィジェット・セクションには、登録されたウィジェットの全リストが含まれています。ウィジェットを登録し、コンポーネントにリンクし、HATS Toolkit で使用可能にするには、<widget> タグを `ComponentWidget.xml` ファイルに追加します。className および displayName を指定する必要があります。

className
ウィジェットをレンダリングするコードを含む Java クラスを識別します。通常、クラス名の形式は *myCompany.myOrg.ClassName* です。

displayName
カスタム・ウィジェットの既知名と、HATS Toolkit でウィジェットのリストに表示される方法を識別します。この名前は、登録済みウィジェットの中で一意である必要があります。カスタム・ウィジェットの displayName の形式は、単純なストリングです。スペースは、displayName に使用できません。ただし、スペースの代わりに下線 (_) を使用することができます。

カスタム・コンポーネントおよびウィジェットの設定に対する HATS Toolkit のサポート

カスタム・コンポーネントおよびカスタム・ウィジェットの設定の変更に GUI サポートを提供できます。この GUI は、作成したカスタム・コンポーネントまたはカスタム・ウィジェットを他の開発者が使用する場合か、または HATS Toolkit で使用可能なプレビュー機能を使用して、設定の異なる組み合わせのテストを簡単に行う場合に役立ちます。基本コンポーネントおよびウィジェット・クラスにより、ICustomPropertySupplier インターフェースがインプリメントされます。このインターフェースを使用すれば、コンポーネントまたはウィジェットによる HATS Toolkit への設定情報の提供ができます。この情報は、コンポーネントまたはウィジェットの設定を変更できるパネルのレンダリングに使用されます。すべての設定をこの GUI で公開する必要はありません。

`getCustomProperties()` メソッドは、**HCustomProperty** でカスタマイズ可能なプロパティー・オブジェクトのベクトルを戻します。各 **HCustomProperty** オブジェクトは、コンポーネントまたはウィジェットの設定を表します。 HATS Toolkit は、各 **HCustomProperty** オブジェクトをそのタイプに基づいてレンダリングしま

す。例えば、HCustomProperty.TYPE_BOOLEAN タイプのオブジェクトは、GUI のチェック・ボックスとしてレンダリングされます。

以下のサンプル・コードは、ウィジェットがその 3 つの設定 (mySetting1、mySetting2、および mySetting3) に対してどのように GUI サポートを提供できるかを示しています。

```
// Returns the number of settings panels (property pages) to be contributed
//by this widget.  The returned value must be greater than or equal to 1 if
//custom properties will be supplied via the getCustomProperties() method.
public int getPropertyPageCount() {
        return 1;
}

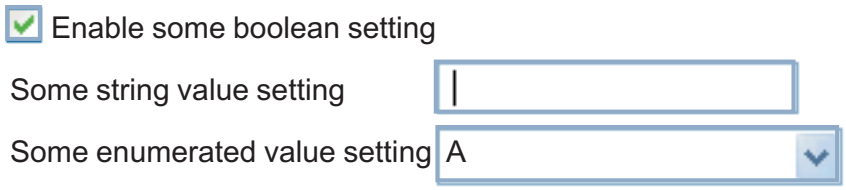
// Returns a Vector (list) of custom properties to be displayed in the GUI
//panel for this component or widget.
public Vector getCustomProperties(int iPageNumber, Properties properties,
        ResourceBundle bundle) {
        Vector props = new Vector();

// Constructs a boolean property that will be rendered as a checkbox
HCustomProperty prop1 = HCustomProperty.new_Boolean("mySetting1",
                "Enable some boolean setting", false, null, null);
props.add(prop1);

// Constructs a string property that will be rendered as a text field
HCustomProperty prop2 = HCustomProperty.new_String("mySetting2",
                "Some string value setting", false, null,
                 null, null, null);
props.add(prop2);

// Constructs an enumeration property that will be rendered as a drop-down
HCustomProperty prop3 = HCustomProperty.new_Enumeration("mySetting3",
                 "Some enumerated value setting", false,
                 new String[] { "A", "B", "C" }, new String[]
                 { "Option A", "Option B", "Option C" }, null, null, null);
props.add(prop3);

return props;     }
```



カスタム・コンポーネントまたはウィジェットのユーザーによって指定された値が、コンポーネントの `recognize()` メソッドに渡された *componentSettings* プロパティー・オブジェクト、またはウィジェットのコンストラクターに渡された *widgetSettings* プロパティー・オブジェクト内で使用可能になります。`getCustomProperties()` メソッドは、設定用のデフォルト値を収集するために実行時に呼び出される可能性があります。

これらのメソッドの引数および使用法については、「HATS API References (Javadoc)」で HCustomProperty クラスの説明を参照してください。 2 ページの『API 資料の使用 (Javadoc)』を参照してください。

第 9 章 HATS 双方向 API の使用

注: グローバル変数のオーバーライド、ライト・ペン (アテンション) ホスト・コンポーネント、およびライト・ペン (選択) ホスト・コンポーネントでは、双方向言語はサポートされません。

双方向 (アラビア語またはヘブライ語) のコード・ページを使用する HATS アプリケーションを作成し、ビジネス・ロジックの追加または独自のカスタム・コンポーネントやウィジェットの作成を実行する場合は、双方向 API を使用して、リッチ・クライアント変換内のホスト・コンポーネントの認識とウィジェットの表示を処理することができます。この章では、この API について説明します。この章の資料を利用する前に、「*HATS* ユーザーと管理者のガイド」で説明されている双方向の概念をよく理解しておく必要があります。

注: VisualText コンポーネントについての資料が、トピック『**SWT Bidi Extensions**』の下の Rational SDP ヘルプ内にあります。

データ変換 API

HostScreen クラスには、テキストをビジュアルから論理、もしくはその反対に変換するための API が 2 つ含まれています。データ抽出を処理するためにカスタム・ウィジェットおよびコンポーネントを作成する際に、これらの API を使用できます。

ConvertVisualToLogical

```
public java.lang.String ConvertVisualToLogical(java.lang.String
inputBuffer, boolean isleft-to-rightVisual,
boolean isleft-to-rightImplicit)
```

規定のストリングをビジュアルから暗黙的なフォーマットに変換し、ストリングの暗黙的なフォーマットを戻します。

inputBuffer
入力ストリングは、ビジュアルのフォーマットです。

isleft-to-rightVisual
true の場合、inputBuffer はビジュアルの左から右の形式です。

isleft-to-rightImplicit
true の場合、出力バッファーは暗黙的な左から右の形式です。

ConvertLogicalToVisual

```
public java.lang.String ConvertLogicalToVisual(java.lang.String
inputBuffer, boolean isleft-to-rightImplicit,
boolean isleft-to-rightVisual)
```

規定のストリングを暗黙的なフォーマットからビジュアルに変換し、ストリングのビジュアル・フォーマットを戻します。

inputBuffer
入力ストリングは、暗黙的なフォーマットです。

isleft-to-rightImplicit
true の場合、inputBuffer は暗黙的な左から右の形式です。

isleft-to-rightVisual
true の場合、出力バッファーはビジュアルの左から右の形式です。

グローバル変数 API

グローバル変数の値を取得するための getter メソッドが 2 つあります。これらのメソッドを使用して、暗黙的またはビジュアル・フォーマットでグローバル変数値を取得できます。これらの 2 つのメソッドは、com.ibm.hats.common.BaseInfo クラスにあります。

getGlobalVariable

```
public IGlobalVariable getGlobalVariable(String name, boolean
createIfNotExist,boolean bidiImplicit)
```

名前付きのグローバル変数を取得します。まだ存在しない場合は、オプションで作成できます。

createIfNotExist
存在しないグローバル変数を作成するかどうかを指定します。

bidiImplicit
true の場合は暗黙的なフォーマットで、false の場合はビジュアル・フォーマットでグローバル変数値を取得するかどうかを指定します。

getSharedGlobalVariable

```
public IGlobalVariable getSharedGlobalVariable(String name, boolean
createIfNotExist,boolean bidiImplicit)
```

名前付きの共用グローバル変数を取得します。まだ存在しない場合は、オプションで作成できます。

createIfNotExist
存在しないグローバル変数を作成するかどうかを指定します。

bidiImplicit
true の場合は暗黙的なフォーマットで、false の場合はビジュアル・フォーマットでグローバル変数値を取得するかどうかを指定します。

BIDI OrderBean

双方向データを正しく表示するため、BIDI OrderBean のメソッドを使用できます。これには、次のパラメーターが含まれています。

BidiString
ストリング。双方向テキストが含まれています。

FromTextVisual
ブール値。ソース双方向テキストがビジュアルかどうかを示します。デフォルトは `true` です。

FromOriLTR
ブール値。ソース双方向テキストの方向が LTR かどうかを示します。デフォルトは `true` です。

ToTextVisual
ブール値。ターゲット双方向テキストがビジュアルかどうかを示します。デフォルトは `true` です。

ToOriLTR
ブール値。ターゲット双方向テキストの方向が LTR かどうかを示します。デフォルトは `true` です。

NeedShape
ブール値。双方向テキストがアラビア語テキストであるかどうかと、形状決定が必要かどうかを示します。デフォルトは `false` です。

CharSet
ストリング。JSP 用の文字のエンコードを定義します。

NumShape
ストリング。数表示形状メソッドを定義します。デフォルトは `Nominal` です。

SymSwap
ブール値。対称スワッピングがオンかどうかを示します。デフォルトは `false` です。

BIDI OrderBean メソッド

setBidiString
```
public void setBidiString (String BdString)
```
双方向テキストが所定のストリングに再配列されるように設定します。必要なパラメーターは **BdString** のみで、再配列が必要な双方向ストリングです。

getBidiString
```
public String getBidiString ()
```
双方向テキストを取得します。再配列する必要がある双方向ストリングを戻します。

setFromTextVisual
```
public void setFromTextVisual (boolean on)
```
ソース双方向テキスト・タイプをビジュアルとして設定します。必要なパラメーターは **on** のみです。`true` の場合は、ビジュアルとしてこのソース双方向テキストを定義します。`false` の場合は、暗黙としてこのソース双方向テキストを定義します。

setFromOriLTR
```
public void setfromOriLTR (boolean on)
```

ソース双方向テキストの方向を LTR として設定します。必要なパラメーターは **on** のみです。true の場合は、LTR としてこのソース双方向テキストを定義します。false の場合は、RTL としてこのソース双方向テキストを定義します。

setToTextVisual

```
public void setToTextVisual (boolean on)
```

ターゲット双方向テキスト・タイプをビジュアルとして設定します。必要なパラメーターは **on** のみです。true の場合は、ビジュアルとしてこのターゲット双方向テキストを定義します。false の場合は、暗黙としてこのターゲット双方向テキストを定義します。

setToOriLTR

```
public void setToOriLTR (boolean on)
```

ターゲット双方向テキストの方向を LTR として設定します。必要なパラメーターは **on** のみです。true の場合は、LTR としてこのターゲット双方向テキストを定義します。false の場合は、RTL としてこのターゲット双方向テキストを定義します。

setEncoding

```
public void setEncoding (String CharSet)
```

文字セットのエンコードを設定します。必要なパラメーターは **CharSet** のみで、文字エンコード名を示します。

setNeedShape

```
public void setNeedShape (boolean on)
```

形状決定を実行する必要があることを設定します。必要なパラメーターは **on** のみです。true の場合は、双方向テキストで形状決定を実行する必要があることを示します。

Order `public void Order ()`

双方向テキストの配列を実行します。パラメーターはありません。

CompressLamAlef

```
public String CompressLamAlef(String input,boolean direction)
```

Lam 文字とその後に続く Alef 文字が LamAlef 1 文字に置換されたストリングを戻します。パラメーターには以下のものがあります。

- Direction: true の場合は、入力テキストがビジュアル・フォーマットであることを示します。false の場合は、入力テキストが暗黙のフォーマットであることを示します。
- Input: 圧縮される LamAlef 文字を含む入力ストリング。

ExpandLamAlef

```
public String ExpandLamAlef(String input,boolean direction)
```

Lam Alef 文字が Lam とその後に続く 1 つの Alef 文字に置換されたストリングを戻します。パラメーターには以下のものがあります。

- Direction: true の場合は、入力テキストがビジュアル・フォーマットであることを示します。false の場合は、入力テキストが暗黙のフォーマットであることを示します。

- Input: 展開される LamAlef 文字を含む入力ストリング。

setNumerals

```
public void setNumerals(String NumShape)
```

出力バッファーの数表示形状を設定します。必要なパラメーターは以下のとおりです。

- NumShape: 次の 3 つの値のいずれか 1 つを持つストリングです。
  - NOMINAL: 数表示はラテン語の形式です。
  - CONTEXTUAL: 数表示は数に従います。
  - NATIONAL: 数表示は各国の形式です。
- Input: 展開される LamAlef 文字を含む入力ストリング。

setSymSwap

```
public void setSymSwap (boolean on)
```

ビジュアル RTL の方向で対称スワッピング・オプションを設定します。必要なパラメーターは **on** のみです。true の場合は、対称スワッピングを使用した RTL 画面での文字のスワッピングが可能になります。false の場合は (デフォルト)、RTL 画面での文字のスワッピングに対称スワッピングは使用できません。

ShapeArabicData

```
public String ShapeArabicData(String strInBuffer,boolean
isLTRVisual, boolean EnableNumSwap)
```

アラビア語データが形状決定されたストリングを戻します。パラメーターには以下のものがあります。

- strInBuffer: 形状決定が必要な双方向ストリング。
- isLTRVisual: 展開される LamAlef 文字を含む入力ストリング。true の場合は、双方向ストリングが左から右へビジュアルになります。false の場合は、双方向ストリングが右から左へビジュアルになります。
- EnableNumSwap: true の場合は、数値スワッピングが使用可能になります。false の場合は、数値スワッピングが使用不可になります。

DeshapeArabicData

```
public String DeshapeArabicData (String strInBuffer,boolean
isLTRVisual,boolean EnableNumSwap)
```

アラビア語データが形状決定解除されたストリングを戻します。パラメーターには以下のものがあります。

- strInBuffer: 形状決定解除が必要な双方向ストリング。
- isLTRVisual: true の場合は、双方向ストリングが左から右へビジュアルになります。false の場合は、双方向ストリングが右から左へビジュアルになります。
- EnableNumSwap: true の場合は、数値スワッピングが使用可能になります。false の場合は、数値スワッピングが使用不可になります。

ConvertLogicalToVisual

```
public java.lang.String ConvertLogicalToVisual(java.lang.String
inputBuffer, boolean isLTRimplicit,
boolean isLTRVisual)
```

規定のストリングを暗黙的なフォーマットからビジュアルに変換し、ストリングのビジュアル・フォーマットを戻します。パラメーターには以下のものがあります。

- InputBuffer: 入力ストリングは、暗黙的なフォーマットです。
- isLTRimplicit: `true` の場合、inputBuffer は暗黙的な左から右の形式です。
- isLTRVisual: `true` の場合、出力バッファーはビジュアルの左から右の形式です。

ConvertVisualToLogical

```
public java.lang.String ConvertVisualToLogical(java.lang.String
inputBuffer, boolean isLTRVisual,
boolean isLTRimplicit)
```

規定のストリングをビジュアルから暗黙的なフォーマットに変換し、ストリングの暗黙的なフォーマットを戻します。パラメーターには以下のものがあります。

- InputBuffer: 入力ストリングは、ビジュアルのフォーマットです。
- isLTRimplicit: `true` の場合、出力バッファーは暗黙的な左から右の形式です。
- isLTRVisual: `true` の場合、inputBuffer はビジュアルの左から右の形式です。

付録 A. HATS Toolkit ファイル

HATS Toolkit を使用してプロジェクトをビルドするときに、プロジェクトの各コンポーネント用のファイルが作成されます。この付録では、システム内での各ファイルの位置、ファイルのソースを表示および編集する方法、および各ファイルを形成するタグについて説明します。

注: これらのソース・ファイルを編集する場合は、HATS Toolkit エディターを使用します。

HATS Toolkit で作成するすべてのファイルは、ご使用のシステム上の、Rational SDP プログラム (Rational Application Developer など) によって管理される 1 つ以上のワークスペースに格納されます。ワークスペース・ディレクトリーは選択可能であり、複数のワークスペース・ディレクトリーを持つことができます。ワークスペースの選択の詳細については、Rational SDP で提供されている情報を参照してください。

この付録に示すファイルの場所はすべて、ご使用のプロジェクトと同じ名前のディレクトリー (ご使用のワークスペース内に作成される) からの相対パスを表しています。

アプリケーション・ファイル (.hap)

アプリケーション・ファイルには、プロジェクトを作成するときに選択した設定を定義する XML タグが含まれています。

アプリケーション・ファイル (.hap) は、*project_name*/profiles ディレクトリーに格納されます。ここで、*project_name* は、プロジェクトを作成するときに付けた名前です。HATS EJB プロジェクトのアプリケーション・ファイル (.hap) は、*project_name*/ejbModule ディレクトリーに格納されます。HATS プロジェクトのアプリケーション・ファイルのソースの表示や編集を行うには、「**HATS Projects** 表示」でご使用のプロジェクトを展開し、「プロジェクト設定」をダブルクリックしてプロジェクト・エディターを開きます。ソースを表示するには、「ソース」タブをクリックします。

アプリケーション・ファイルは、プロジェクト・エディターのタブをどれでも使用して変更できます。いずれかのタブで変更を行うと、HATS Toolkit はその他のタブ上で該当情報を更新します。

<application> タグ

<application> タグは、プロジェクトを囲むタグです。

<application> タグの属性には以下のものがあります。

active この属性は、HATS では使用しません。HATS 限定版との互換性のために含まれています。

configured
この属性は、HATS では使用しません。HATS 限定版との互換性のために含まれています。

description
プロジェクトを作成するときに入力する説明を指定します。

template
プロジェクトの作成時にプロジェクト用として選択したテンプレートの名前を指定します。デフォルト・テンプレートは predefined.rcp.templates.Modern です。

<connections> タグ

<connections> タグは、このプロジェクトの接続を定義するすべての connection タグのコンテナーです。

<connections> タグの属性には以下のものがあります。

default
デフォルト接続の名前を指定します。デフォルトでは、デフォルト接続の名前は新規 HATS プロジェクト・ウィザードで指定した接続値を使用して作成される main の名前に設定されます。

<connection> タグ

<connection> タグは、プロジェクトに対して定義されている接続を識別し、接続が定義されている接続ファイル (.hco) を指し示します。

<connection> タグの属性には以下のものがあります。

name 接続の作成時に入力した名前を指定します。

<eventPriority> タグ

<eventPriority> タグは、プロジェクトについて定義した画面イベントを囲むタグです。<eventPriority> タグ内のイベント・タグの順序は、新しいホスト画面が検出されたときにチェックされる画面イベントの順序を表します。このタグには属性はありません。

<event> タグ

<event> タグは、プロジェクトに対して定義した画面イベントを指定します。

<event> タグの属性には以下のものがあります。

enabled
新しいホスト画面が検出されたときに、画面イベントの画面認識基準をチェックするかどうかを指定します。有効な値は `true` および `false` です。デフォルト値は `true` です。

name 画面イベントを定義するときに付けた名前を指定します。画面イベント・ファイルをフォルダー (またはグループ) の下に格納する場合は、そのフォルダーの名前がファイル名の前に付加されます。

type これが画面組み合わせイベントであることを指定します。使用可能な属性は **screenCombination** です。

<classSettings> タグ

<classSettings> タグは、プロジェクトに含める Java クラスを囲むタグです。このタグには属性はありません。

<class> タグ

<class> タグは、<setting> タグで囲まれた属性を持つクラスを指定します。

<class> タグの属性には以下のものがあります。

name 次のいずれかの Java クラスを指定します。

- com.ibm.hats.common.AppletSettings
- com.ibm.hats.common.ApplicationKeypadTag
- com.ibm.hats.common.ClientLocale
- com.ibm.hats.common.DBCSSettings
- com.ibm.hats.common.DefaultConnectionOverrides
- com.ibm.hats.common.DefaultGVOverrides
- com.ibm.hats.common.HostKeypadTag
- com.ibm.hats.common.KeyboardSupport
- com.ibm.hats.common.OIA
- com.ibm.hats.common.RuntimeSettings
- com.ibm.hats.rcp.transform.widgets.*name*
- com.ibm.hats.rcp.ui.views.ToolBarSettings
- com.ibm.hats.transform
- com.ibm.hats.transform.components.*name*
- com.ibm.hats.transform.DefaultRendering

<setting> タグ

<setting> タグは、<setting> タグを囲んでいるクラスに関連する設定を指定します。<setting> タグには、クラスごとに **name** と **value** のペアが含まれます。次のセクションで、各クラスの **name** と **value** のペアについて説明します。

com.ibm.hats.common.AppletSettings

com.ibm.hats.common.AppletSettings クラスの場合、name は リッチ・クライアント アプリケーションの非同期更新機能のカスタマイズ可能な設定を指定します。

enable

`true` の場合は、リッチ・クライアント アプリケーションの非同期更新機能が使用可能になります。デフォルトは **true** です。

com.ibm.hats.common.ApplicationKeypadTag

com.ibm.hats.common.ApplicationKeypadTag クラスの場合、name は以下のカスタマイズ可能な設定を指定します。

show `true` の場合、アプリケーションの中でキーパッドを表示します。

showDefault
`true` の場合、表示をデフォルトの変換に変更するためのキーをアプリケーション・キーパッドに表示します。

showDisconnect
`true` の場合、ホストから切断するためのキーをキーパッドに表示します。

showKeyboardToggle
`true` の場合、アプリケーション・キーパッドの中に、ホスト・キーボードの表示を切り替えるためのキーを表示します。

showRefresh
`true` の場合、ブラウザー・ウィンドウの内容をオリジナルの変換を使用して最新表示し、入力フィールドを元の値に復元するキーを、キーパッドに表示します。

showReverse
`true` の場合、双方向サポート用のキーをアプリケーション・キーパッドに表示します。

com.ibm.hats.common.ClientLocale

com.ibm.hats.common.ClientLocale クラスの場合、name は常に `locale` です。locale 設定の値により、ボタンのキャプションおよびメッセージを表示するのに使用される言語が指定されます。値は次のいずれかです。デフォルトは **accept-language** です。

ロケールの国別コードを示す文字

ar アラビア語

cs チェコ語

de ドイツ語

en 英語

es スペイン語

fr フランス語

hu ハンガリー語

it イタリア語

ja 日本語

ko 韓国語

pl ポーランド語

pt_BR
ブラジル・ポルトガル語

ru ロシア語

tr トルコ語

zh 中国語 (簡体字)

zh_TW
中国語 (繁体字)

accept-language
言語はご使用のブラウザーの `Accept-Language` HTTP ヘッダーから取得されます。

com.ibm.hats.common.DBCSSettings

com.ibm.hats.common.DBCSSettings クラスの場合、autoConvertSBCStoDBCS、setATOKDefaultModetoRoman、および showUnprotectedSISOSpace という 3 つの設定値があります。

- **autoConvertSDBCtoDBCS** 属性の有効な値は以下のとおりです。

  true 3270 および 3270E G タイプまたは 5250 G タイプおよび J タイプ・フィールドの場合、1 バイト文字を 2 バイト文字に自動変換します。

  false 3270 および 3270E G タイプまたは 5250 G タイプおよび J タイプ・フィールドの場合、1 バイト文字を 2 バイト文字に自動変換しません。

  デフォルトは **false** です。詳しくは、「*HATS* ユーザーと管理者のガイド」のセクション『プロジェクト設定エディター』を参照してください。

- **setATOKDefaultModetoRoman** 属性の有効な値は以下のとおりです。

  true ATOK のデフォルトの入力モードを Roman に設定します。

  false ATOK のデフォルトの入力モードを Hanji に設定します。

  デフォルトは **false** です。

- **showUnprotectedSISOSpace** 属性の有効な値は以下のとおりです。

  true 無保護のシフトイン文字またはシフトアウト文字をスペースで表示します。

  false 無保護のシフトイン文字またはシフトアウト文字を表示するためにスペースを使用しません。

  デフォルトは **true** です。詳しくは、「*HATS* ユーザーと管理者のガイド」のセクション『プロジェクト設定エディター』を参照してください。

com.ibm.hats.common.DefaultConnectionOverrides

com.ibm.hats.common.DefaultConnectionOverrides クラスの場合、`allowAll` という name 属性の <setting> タグが少なくとも 1 つは常に存在します。この <setting> タグは、接続パラメーターのオーバーライドに関して選択されたデフォルトのセキュリティー・ポリシーを示します。接続のオーバーライドに関して選択されたセキュリティー・ポリシーに対するすべての例外は、追加の <setting> タグを使用して記録され、その name 属性は、例外接続パラメーターの名前に設定されます。

name 属性の有効な値は以下のとおりです。

true エンド・ユーザーは、指定された接続パラメーターをオーバーライドできます。指定された接続パラメーターが「`allowAll`」である場合、これは、指定されていないすべての接続パラメーターをクライアント要求でオーバーライドできるという意味になります。

false エンド・ユーザーは、指定された接続パラメーターをオーバーライドできません。指定された接続パラメーターが「allowAll」である場合、これは、指定されていないどの接続パラメーターもオーバーライドできないという意味になります。

「allowAll」のデフォルト設定は **false** です。

com.ibm.hats.common.DefaultGVOverrides

com.ibm.hats.common.DefaultGVOverrides クラスの場合、allowAll という name 属性の <setting> タグが少なくとも 1 つは常に存在します。この <setting> タグは、グローバル変数のオーバーライドに関して選択されたデフォルトのセキュリティー・ポリシーを示します。選択されたセキュリティー・ポリシーに対するすべての例外が追加の <setting> タグを使用して記録され、その name 属性は、hatsgv_*variableName* (通常のグローバル変数例外の場合) または hatssharedgv_*variableName* (共用グローバル変数例外の場合) に設定されます。

name 属性の有効な値は以下のとおりです。

true エンド・ユーザーは、指定された接続パラメーターをオーバーライドできます。指定された接続パラメーターが「allowAll」である場合、これは、指定されていないすべての接続パラメーターをクライアント要求でオーバーライドできるという意味になります。

false エンド・ユーザーは、指定された接続パラメーターをオーバーライドできません。指定された接続パラメーターが「allowAll」である場合、これは、指定されていないどの接続パラメーターもオーバーライドできないという意味になります。

「allowAll」のデフォルト設定は **false** です。

com.ibm.hats.common.HostKeypadTag

com.ibm.hats.common.HostKeypadTag クラスの場合、name は以下のカスタマイズ可能な設定を指定します。

show true の場合、アプリケーションの中でホスト・キーパッドを表示します。

showAltView
true の場合、AltView キーをホスト・キーパッドに表示します。

showAttention
true の場合、ATTN キーをホスト・キーパッドに表示します。

showClear
true の場合、CLEAR キーをホスト・キーパッドに表示します。

showEnter
true の場合、Enter キーをホスト・キーパッドに表示します。

showF1 – showF24
`true` の場合、対応する番号の付いたファンクション・キーをホスト・キーパッドに表示します。

showFieldExit
`true` の場合、Field Exit キーをホスト・キーパッドに表示します。

showFieldMinus
`true` の場合、Field Minus キーをホスト・キーパッドに表示します。

showFieldPlus
`true` の場合、Field Plus キーをホスト・キーパッドに表示します。

showHelp
`true` の場合、Help キーをホスト・キーパッドに表示します。

showPA1
`true` の場合、PA1 キーをホスト・キーパッドに表示します。

showPA2
`true` の場合、PA2 キーをホスト・キーパッドに表示します。

showPA3
`true` の場合、PA3 キーをホスト・キーパッドに表示します。

showPageDown
`true` の場合、Page Down キーをホスト・キーパッドに表示します。

showPageUp
`true` の場合、Page Up キーをホスト・キーパッドに表示します。

showPrint
`true` の場合、出力を印刷するための PRINT キーをホスト・キーパッドに表示します。

showReset
`true` の場合、RESET キーをホスト・キーパッドに表示します。

showSystemRequest
`true` の場合、SYSREQ キーをホスト・キーパッドに表示します。

style value=`true` として定義したキーをホスト・キーパッドにどのように表示するかを指定します。有効な値は `buttons` または `links` です。デフォルトは **buttons** です。

com.ibm.hats.common.KeyboardSupport

com.ibm.hats.common.KeyboardSupport クラスの場合、name は以下のカスタマイズ可能な設定を指定します。

enable
プロジェクトでキーボード・サポートが使用可能かどうかを指定します。キーボード・サポートが使用可能の場合、エンド・ユーザーは物理的なキーボードのキーを使用してホストと対話できます。エンド・ユーザーは、F1、SYSREQ、RESET、ATTN キーなどホストの補助キーに特定の物理的なキーをマップして押すことができます。エンド・ユーザーは、マップされた物理的なキーボードのキーを使用してブラウザーと対話する場合、キーボード・サポートを切り替えて使用不可にできます。

注: HATS テーマをデフォルトのエミュレーター・スタイルから最新のアプリケーション・スタイルに変更できるウィザードを表示するには、これを true に設定する必要があります。

initialState
true の場合、ホスト・キーボードの初期状態はオン (ユーザーは物理的なキーボードを使用してアプリケーションと対話できる) になります。

supportAllKeys
true の場合は、表示されているボタンやリンクに関係なく、すべてのマップされたキーがサポートされます。false の場合は、以下のいずれかになります。

- 現在のページに認識されたホスト機能がボタンやリンクとしてまったく表示されていない場合、マップされたホスト機能のすべてをサポート。
- 認識されたホスト機能がボタンやリンクとして表示されている場合、それらのホスト機能のみサポート。

com.ibm.hats.common.OIA

com.ibm.hats.common.OIA クラスの場合、name は以下のカスタマイズ可能な設定を指定します。

active true の場合、オペレーター情報域 (OIA) はプロジェクトで可視の状態にあります。デフォルトは **true** です。

注: HATS テーマをデフォルトのエミュレーター・スタイルから最新のアプリケーション・スタイルに変更できるウィザードを表示するには、これを true に設定する必要があります。

appletActive
true の場合、非同期更新サポートが使用可能であれば、標識が OIA に表示されます。デフォルトは **false** です。

autoAdvanceIndicator
true の場合、ブラウザーでサポートしていれば、自動拡張が使用可能かどうか、OIA に表示されます。デフォルトは **false** です。

bidiControls
true の場合、ブラウザーでサポートしていれば、現在の双方向制御を OIA に表示し編集状況を示すことができます。デフォルトは **true** です。

cursorPosition
true の場合、1391 など、ホストの絶対カーソル位置が OIA に表示されます。デフォルトは **false** です。

cursorRowColumn
true の場合、18/031 など、ホストのカーソルの行とカラムが OIA に表示されます。デフォルトは **true** です。

fieldData
true の場合、numeric only や field exit required などのフィールド拡張データが OIA に表示されます。デフォルトは false です。

inputInhibited
`true` の場合、キーボードがロックされ、キーボードから入力不可となっているかどうか、OIA に表示されます。デフォルトは **true** です。

insertMode
`true` の場合、ブラウザーがサポートしていれば、上書きモードが使用可能になっているかどうか、OIA に表示されます。デフォルトは **true** です。

msgWaiting
true の場合、ホスト・システムにセッションについての 1 つ以上のメッセージがあれば、標識が表示されます。設定は 5250 ホスト・システムにのみ適用されます。

sslCheck
`true` の場合、Host On-Demand 接続が SSL で保護されているかどうか、OIA に表示されます。デフォルトは **true** です。

systemWait
`true` の場合、データの戻りを待つ間にシステムをロックするかどうか、OIA に表示されます。デフォルトは **true** です。

typeAheadField
`true` の場合、OIA に先行入力フィールドを表示します。このフィールドにはユーザーの入力どおりに先行入力データが表示されますが、このフィールドを直接編集することはできません。この設定は、先行入力サポートが有効な場合にのみ有効です。 「*HATS* ユーザーと管理者のガイド」の『先行入力サポートを使用可能にする』を参照してください。デフォルトは **false** です。

com.ibm.hats.common.RuntimeSettings

com.ibm.hats.common.RuntimeSettings クラスの場合、name は以下のカスタマイズ可能な設定を指定します。

autoEraseFields
変更された入力フィールドに変更されたデータが入力される前に、そのフィールドに `[erasefld]` を適用するかどうかを指定します。デフォルト値は **true** です。この値が `false` に設定されている場合は、スペース文字を使用して、ホストが以前にそのフィールドに入力したデータを置き換えることができます。

注**:**

1. 複数の入力フィールドとしてレンダリングされるホスト・フィールドは、自動的にはクリアされません。例えば、1 つの行から別の行にまたがる長いホスト・フィールドは複数の入力フィールドとしてレンダリングされ、更新前に自動的にはクリアされません。
2. この設定は、プロジェクト・レベルでのみ指定可能です。単一の変換に対して指定することはできません。

enableArrowKeyNavigation
`true` に設定されている場合、レンダリングされたホスト画面内のフィールド間をキーボードの矢印キーでナビゲートできます。デフォルトは **false** です。

enableAutoAdvance

カーソルが入力フィールドの終わりにきたとき (つまり、入力フィールドが完全に入力されたとき) に、カーソルが次の入力フィールドに移動するかどうかを指定します。 `true` の場合、カーソルが入力フィールドの終わりにきたときに、カーソルが次の入力フィールドに移動します。 `false` の場合、ユーザーが明示的に移動させない限り、カーソルは次の入力フィールドに移動しません。デフォルトは **false** です。

enableAutoTabOn

カーソルが入力フィールドの終わりにきたとき (つまり、入力フィールドが完全に入力されたとき) に、タブ・キーによってカーソルを次の入力フィールドに移動させるかどうかを指定します。 `true` に設定されている場合、カーソルの位置が現行フィールドの終わりにきたときに、ブラウザーでの表示フィールドの順序に基づいて、タブ・キーによって現行フィールド内のカーソルを次のフィールドに移動させます。 `false` に設定されている場合、ユーザーが明示的に移動しない限り、タブ・キーがカーソルを次の入力フィールドに移動させることはありません。デフォルトは **false** です。

enableOverwriteMode

`true` の場合、入力フィールドに入力されたテキストにより、カーソル位置にあるテキストが上書きされます (1 文字が 1 文字に置き換わる)。`false` の場合、入力フィールドに入力されたテキストがカーソル位置に挿入され、既存のテキストが前に押し出されます。ユーザーは、Insert キーを使用して、この初期設定を切り替えることができます。デフォルトは **true** です。

enableTypeAhead

これが `true` に設定されている場合、ユーザーはホストから送信される次の画面が HATS で受信されて処理される前に、その画面の入力フィールドを対象とするデータを入力し始めることができます。次の画面が受信されると、HATS は、入力をホストに送信するキーを含め、既に入力されていたデータ (先行入力データ) を送信します。 「*HATS* ユーザーと管理者のガイド」の『先行入力サポートを使用可能にする』を参照してください。デフォルトは **false** です。

includeLabelsInTabOrder

`true` の場合、保護された読み取り専用ラベルがデフォルトのタブ移動順序に組み込まれます。デフォルトでは、タブ移動順序にはパネル上の入力フィールドのみが組み込まれます。この設定は、読み取り専用ラベルも組み込む必要があることを意味します。

selectAllOnFocus

`true` の場合、フィールドがフォーカスを受け取ったときに、フィールド内のすべてのテキストが選択されます。これは、Web アプリケーションの通常の動作です。`false` の場合、フィールドがフォーカスを受け取ったときに、テキストは選択されません。これは、端末エミュレーターの通常の動作です。

注**:**

1. Web アプリケーションの場合:
   - デフォルトは **true** です。
   - この設定は、**enableOverwriteMode** 設定の動作には影響しません。

- この設定は、アプリケーションのブラウザーとして Internet Explorer が使用される場合に限り有効です。

2. リッチ・クライアント アプリケーションの場合:
   - デフォルトは **false** です。
   - 選択すると、文字がフィールドへのユーザー入力として上書きされるという点で、この設定は **enableOverwriteMode** 設定と同様に機能します。
   - キーボードを使用してフィールドに Tab で移動した場合に限り、テキストが選択されます。フィールド内でマウスをクリックしても、テキストは選択されません。

suppressUnchangedData

`true` の場合、内容がフォームのレンダリング時と変わらないすべてのフィールドを使用不可にします。 `false` の場合、ブラウザーから受信したフィールドの内容と現在の表示スペースの内容が同一であっても、そのフィールドの内容をホストに送信します。デフォルトは **false** です。

com.ibm.hats.rcp.transform.widgets.*name*

ウィジェットの設定の詳細については、「*HATS* ユーザーと管理者のガイド」を参照してください。

com.ibm.hats.rcp.ui.views.ToolBarSettings

com.ibm.hats.rcp.ui.views.ToolBarSettings クラスの場合、name は以下のカスタマイズ可能な設定を指定します。

displayAs

メイン変換ビュー・ツールバーの項目の表示方法を指定します。有効な値は、`TEXT`、`IMAGE`、および `BOTH` です。デフォルトは **TEXT** です。

show メイン変換ビュー・ツールバーを表示するかどうかを指定します。有効な値は `true` および `false` です。デフォルトは **true** です。

com.ibm.hats.transform

com.ibm.hats.transform クラスの場合、name は以下のカスタマイズ可能な設定を指定します。

alternate

alternateRenderingSet が指定されている場合、値は `DEFAULT` です。それ以外の場合は、未指定です。

alternateRenderingSet

HATS コンポーネント・タグの変換中にレンダリングするものが何もないことが検出された場合にデフォルト・レンダリングに使用するレンダリング・セットの名前を指定します。

com.ibm.hats.transform.components.*name*

コンポーネントの設定の詳細については、「*HATS* ユーザーと管理者のガイド」を参照してください。

com.ibm.hats.transform.DefaultRendering

com.ibm.hats.transform.DefaultRendering クラスの場合、name は常に applicationDefaultRenderingSetName です。 value は、プロジェクトのデフォルトのレンダリング・セットとして定義するレンダリング・セットの名前を指定します。 value 設定で指定したレンダリング・セット名は、<defaultRendering> タグに指定した default 属性の値と一致しなければなりません。

<textReplacement> タグ

<textReplacement> タグは、プロジェクトの中で定義するテキスト置換値を囲むタグです。このタグには属性はありません。

<replace> タグ

<replace> タグは、プロジェクト内のテキスト置換値を指定します。

注: 双方向コード・ページを使用している場合は、「*HATS* ユーザーと管理者のガイド」を参照してください。

<replace> タグの属性には以下のものがあります。

caseSensitive
テキスト置換値の大文字小文字の区別が一致した場合のみテキスト置換を行うかどうかを指定します。有効な値は true および false です。デフォルトは false です。

from 置換するテキストを指定します。**from** 属性に指定するテキストは引用符で囲む必要があります。

to テキストをテキストまたは HTML コード (Web のみ) で置き換える場合に、**from** 属性で指定した値の代わりに挿入する置き換えストリングを指定します。 **to** 属性の置き換えストリングは引用符で囲む必要があります。テキストをボタンまたはリンクと置換する場合、引用符の中にボタンやリンク用のコードを追加しなければなりません。

regularExpression
テキスト置換アルゴリズムの一部として、Java 正規表現のサポートを使用するかどうかを指定します。正規表現はストリングのセットを記述する文字のパターンです。正規表現を使用してパターンのオカレンスを検索、変更できます。有効な値は true および false です。デフォルトは false です。

toImage
テキストをイメージと置換する場合、**from** 属性で指定した値の代わりに挿入するイメージのパスと名前を指定します。**toImage** 属性で指定するイメージのパスと名前は引用符で囲む必要があります。

matchLTR
双方向コード・ページを使用する場合、**from** 属性で指定した値を LTR 画面で置換するかどうかを指定します。有効な値は true および false です。デフォルトは true です。

matchRTL

双方向コード・ページを使用する場合、**from** 属性で指定した値を RTL 画面で置換するかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `false` です。

matchReverse

双方向コード・ページを使用する場合、表示される画面のセクションが元のページの方向から反転した際、**from** 属性で指定した値を置換するかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `false` です。

注: テキスト置換を使用するときは、注意が必要です。ストリングの文字数が異なるテキスト置換を行うと、画面の表示が変化することがあります。画面上の領域を提示するために使用されるウィジェットによっては、画面上の 1 行に収められているテキストが、縮小または拡張されたり、次の行に継続したりすることがあります。

<defaultRendering> タグ

<defaultRendering> タグは、プロジェクトの中で定義するすべてのレンダリング・セットを囲むタグです。

<defaultRendering> タグの属性は次のとおりです。

default

プロジェクトの中でデフォルトのレンダリングに使用するレンダリング・セットの名前を指定します。default 属性で指定したレンダリング・セット名は、com.ibm.hats.transform.DefaultRendering という名前のクラス設定の value 属性に指定した値と一致していなければなりません。

<renderingSet> タグ

<renderingSet> タグは、レンダリング・セットで定義したレンダリング項目を囲むタグです。

<renderingSet> タグの属性には以下のものがあります。

name レンダリング・セットの作成時に指定した名前。

description

レンダリング・セットの作成時に指定した説明。

layout

変換された画面のフィールドおよびテキスト内にある不要なブランクを除去するコンパクト・レンダリングを使用するかどうかを示します。この属性は、コンパクト・レンダリングをデフォルトにする場合にのみ使用するようにしてください。 COMPACT が、layout に有効な唯一の値です。デフォルトではレンダリング・セットにこの属性は指定されておらず、コンパクト・レンダリングは使用されません。

separated

各フィールドを区別し、HTML およびブランク・スペースの量を削減するために、変換された画面でインラインの span タグを使用して出力をレンダリングするかどうかを示します。モバイル装置用に最適化された Web アプ

リケーションの場合、これがデフォルトです。デフォルトでは、レンダリング・セットにこの属性は指定されていません。

table 出力を表でレンダリングして、元のホスト画面のレイアウトを保持するようにするかどうかを示します。モバイル装置用に最適化されていない Web アプリケーションの場合、これがデフォルトです。

<renderingItem> タグ

<renderingItem> タグは、特定のレンダリング項目を囲むタグです。

<renderingItem> タグの属性には以下のものがあります。

componentIdentifier
コンポーネント情報を変換と調整するために使用するレンダリング項目の名前。デフォルト設定は、画面組み合わせイベントの名前です。

associatedScreen
このレンダリング項目の作成に使用したキャプチャーされた画面の名前。

description
レンダリングの作成時に入力した説明。

enabled
レンダリングが使用可能かどうかを表します。プロジェクト設定の「レンダリング」ページのチェック・ボックスの状態を反映します。

endCol
このレンダリング項目を適用するホスト画面の最後のカラムです。-1 は、ホスト画面の右端のカラムを意味します。

endRow
このレンダリング項目を適用するホスト画面の最後の行です。-1 は、ホスト画面の最後の行を意味します。

startCol
このレンダリング項目を適用するホスト画面の最初のカラムです。

startRow
このレンダリング項目を適用するホスト画面の最初の行です。

type 変換する対象となる内容を持つホスト・コンポーネント。この属性の値は、ホスト・コンポーネントの完全なクラス名です。この属性は必須であり、デフォルト値はありません。

widget
ホスト・コンポーネントは、このタグに指定したウィジェットに変換されます。

それぞれ固有のレンダリング項目には、次のタグも含まれています。

componentSettings
<componentSettings> タグは、このレンダリング項目のコンポーネント用に変更された設定を囲むタグです。このタグには属性はありません。

setting
<setting> タグは、このレンダリング項目のコンポーネント用に変更された設定を囲むタグです。

<setting> タグの属性には以下のものがあります。

name コンポーネントのカスタマイズ可能な設定の名前を指定します。使用可能な設定はコンポーネントによって異なります。

コンポーネントの設定の詳細については、「*HATS* ユーザーと管理者のガイド」を参照してください。

value コンポーネントのカスタマイズ可能な設定の値を指定します。デフォルト値は、設定により異なります。

widgetSettings

<widgetSettings> タグは、このレンダリング項目のウィジェット用に変更された設定を囲むタグです。このタグには属性はありません。

setting

<setting> タグは、このレンダリング項目のウィジェット用に変更された設定を囲むタグです。

<setting> タグの属性には以下のものがあります。

name ウィジェットのカスタマイズ可能な設定の名前を指定します。使用可能な設定は、ウィジェットによって異なります。

ウィジェットの設定の詳細については、「*HATS* ユーザーと管理者のガイド」を参照してください。

value ウィジェットのカスタマイズ可能な設定の値を指定します。デフォルト値は、設定により異なります。

textReplacements

<textReplacements> タグは、このレンダリング項目に指定されたテキスト置換を囲むタグです。このタグには属性はありません。

replace

<replace> タグは、このレンダリング項目のテキスト置換値を指定します。

<replace> タグの属性には以下のものがあります。

caseSensitive

テキスト置換値の大文字小文字の区別が一致した場合のみテキスト置換を行うかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `false` です。

from 置換するテキストを指定します。**from** 属性に指定するテキストは引用符で囲む必要があります。

to **from** 属性で指定した値の代わりに挿入する置き換えストリングを指定します。**to** 属性の置き換えストリングは引用符で囲む必要があります。

regularExpression

テキスト置換アルゴリズムの一部として、Java 正規表現のサポートを使用するかどうかを指定します。正規表現はストリングのセットを記述する文字のパターンです。正規表現を使用してパターンのオカレンスを検索、変更できます。有効な値は `true` および `false` です。デフォルトは `false` です。

toImage
テキストをイメージと置換する場合、**from** 属性で指定した値の代わりに挿入するイメージのパスと名前を指定します。**toImage** 属性で指定するイメージのパスと名前は引用符で囲む必要があります。

matchLTR
双方向コード・ページを使用する場合、**from** 属性で指定した値を LTR 画面で置換するかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `true` です。

matchRTL
双方向コード・ページを使用する場合、**from** 属性で指定した値を RTL 画面で置換するかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `false` です。

matchReverse
双方向コード・ページを使用する場合、表示される画面のセクションが元のページの方向から反転した際、**from** 属性で指定した値を置換するかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `false` です。

注: テキスト置換を使用するときは、注意が必要です。ストリングの文字数が異なるテキスト置換を行うと、画面上の HTML 表現が変化することがあります。画面上の領域を提示するために使用されるウィジェットによっては、画面上の 1 行に収められているテキストが、縮小または拡張されたり、次の行に継続したりすることがあります。

<globalRules> タグ

<globalRules> タグは、プロジェクトの中で定義するグローバル規則を囲むタグです。このタグには属性はありません。

<rule> タグ

<rule> タグは、グローバル規則を定義します。

プロジェクト・レベル規則用の <rule> タグの属性は、画面カスタマイズ・レベルのグローバル規則用の属性と同じです。しかし、同じ入力フィールドを使用してプロジェクト・レベルと画面カスタマイズ・レベルのグローバル規則を作成すると、画面カスタマイズ・レベル規則の優先順位の方が高くなります。 <rule> タグの属性には以下のものがあります。

associatedScreen
グローバル規則を定義する際の基準となる、プロジェクトの中の画面キャプチャーの名前。

description
グローバル規則の作成時に入力した説明。

enabled
グローバル規則が使用可能かどうかを表します。プロジェクト設定の「レンダリング」ページのチェック・ボックスの状態を反映します。

endCol
このグローバル規則を適用するべきホスト画面の最後のカラムです。-1 は、ホスト画面の右端のカラムを意味します。

endRow
このグローバル規則を適用するべきホスト画面の最後の行です。-1 は、ホスト画面の最後の行を意味します。

name 「プロジェクト設定」の「レンダリング」ページのグローバル規則のリストに表示される名前。

startCol
このグローバル規則を適用するべきホスト画面の最初のカラムです。

startRow
このグローバル規則を適用するべきホスト画面の最初の行です。

transformationFragment
グローバル規則に関連付けられた変換フラグメント・ファイルの名前。このファイルには、ホスト・コンポーネントの変換方法を指定する情報が含まれています。該当するフィールドがホスト画面に存在する場合、この情報は変換に組み込まれます。

type このグローバル規則のパターン・タイプ・コンポーネントで、「グローバル規則の作成」ウィザードの最初のページで取得します。タイプは次のいずれかです。

com.ibm.hats.transform.components. InputFieldByTextPatternComponent
このパターン・コンポーネントは、フィールド近くのテキストを基にしてホスト画面上の入力フィールドを認識します。

com.ibm.hats.transform.components. AllInputFieldsPatternComponent
このパターン・コンポーネントは、ホスト画面上のすべての入力フィールドを認識します。

com.ibm.hats.transform.components. InputFieldBySizePatternComponent
このパターン・コンポーネントは、入力フィールドのサイズを基にしてホスト画面上の入力フィールドを認識します。

com.ibm.hats.transform.components. InputFieldByPositionPatternComponent
このパターン・コンポーネントは、ホスト画面上のフィールドの位置により、ホスト画面上の入力フィールドを認識します。

それぞれ固有のグローバル規則には、次のタグも含まれています。

componentSettings
<componentSettings> タグは、<rule> タグの型属性で指定したパターン・タイプ・コンポーネント用に定義された設定を囲むタグです。このタグには属性はありません。

setting
<setting> タグは、<rule> タグの型属性で指定したパターン・タイプ・コンポーネント用に定義された設定を囲むタグです。

<setting> タグの属性には以下のものがあります。

name パターン・タイプ・コンポーネントのカスタマイズ可能な設定の名前を指定します。使用可能な設定はコンポーネントによって異なります。

- com.ibm.hats.transform.components.InputFieldByTextPattern Component の場合、name 属性の設定は以下のとおりです。

  caseSensitive
  パターンを認識する前に、テキスト設定で大文字小文字の区別を一致させる必要があるかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `true` です。

  immediatelyNextTo
  変換する入力フィールドを指定します。有効な値は以下のとおりです。

  true 最も近い入力フィールドのみ変換の必要があることを指定します。

  false すべての入力フィールドを変換する必要があることを指定します。

  デフォルトは `false` です。

  location
  テキスト設定で指定した保護フィールドのテキストを、適用するグローバル規則の入力フィールドに対してどの位置に置くかを指定します。有効な値は以下のとおりです。

  ABOVE
  入力フィールドの上に表示するテキストを指定します。

  BELOW
  入力フィールドの下に表示するテキストを指定します。

  LEFT 入力フィールドの左に表示するテキストを指定します。

  RIGHT
  入力フィールドの右に表示するテキストを指定します。

  デフォルトは RIGHT です。

  text ホスト画面の保護フィールドに表示するテキストを指定します。有効な値は、ホスト画面上の保護フィールドの中の任意のテキストです。

- com.ibm.hats.transform.components.AllInputFieldsPattern Component の場合、コンポーネント設定はありません。

- com.ibm.hats.transform.components.InputFieldBySizePattern Component の場合、**name** 属性の設定は **fieldSize** です。有効な値は、ホスト画面上の任意の入力フィールドのサイズです。
- com.ibm.hats.transform.components. InputFieldByPositionPatternComponent の場合、**name** 属性の設定は **enableFieldLength** です。有効な値は `true` および `false` です。

  注**: enableFieldLength** が指定される場合、フィールドが認識されるためには、そのフィールド全体 (**fieldSize** 属性で指定されたもの) が定義された領域の境界内になければなりません。領域の境界は、**startRow**、**endRow**、**startCol**、および **endCol** 属性の値によって定義されます。

接続ファイル (.hco)

HATS プロジェクトで定義するそれぞれの接続は、接続ファイルによって表現されます。接続ファイル (.hco) は、*project_name*/Connections フォルダーに格納されます。ここで、*project_name* は、プロジェクトを作成するときに付けた名前です。新規 HATS プロジェクト・ウィザードで指定した接続値を使用して作成されるデフォルト接続は、main.hco に保管されます。

<hodconnection> タグ

<hodconnection> タグは、接続定義の始まりであり、接続に関するいくつかの特性を指定します。

<hodconnection> タグの属性には以下のものがあります。

certificateFile
プロジェクトの SSL 証明書をインポートした場合は、インポート元のファイルの名前を指定します。

codePage
この接続で使用するコード・ページの数値を指定します。デフォルト値は、プロジェクトの作成時に選択した値です。接続ごとに、別のコード・ページを使用できます。コード・ページ番号については、codePageKey 属性の説明を参照してください。

codePageKey
数字コード・ページに対応する使用目的キーを指定します。デフォルト値は KEY_US です。codePage の有効な値と、それぞれの場所または使用目的キーは以下のとおりです。

表 *18.* コード・ページと使用目的キー

コード・ページ	使用目的キー
037	KEY_BELGIUM KEY_BRAZIL KEY_CANADA KEY_NETHERLANDS KEY_PORTUGAL KEY_US
273	KEY_AUSTRIA KEY_GERMANY
274	KEY_BELGIUM_OLD
275	KEY_BRAZIL_OLD
277	KEY_DENMARK KEY_NORWAY
278	KEY_FINLAND KEY_SWEDEN
280	KEY_ITALY
284	KEY_SPAIN KEY_LATIN_AMERICA
285	KEY_UNITED_KINGDOM
297	KEY_FRANCE
420	KEY_ARABIC
424	KEY_HEBREW
500	KEY_MULTILINGUAL
803	KEY_HEBREW_OLD
838	KEY_THAI
870	KEY_BOSNIA_HERZEGOVINA KEY_CROATIA KEY_CZECH KEY_HUNGARY KEY_POLAND KEY_ROMANIA KEY_SLOVAKIA KEY_SLOVENIA
871	KEY_ICELAND
875	KEY_GREECE
924	KEY_MULTILINGUAL_ISO_EURO
930	KEY_JAPAN_KATAKANA
933	KEY_KOREA_EX
937	KEY_ROC_EX
939	KEY_JAPAN_ENGLISH_EX

表 *18.* コード・ページと使用目的キー *(*続き*)*

コード・ページ	使用目的キー
1025	KEY_BELARUS KEY_BULGARIA KEY_MACEDONIA KEY_RUSSIA KEY_SERBIA_MONTEGRO
1026	KEY_TURKEY
1047	KEY_OPEN_EDITION
1112	KEY_LATVIA KEY_LITHUANIA
1122	KEY_ESTONIA
1123	KEY_UKRAINE
1137	KEY_HINDI
1140	KEY_BELGIUM_EURO KEY_BRAZIL_EURO KEY_CANADA_EURO KEY_NETHERLANDS_EURO KEY_PORTUGAL_EURO KEY_US_EURO
1141	KEY_AUSTRIA_EURO KEY_GERMANY_EURO
1142	KEY_DENMARK_EURO KEY_NORWAY_EURO
1143	KEY_FINLAND_EURO KEY_SWEDEN_EURO
1144	KEY_ITALY_EURO
1145	KEY_LATIN_AMERICA_EURO KEY_SPAIN_EURO
1146	KEY_UNITED_KINGDOM_EURO
1147	KEY_FRANCE_EURO
1148	KEY_MULTILINGUAL_EURO
1149	KEY_ICELAND_EURO
1153	KEY_BOSNIA_HERZEGOVINA_EURO KEY_CROATIA_EURO KEY_CZECH_EURO KEY_HUNGARY_EURO KEY_POLAND_EURO KEY_ROMANIA_EURO KEY_SLOVAKIA_EURO KEY_SLOVENIA_EURO
1154	KEY_BELARUS_EURO KEY_BULGARIA_EURO KEY_MACEDONIA_EURO KEY_RUSSIA_EURO KEY_SERBIA_MONTEGRO_EURO

表 *18.* コード・ページと使用目的キー *(*続き*)*

コード・ページ	使用目的キー
1155	KEY_TURKEY_EURO
1156	KEY_LATVIA_EURO KEY_LITHUANIA_EURO
1157	KEY_ESTONIA_EURO
1158	KEY_UKRAINE_EURO
1160	KEY_THAI_EURO
1166	KEY_KAZAKHSTAN_EURO
1364	KEY_KOREA_EURO
1371	KEY_ROC_EURO
1388	KEY_PRC_EX_GBK
1390	KEY_JAPAN_KATAKANA_EX_EURO
1399	KEY_JAPAN_ENGLISH_EX_EURO

connecttimeout
HATS がホストに接続を試みる時間を指定します。1 から 2147483647 までの秒数を指定します。初期デフォルト値は 120 秒です。

description
作成したときの接続の説明を指定します。

disableFldShp
双方向コード・ページを使用する場合、パスワード・フィールドのアラビア語データを別個の形式または整形された形式でホストに送信するかどうか指定します。有効な値は `true` および `false` です。初期デフォルト値はありません。

disableNumSwapSubmit
双方向コード・ページを使用する場合、アラビア (西洋) 数字の入力を使用不可にするかどうか、つまり、RTL 画面にアラビア (インド) 数字のみを入力できるようにするかどうかを指定します。これは、送信時に、すべての数字がアラビア (西洋) 数字として送信されるようにするために行います。有効な値は `true` および `false` です。初期デフォルト値はありません。

disconnecttimeout
HATS がホストとの接続の切断を試みる時間を指定します。秒数を 1 から 2147483647 の範囲で指定します。初期デフォルト値は 120 秒です。

enableScrRev
双方向コード・ページを使用する場合、表示テキストおよび入力フィールドの方向を反転するのに必要な「画面を反転」ボタンを表示するアプリケーションのページを指定します。有効な値は以下のとおりです。

(blank)
「画面を反転」ボタンはどの画面にも配置されません。

Customized
「画面を反転」ボタンは、画面カスタマイズに一致する画面、および画面カスタマイズに一致しない画面に配置されます。つまり、す

べての画面に配置されます。「画面を反転」ボタンを、画面カスタマイズに一致する画面にのみ配置するオプションはありません。

Non-customized
「画面を反転」ボタンは、画面カスタマイズに一致しない画面にのみ配置されます。

初期デフォルト値はありません。

host 接続先ホストの名前を指定します。

hostSimulationName
ライブ接続の代わりに使用するホスト・シミュレーション・トレース・ファイルの名前を指定します。

LUName
拡張 3270 セッションのみ有効 (TNEnhanced="true")。LUName プロパティー (拡張ネゴシエーションで使用する LU 名) を設定します。値はストリング・フォーマットです。LUName の最大長は 17 文字です。デフォルト値はありません。3270 HATS プロジェクトに対して印刷サポートを構成するには、ホスト・タイプとして 3270E を指定する必要があります。接続設定のリストに LUName パラメーターを追加するときは、プリンター LU 名ではなく、ご使用のディスプレイ LU 名またはディスプレイ LU のプールを使用してください。

LUNameSource
拡張 3270 セッションのみ有効 (TNEnhanced="true")。接続の LU 名のソースを指定します。有効な値は以下のとおりです。

automatic
LU 名は、接続が確立された際、自動的に割り当てられます。

prompt
エンド・ユーザーに LU 名の入力を求めるプロンプトが表示されます。プールが使用可能な場合、プロンプトは使用できません。

session
HTTP セッション変数を使用して LU 名を定義します。LUName 属性がセッション変数の名前を表します。プールが使用可能な場合、セッションは使用できません。

value LU 名は LUName 属性で定義します。

初期デフォルト値はありません。

port ホストへの接続に使用するポートの番号を指定します。有効なポートの範囲は、0 から 65535 です。初期デフォルト値は 23 です。

screenSize
ホスト端末に表示する行数とカラム数を指定します。screenSize の有効な値は以下のとおりです。

- 2=24x80
- 3=32x80
- 4=43x80
- 5=27x132

- 6=24x132 (VT のみ)

初期デフォルトの画面サイズは、24 x 80 です。

sessionType
ホスト端末に表示する端末のタイプを指定します。type の有効な値は以下のとおりです。

- 1=3270
- 2=5250
- 3=VT

初期デフォルト値は 3270 です。

singlelogon
プロジェクトにユーザー・リストが定義されている場合、1 つのユーザー ID を一度に複数回使用可能にするかどうかを指定します。有効な値は以下のとおりです。

true ユーザー ID は、一度に 1 回しか使用できません。

false ユーザー ID は、同時に複数回接続することができます。

初期デフォルト値は `false` です。

SSL SSL を使用可能にするかどうかを指定します。有効な値は以下のとおりです。

true このプロジェクトに対して SSL が使用可能にされます。

false このプロジェクトに対して SSL は使用可能にされません。

TNEnhanced
3270 接続でのみ有効です。接続が TN3270E 接続かどうかを指定します。有効な値は `true` および `false` です。初期デフォルト値は `true` です。

VTTerminalType
VT 接続でのみ有効です。VT 端末のタイプを表します。有効な値は以下のとおりです。

- 1=VT420_7
- 2=VT420_8
- 3=VT100
- 4=VT52

WFEnabled
5250 接続でのみ有効です。その接続が 5250W 接続であるかどうかを指定します。有効な値は `true` および `false` です。初期デフォルト値は `false` です。

workstationID
5250 および 5250W 接続でのみ有効です。workstationIDSource 属性が session または value に設定されている場合は、接続用に HTTP セッション変数またはワークステーション ID を指定します。初期デフォルト値はありません。

workstationIDSource

5250 および 5250W 接続でのみ有効です。接続用のワークステーション ID のソースを指定します。有効な値は以下のとおりです。

automatic

ワークステーション ID は、接続が確立された際、自動的に割り当てられます。

prompt

エンド・ユーザーにワークステーション ID の入力を求めるプロンプトが表示されます。プールが使用可能な場合、プロンプトは使用できません。

session

HTTP セッション変数を使用してワークステーション ID を定義します。ワークステーション ID の属性がセッション変数の名前を表します。プールが使用可能な場合、セッションは使用できません。

value ワークステーション ID は workstationID 属性で定義します。

初期デフォルト値はありません。

<otherParameters> タグ

<otherParameters> タグは、その他の Host On-Demand セッション・パラメーターを指定します。

HATS がサポートする Host On-Demand セッション・パラメーターには以下のものがあります。

ENPTUI

5250 ホストに接続したプロジェクトで、拡張非プログラマブル端末ユーザー・インターフェース (ENPTUI) 用の表示データ・ストリーム (DDS) のキーワードを使用できるかどうかを決めます。有効な値は `true` および `false` です。デフォルト値は `false` です。

Lamalef

LamAlef プロパティー (LamAlef を展開するか圧縮するか) を設定します。このプロパティーはアラビア語セッションのみに適用されます。値はストリング・フォーマットです。有効な値は以下のとおりです。

- LAMALEF_ON
- LAMALEF_OFF

デフォルト値は LAMALEF_OFF です。

numeralShape

numeralShape プロパティーを設定します。このプロパティーは双方向セッションのみに適用されます。値はストリング・フォーマットです。デフォルト値は NOMINAL です。

numericSwapEnabled

数値スワッピング・プロパティーを設定します。このプロパティーはアラビア語 3270 セッションのみに適用されます。有効な値は `true` および `false` です。デフォルト値は `true` です。

roundTrip

roundTrip プロパティーを設定します。このプロパティーは双方向セッションのみに適用されます。値はストリング・フォーマットです。有効な値は以下のとおりです。

- ROUNDTRIP_ON
- ROUNDTRIP_OFF

デフォルトは ROUNDTRIP_ON です。

SecurityProtocol

SecurityProtocol プロパティーを設定します。このプロパティーは、セキュリティー確保のために TLS v1.0 プロトコルまたは SSL プロトコルのどちらを使用するかを示します。値はストリング・フォーマットです。デフォルト値は TLS です。

SSLServerAuthentication

SSLServerAuthentication プロパティーを設定します。このプロパティーは、SSL サーバーを使用可能にするかどうかを示します。有効な値は `true` および `false` です。デフォルト値は `false` です。

symmetricSwapEnabled

対称スワッピング・プロパティーを設定します。このプロパティーはアラビア語 3270 セッションのみに適用されます。有効な値は `true` および `false` です。デフォルト値は `true` です。

textOrientation

textOrientation プロパティーを設定します。このプロパティーは双方向セッションのみに適用されます。値はストリング・フォーマットです。有効な値は以下のとおりです。

- LEFT_TO_RIGHT
- RIGHT_TO_LEFT

デフォルト値は LEFT_TO_RIGHT です。

ThaiDisplayMode

タイ語表示モード・プロパティーを設定します。このプロパティーはタイ語セッションのみに適用されます。値はストリング・フォーマットです。デフォルト値は THAI_MODE_5 です。

workstationID

workstationID プロパティーを設定します。このプロパティーは、5250 用拡張ネゴシエーションで使用されます。値はストリング・フォーマットです。小文字はすべて大文字に変換されます。デフォルト値はありません。

Kerberos チケットのサポート

Kerberos チケットのサポート (5250 を使用するリッチ・クライアント・プロジェクトの場合のみ) を有効にするには、application.hap ファイルの `<otherParameters>` タグの内側に以下の行を追加します。

```
<parameter name="ssoEnabled" value="true"/>

<parameter name="ssoType" value="ssoAcquireKerberosTicket"/>
```

<classSettings> タグ

<classSettings> タグは、接続定義に含める Java クラスを囲むタグです。

<class> タグ

<class> タグは、<settings> タグで囲まれた属性を持つクラスを指定します。

<class> タグの属性には以下のものがあります。

name 次のいずれかの Java クラスを指定します。

- com.ibm.hats.common.HATSPrintSettings
- com.ibm.hats.common.NextScreenSettings

name 属性に指定するクラス名は引用符で囲む必要があります。

<setting> タグ

<setting> タグは、<setting> タグを囲んでいるクラスに関連する設定を指定します。

<setting> タグの属性には以下のものがあります。

name <class> タグの name 属性によって定義したクラスのカスタマイズ可能な設定の名前を指定します。使用可能な設定は、クラスによって異なります。

com.ibm.hats.common.HATSPrintSettings クラスの場合、カスタマイズ可能な設定は以下のとおりです。

printFontName
出力の印刷に使用するフォントを指定します。有効な値は、codePage 属性の値によって異なります。

printNumSwapSupport
数値スワッピングを使用可能にするかどうかを指定します。printRTLSupport が使用可能な場合、このプロパティーはアラビア語 3270 セッションのみに適用されます。英語の数表示は RTL 画面でアラビア数字に置き換わり、アラビア数字は RTL 画面で英語の数表記に置き換わります。有効な値は `true` および `false` です。デフォルト値は `true` です。

printOrientation
印刷出力をページ上にどのように配置するかを指定します。printOrientation の有効な値は以下のとおりです。

PDF_ORIENTATION_PORTRAIT
用紙を縦に使用します。

PDF_ORIENTATION_LANDSCAPE
用紙を 90 度右回転します。

printPaperSize
出力を印刷する用紙のサイズを指定します。printPaperSize の有効な値は以下のとおりです。

ISO_A3
ISO/DN & JIS A4、297 x 420 mm

ISO_A4
ISO/DN & JIS A4、210 x 297 mm

ISO_A5
ISO/DN & JIS A4、148 x 210 mm

ISO_B4
ISO/DN B4、250 x 353 mm

ISO_B5
ISO/DN B5、176 x 250 mm

JIS_B4
JIS B4、257 x 364 mm

JIS_B5
JIS B5、182 x 257 mm

ISO_C5
ISO/DN C5、162 x 229 mm

ISO_DESIGNATED_LONG
ISO/DN Designated Long、110 x 220 mm

EXECUTIVE
Executive、7 1/4 x 10 1/2 in

LEDGER
Ledger、11 x 17 in

NA_LETTER
North American Letter、8 1/2 x 11 in

NA_LEGAL
North American Legal、8 1/2 x 14 in

NA_NUMBER_9_ENVELOPE
North American #9 Business Envelope、3 7/8 x 8 7/8 in

NA_NUMBER_10_ENVELOPE
North American #10 Business Envelope、4 1/8 x 9 1/2 in

MONARCH_ENVELOPE
Monarch Envelope、3 7/8 x 7 1/2 in

CONTINUOUS_80_COLUMNS
Data Processing 80 Columns Continuous Sheet、8 x 11 in

CONTINUOUS_132_COLUMNS
Data Processing 132 Columns Continuous Sheet、13 1/5 x 11 in

printRTLSupport
RTL 印刷のサポートを使用可能にするかどうかを指定します。このプロパティーはアラビア語 3270 セッションのみに適用されます。

双方向ファイルは RTL ファイルまたは LTR ファイルのどちらでもかまいません。有効な値は `true` および `false` です。デフォルト値は `true` です。

printSupport
プロジェクトに印刷機能が含まれているかどうかを指定します。printSupport の有効な値は、`true` および `false` です。初期デフォルト値は `false` です。

printSymSwapSupport
対称スワッピングが使用可能かどうかを指定します。スワッピング文字は RTL 画面でスワップされます。printRTLSupport が使用可能な場合、このプロパティーはアラビア語 3270 セッションのみに適用されます。有効な値は `true` および `false` です。デフォルト値は `true` です。

printURL
5250 サーバーの System i® Access for Web プリンター出力ウィンドウの URL を指定します。デフォルトの URL は、http://*hostname*/webaccess/iWASpool です。*hostname* は 5250 サーバーの名前です。

com.ibm.hats.common.NextScreenSettings クラスの場合、カスタマイズ可能な設定は以下のとおりです。

default.appletDelayInterval
非同期更新モードで実行中のセッションについて、フル・ホスト画面が到達するまでにサーバーが待機する最大時間 (ミリ秒単位) を指定します。初期デフォルト値は 400 ミリ秒です。

default.blankScreen
接続開始時に受信する空白画面の取り扱い方法を指定します。有効な値は以下のとおりです。

normal
空白画面を表示します。

sendkeys
default.blankScreen.keys 設定に定義されたホスト・キーを送信します。

timeout
接続がタイムアウトになり、エラー・メッセージが発行されるまで待機します。

デフォルトは `normal` です。

default.blankScreen.keys
default.blankScreen に sendkeys を設定した場合、送信するホスト・キーを指定します。

default.delayInterval
初期画面更新後に次の画面更新に到達するまでにサーバーが待機する最大時間 (ミリ秒単位) を指定します。初期デフォルト値は 1200 ミリ秒です。

default.delayStart
ホスト接続の準備が完了してから最初のフル・ホスト画面が到達するまでにサーバーが待機する最小時間 (ミリ秒単位) を指定します。初期デフォルト値は 2000 ミリ秒です。

nextScreenClass
正確性を重視する場合に、速度が最適化されたデフォルトのアルゴリズムをオフにするクラスを指定します。value 属性のクラスは、com.ibm.hats.runtime.TimingNextScreenBean です。その結果、画面遷移が遅くなる可能性があります。`default.delayInterval` 設定では、画面遷移に要する時間は最小 (ミリ秒単位) に設定されています。`default.delayInterval` のデフォルト値は 1200 ミリ秒ですが、ネットワークやホスト・アプリケーションに合わせてカスタマイズできます。この値を大きくした場合、HATS はホスト画面が安定するまで最低でもこの時間待機することに注意してください。

oiaLockMaxWait
ホスト画面が安定し、OIA システム・ロックの状況が確実にリリースされた後 HATS が待機する最大時間 (ミリ秒単位) を指定します。値の範囲は、0 ミリ秒から 600000 ミリ秒です。初期デフォルト値は 300000 ミリ秒です。

value この設定の値は、個別の設定の記述に組み込まれています。

<poolsettings> タグ

<poolsettings> タグは、接続のプール・パラメーターを定義します。

<poolsettings> タグの属性には以下のものがあります。

enabled
この接続でプールが有効かどうかを指定します。enabled の有効な値は、`true` および `false` です。初期デフォルト値は `false` です。

maxbusytime
エンド・ユーザーが使用している接続が終了するまでの秒数。アクティブ接続を終了させない場合は、このフィールドを -1 に設定します。この設定は、プール可能な接続と、プール不能な接続の両方で有効です。プール可能な接続の場合、プール内の有効接続数が、接続状態のまま保持するように指定した最小接続数よりも少なくなると、接続がプールに戻ります。そうでない場合、この接続は廃棄されます。プール不能な接続の場合、接続は廃棄されます。有効な秒数は -1 または 60 から 2147483647 の範囲です。デフォルトは -1 (最大使用中時間なし) です。

maxconnections
プール内でアクティブにすることのできる接続の最大数。この設定は、プール可能な接続でのみ有効です。有効な接続の数は 1 から 2147483647 の範囲です。デフォルトは、1 です。指定した最大値に達した場合、追加の接続要求を受け取ると、HATS では、次の有効な接続を待機するか、新しい接続を作成することができます。

maxidletime
エンド・ユーザーが使用していない接続が、終了してプールから除去される

までの秒数。非アクティブ接続を終了させない場合は、このフィールドを -1 に設定します。この設定は、プール可能な接続でのみ有効です。最小接続数に指定した数の接続は、使用中かどうかに関係なく、接続状態のまま保たれます。有効な秒数は -1 または 60 から 2147483647 の範囲です。デフォルトは -1 (最大アイドル時間なし) です。

minconnections
プール内、接続状態のまま保持するアイドル接続の数。この設定は、接続がプール可能であり、かつ、切断までの最大アイドル時間が -1 以外の値に設定されている場合にのみ有効です。有効な接続数は、0 から 2147483647 です。デフォルトは 0 (接続を接続状態のまま保持しない) です。

overflowallowed
最大接続数に達した場合に、非プール接続を新たに作成するかどうかを指定します。この値が false の場合は、プール接続が使用可能になるまでに待機する秒数を指定する必要があります。待機時間が経過しても接続が使用可能にならない場合、HATS からエラーが戻ります。この値が true の場合は、非プール接続が新たに作成されます。エンド・ユーザーがこのタイプの接続の使用を終了したとき、接続はプールに戻されずに廃棄されます。

waittimeout
最大接続数に達した場合、別の接続要求が到着したときに、プール接続が使用可能になるまでに待機する秒数。有効な秒数は、0 から 2147483647 までか、-1 (いつまでも待機する) です。デフォルト値は 120 です。

<userconfig> タグ

<userconfig> タグは、接続のユーザー・リストを定義します。 <userconfig> タグの内側のタグおよびデータは複雑であるため、手動編集を行うと破損する可能性があります。ユーザー・リストの整合性を保護するために、<userconfig> データは手動編集しないでください。ユーザー・リストを作成または変更する場合には、代わりに接続エディターの「ユーザー・リスト」タブを使用してください。デフォルトではホスト接続にこのタグは指定されておらず、ユーザー・リストはありません。

画面組み合わせファイル (.evnt)

画面組み合わせファイルは、ホスト画面をどのように認識するかを定義するほか、画面が認識されたときに HATS が実行するアクション、画面組み合わせの終了を定義する方法、および画面間をナビゲートする方法も定義します。画面組み合わせ (.evnt) ファイルは *project_name*/**profiles/events/screencombinations** ディレクトリーに格納されています。画面組み合わせファイルのソースを表示および編集するには、HATS Projects 表示で目的の画面組み合わせの名前をダブルクリックして、画面組み合わせエディターを開きます。そして、「ソース」タブをクリックして、ファイルのソースを表示することができます。エディターで、「開始画面」、「レンダリング **(Render)**」、「ナビゲーション」、「終了画面」、「アクション」、「テキスト置換」、または「ソース」タブを使用して、画面組み合わせファイルを変更することができます。いずれかのタブで変更を行うと、HATS Toolkit はその他のタブ上で該当情報を更新します。画面組み合わせイベント・ファイルには、ホスト画面をどのように認識するか、およびホスト画面が認識されたときに行うアクションとナビゲーションを定義するタグが含まれています。

画面組み合わせにより、いくつかのタグが画面カスタマイズ (.evnt) ファイルで検出されたタグに追加されます。

<combinations> タグ

これは、組み合わせ情報のコンテナーです。 type 属性および画面組み合わせコンポーネントの詳細を説明するレンダリング項目で構成されます。

type type パラメーターにより、画面変換が集約される方法を判別します。

文字列値が *dynamic* に設定されている場合、ユーザーが画面変換を使用している間、画面変換によって、結合領域に画面を追加できます。

文字列値が *normal* に設定されているか、または欠落している場合、ユーザーが画面変換と相互作用できるようになる前に、個々の画面が結合します。リッチ・クライアント画面結合は通常処理に制限されます。

<enddescription> タグ

これは、画面結合終了画面に達したかどうかを判別するために使用する画面基準について説明します。タグおよび詳細は description タグと一致します。終了画面に関連した画面には、属性 associatedScreen があります。

associatedScreen
これは、終了画面に関連した画面キャプチャーです。

<navigation> タグ

ナビゲーションには、データの収集および配置を行うために、画面間を移動するのに必要なコマンドが含まれます。 screenUp および screenDown タグで構成されます。

<screenUp> タグ

組み合わせ内で画面を逆方向にトラバースするのに必要なコマンド。これは、データを画面組み合わせ内の正しい位置に戻すために使用されます。
keyPress、setCursor、および sendText タグで構成されます。

<screenDown> タグ

組み合わせ内で画面を順方向にトラバースするのに必要なコマンド。これは、データを画面組み合わせ内の正しい位置に戻すと同時に、画面組み合わせビューを作成するために使用されます。 keyPress、setCursor、および sendText タグで構成されます。

<keyPress> タグ

このナビゲーション・コマンドは sendKey に相当します。 HOD コマンド送信用の value 属性があります。これは、有効な HOD キー・コマンドでなければなりません。

value keyPress タグの value 属性は、有効な HOD キー・コマンドでなければなりません。

<setCursor> タグ

このナビゲーション・コマンドにより、カーソルを画面上に配置できます。カーソル配置用の row および column 属性があります。

row row 属性は、画面上の位置に相当する、1 を基準とした整数でなければなりません。これは、カーソル位置の垂直コンポーネントを位置決めします。

column
column 属性は、画面上の位置に相当する、1 を基準とした整数でなければなりません。これは、カーソル位置の水平コンポーネントを位置決めします。

<sendText>

このナビゲーション・コマンドは sendKey に相当します。テキスト送信用の value 属性があります。これは、ホスト・フィールドの有効なテキストでなければなりません。

value sendText タグの value 属性は、ホスト・フィールドの有効なテキストでなければなりません。

画面カスタマイズ・ファイル (.evnt)

画面カスタマイズ・ファイルは、ホスト画面をどのように認識するかを定義するほか、画面が認識されたときに HATS が行うアクションも定義します。

画面カスタマイズ (.evnt) ファイルは、*project_name*/**profiles/events/screencustomizations** ディレクトリーに格納されています。画面カスタマイズ・ファイルのソースを表示および編集するには、「**HATS** プロジェクト表示」で目的の画面カスタマイズの名前をダブルクリックして、画面カスタマイズ・エディターを開きます。そして、「ソース」タブをクリックして、ファイルのソースを表示することができます。

エディターで、「画面認識基準」、「アクション」、「テキスト置換」、または「ソース」タブを使用して、画面カスタマイズ・ファイルを変更することができます。いずれかのタブで変更を行うと、HATS Toolkit はその他のタブ上で該当情報を更新します。

画面カスタマイズ・イベント・ファイルには、ホスト画面をどのように認識するか、およびホスト画面が認識されたときに行うアクションを定義するタグが入っています。

<event> タグ

画面カスタマイズの定義を開始します。event タグには以下の属性があります。

description
画面カスタマイズを作成するときに説明を付加してある場合は、この属性でその説明が定義されます。

type 画面カスタマイズの場合は、type は常に screenRecognize です。結合画面の場合、type は screenCombination です。

<actions> タグ

これは、画面カスタマイズ用に定義されたすべてのアクションを囲むタグです。このタグには属性はありません。

<apply> タグ

変換を適用するためのアクションを定義します。<apply> タグの属性には以下のものがあります。

applyGlobalRules
ホスト画面で HATS がグローバル規則を検索するかどうかを指定します。デフォルトは `true` です。

applyTextReplacement
このホスト画面で HATS がテキスト置換を検索するかどうかを指定します。デフォルトは `true` です。テキスト置換の使用方法について詳しくは、<replace> タグを参照してください。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

immediateKeyset
プロジェクトのエンド・ユーザーが押すと同時にホストに送られるホスト・キーを定義します。ホストにただちに送信するホスト・キーを定義していない場合は、この属性の値は空です。

template
適用する変換を囲むテンプレート・ファイルの名前を示します。デフォルトのテンプレートを使用して変換を囲む場合は、この属性の値は空です。

transformation
このアクション用として適用する変換ファイルの名前を示します。

<insert> タグ

グローバル変数またはストリングを挿入するためのアクションを定義します。<insert> タグの属性には以下のものがあります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

row ホスト画面上で値を挿入する位置の開始行を定義します。

col ホスト画面上で値を挿入する位置の開始カラムを定義します。

source
挿入する値が、ストリングか、グローバル変数の値かを指定します。有効な値は string および variable です。

value ホスト画面に挿入するストリングか、値がとられるグローバル変数の名前を指定します。

fill 挿入する値のソースが索引付きグローバル変数である場合、fill は、グローバル変数の索引を連結して指定位置に挿入するか、ホスト画面上の長方形の領域に挿入するかを指定します。有効な値は concatenate および rectangular です。

index 挿入する値のソースが索引付きグローバル変数である場合に、index は、ホスト画面に挿入する値として使用する索引の番号を指定します。

shared
挿入する値のソースがグローバル変数である場合、shared は、このグローバル変数を同じリッチ・クライアント環境内で稼働するアプリケーション間で共用するかどうかを指定します。

<extract> タグ

グローバル変数を抽出するためのアクションを定義します。<extract> タグの属性には以下のものがあります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

srow ホスト画面上から抽出するテキストの開始行を定義します。

erow ホスト画面上から抽出するテキストの終了行を定義します。

scol ホスト画面上から抽出するテキストの開始カラムを定義します。

ecol ホスト画面上から抽出するテキストの終了カラムを定義します。

name 抽出したテキストを入れるグローバル変数の名前を指定します。これは、既存のグローバル変数でも新規のグローバル変数でも構いません。

overwrite
既存のグローバル変数の値を抽出したテキストで上書きするかどうかを指定します。有効な値は `true` および `false` です。

indexed
抽出するテキストが単一のストリングか、ストリングのリストかを指定します。リストの場合、各ストリングは抽出された領域内の 1 行のテキストに対応します。有効な値は true および false です。

index 既存のグローバル変数に索引が付いている場合は、この属性は、抽出した値が書き込まれる索引番号を指定します。この属性の効果は、**overwrite** 属性の値によって異なります。`overwrite=true` であれば、既存の変数内の指定された索引位置から始まる部分が、抽出された値で上書きされます。`overwrite=false` であれば、既存の変数内の指定された索引位置に、抽出された値が挿入されます。

shared
shared は、グローバル変数を同じリッチ・クライアント環境内で稼働するアプリケーション間で共用するかどうかを指定します。

<set> タグ

グローバル変数を設定するためのアクションを定義します。<set> タグの属性には以下のものがあります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは true です。

name 設定するグローバル変数の名前を指定します。これは、既存のグローバル変数でも新規のグローバル変数でも構いません。

shared
設定するグローバル変数を同じリッチ・クライアント環境内で稼働するアプリケーション間で共用するかどうかを指定します。

type 設定するグローバル変数の値を、定数からとるか計算値からとるかを指定します。有効な値は `string` および `calculate` です。

value このグローバル変数に割り当てる値を指定します。

overwrite
既存のグローバル変数の値を、設定した値で上書きするかどうかを指定します。有効な値は `true` および `false` です。

index 設定した値を既存の索引付きグローバル変数に書き込む場合、この属性は、設定した値を書き込む索引の番号を指定します。この属性の効果は、**overwrite** 属性の値によって異なります。`overwrite=true` であれば、既存の変数内の指定された索引位置から始まる部分が、設定した値で上書きされます。`overwrite=false` であれば、既存の変数内の指定された索引位置に、設定した値が挿入されます。

op1 計算される値の第 1 オペランドが、定数か、既存のグローバル変数の値かを指定します。有効な値は、定数、またはグローバル変数の名前です。

op1_type
計算される値の第 1 オペランドを、定数として設定するか、既存のグローバル変数からの値として設定するかを指定します。有効な値は `string` および `variable` です。

op1_index
計算される値の第 1 オペランドの値のソースが索引付きグローバル変数である場合、op1_index は、計算対象の値として使用する索引の番号を指定します。

op1_shared
op1 の値がグローバル変数である場合、shared は、このグローバル変数を同じリッチ・クライアント環境内で稼働するアプリケーション間で共用するかどうかを指定します。

op 計算される値の第 1 オペランドと第 2 オペランドの間で行う演算のタイプを指定します。有効な値は、+ (加算)、- (減算)、* (乗算)、/ (除算)、% (モジュロ) などの演算を連結したものです。

op2 計算される値の第 2 オペランドが、定数か、既存のグローバル変数の値かを指定します。有効な値は、定数、またはグローバル変数の名前です。

op2_type
計算される値の第 2 オペランドを、定数として設定するか、既存のグローバル変数からの値として設定するかを指定します。有効な値は `string` および `variable` です。

op2_index
計算される値の第 2 オペランドの値のソースが索引付きグローバル変数である場合、op2_index は、計算対象の値として使用する索引の番号を指定します。

op2_shared
op2 の値がグローバル変数である場合、shared は、このグローバル変数を同じリッチ・クライアント環境内で稼働するアプリケーション間で共用するかどうかを指定します。

dec 計算された値を丸めた後に残す小数点以下の桁数を指定します。有効な値は 0 から 999 です。

<execute> タグ

ビジネス・ロジックを実行するためのアクションを定義します。<execute> タグの属性には以下のものがあります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

class ビジネス・ロジックを含める Java クラスの名前を指定します。class 値は必須です。

method
クラス内の、ビジネス・ロジックを実行するメソッドの名前を指定します。method 値は必須です。

package
ファイル・システム上でこの Java クラスが入っているパッケージの名前を指定します。package 値はオプションです。

<show> タグ

URL を表示するためのアクションを定義します。<show> タグには以下の属性があります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

template
このアクションに使用するテンプレートを指定します。

url 表示する Web ページの URL を示します。この属性は必須です。

<forwardtoURL> タグ

プロジェクトから統合オブジェクトを呼び出す JSP に制御を引き渡すためのアクションを定義します。<forwardtoURL> タグには以下の属性があります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

startStateLabel
統合オブジェクト・チェーンを持つ JSP に制御を転送する場合、実行するチェーンの中の最初の統合オブジェクトの開始状態ラベルを指定します。

url 統合オブジェクト JSP の URL を指定します。

<disconnect> タグ

デフォルト接続を切断します。このアクションを使用する場合は注意が必要です。このアクションは、回復できないイベントに対してのみ使用してください。<disconnect> タグには以下の属性があります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

<play> タグ

マクロを実行するためのアクションを定義します。<play> タグには以下の属性があります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

macro 実行するマクロの名前を指定します。この属性は必須です。

<perform> タグ

任意の接続 (デフォルト接続でなくてもよい) に対してマクロを実行するためのアクションを定義します。このアクションは、現在のホスト画面には影響しません。<perform> タグには以下の属性があります。

connection
マクロを実行する接続。デフォルトは `main` です。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

macro 実行するマクロの名前。

<pause> タグ

通常処理を続行するまでの待ち時間のアクションを定義します。<pause> タグには以下の属性があります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

time 通常処理を続行するまでの一時停止時間 (ミリ秒単位) を指定します。

<sendkey> タグ

アクションを実行するためにホスト画面に指定したキーを送信するアクションを定義します。<sendkey> タグには以下の属性があります。

enabled
このアクションが使用可能かどうかを表します。デフォルトは `true` です。

key ホスト画面に送信するキーを示します。

row ホスト画面上でキーを挿入する開始行を定義します。

col ホスト画面上でキーを挿入する開始カラムを定義します。

<globalRules> タグ

<globalRules> タグは、画面イベントに定義するグローバル規則を囲むタグです。このタグには属性はありません。

<rule> タグ

<rule> タグは、グローバル規則を定義します。

画面カスタマイズ・レベル規則用の <rule> タグの属性は、プロジェクト・レベルのグローバル規則用の属性と同じです。しかし、同じ入力フィールドを使用して画面カスタマイズ・レベルとプロジェクト・レベルのグローバル規則を作成すると、画面カスタマイズ・レベル規則の優先順位の方が高くなります。 <rule> タグの属性には以下のものがあります。

associatedScreen
グローバル規則を定義する際の基準となる、プロジェクトの中の画面キャプチャーの名前。

componentSettings
グローバル規則に対して構成する設定 (例えば、認識基準)。

description
グローバル規則の作成時に入力した説明。

enabled
グローバル規則が使用可能かどうかを表します。プロジェクト設定の「レンダリング」ページのチェック・ボックスの状態を反映します。

endCol
このグローバル規則を適用するべきホスト画面の最後のカラムです。-1 は、ホスト画面の右端のカラムを意味します。

endRow
このグローバル規則を適用するべきホスト画面の最後の行です。-1 は、ホスト画面の最後の行を意味します。

name 「プロジェクト設定」の「レンダリング」ページのグローバル規則のリストに表示される名前。

startCol
このグローバル規則を適用するべきホスト画面の最初のカラムです。

startRow
このグローバル規則を適用するべきホスト画面の最初の行です。

transformationFragment
グローバル規則に関連付けられた変換フラグメント・ファイルの名前。このファイルには、ホスト・コンポーネントの変換方法を指定する情報が含まれています。該当するフィールドがホスト画面に存在する場合、この情報は変換に組み込まれます。

type このグローバル規則のパターン・タイプ・コンポーネントで、「グローバル規則の作成」ウィザードの最初のページで取得します。タイプは次のいずれかです。

com.ibm.hats.transform.components.InputFieldByTextPatternComponent
このパターン・コンポーネントは、フィールド近くのテキストを基にしてホスト画面上の入力フィールドを認識します。

com.ibm.hats.transform.components.AllInputFieldsPatternComponent
このパターン・コンポーネントは、ホスト画面上のすべての入力フィールドを認識します。

com.ibm.hats.transform.components.InputFieldBySizePatternComponent
このパターン・コンポーネントは、入力フィールドのサイズを基にしてホスト画面上の入力フィールドを認識します。

それぞれ固有のグローバル規則には、次のタグも含まれています。

componentSettings
<componentSettings> タグは、<rule> タグの型属性で指定したパターン・タイプ・コンポーネント用に定義された設定を囲むタグです。このタグには属性はありません。

setting
<setting> タグは、<rule> タグの型属性で指定したパターン・タイプ・コンポーネント用に定義された設定を囲むタグです。

<setting> タグの属性には以下のものがあります。

name パターン・タイプ・コンポーネントのカスタマイズ可能な設定の名前を指定します。使用可能な設定はコンポーネントによって異なります。

- com.ibm.hats.transform.components.InputFieldByTextPattern Component の場合、name 属性の設定は以下のとおりです。

  caseSensitive
  パターンを認識する前に、テキスト設定で大文字小文字の区別を一致させる必要があるかどうかを指定します。有効な値は `true` および `false` です。デフォルトは `true` です。

  immediatelyNextTo
  変換する入力フィールドを指定します。有効な値は以下のとおりです。

  true 最も近い入力フィールドのみ変換の必要があることを指定します。

  false すべての入力フィールドを変換する必要があることを指定します。

  デフォルトは `false` です。

  location
  テキスト設定で指定した保護フィールドのテキストを、適用するグローバル規則の入力フィールドに対してどの位置に置くかを指定します。有効な値は以下のとおりです。

ABOVE
入力フィールドの上に表示するテキストを指定します。

BELOW
入力フィールドの下に表示するテキストを指定します。

LEFT 入力フィールドの左に表示するテキストを指定します。

RIGHT
入力フィールドの右に表示するテキストを指定します。

デフォルトは RIGHT です。

text ホスト画面の保護フィールドに表示するテキストを指定します。有効な値は、ホスト画面上の保護フィールドの中の任意のテキストです。

- com.ibm.hats.transform.components.AllInputFieldsPattern Component の場合、コンポーネント設定はありません。
- com.ibm.hats.transform.components.InputFieldBySizePattern Component の場合、name 属性の設定は fieldSize です。有効な値は、ホスト画面上の任意の入力フィールドのサイズです。

<associatedScreens> タグ

<associatedScreens> タグは、その後に続く screen タグを囲むタグです。

<screen> タグ

画面カスタマイズに関連した画面を定義します。<screen> タグには以下の属性があります。

name 取り込んだ画面 (画面認識基準およびアクションが定義されているもの) の名前を指定します。

<description> タグ

<description> タグは、画面カスタマイズに関連した説明を囲むためのタグで、<oia> タグと <string> タグから成っています。 description タグには属性はありません。

<oia> タグ

画面カスタマイズ .evnt ファイルの中の <oia> タグは、一致する必要があるオペレーター情報域 (OIA) 条件を指定します。このタグはオプションです。デフォルトでは、禁止状況になるまで待機します。<oia> タグの属性には以下のものがあります。

status NOTINHIBITED の場合は、OIA の禁止を解除しなければ一致は起こりません。DONTCARE の場合は、OIA 禁止状況は無視されます。これは、OIA を

まったく指定しない場合と同じ振る舞いとなります。有効な値は NOTINHIBITED および DONTCARE です。これは必須属性です。

optional
false の場合は、画面認識時にこの記述子は非オプションと見なされます。記述子に複数の非オプションの記述子および複数のオプション記述子が含まれる場合は、非オプション記述子が最初に検査されます。非オプション記述子がすべて一致する場合は、画面が一致します。一致しない非オプション記述子が 1 つでも存在する場合は、オプション記述子が検査されます。オプション記述子のいずれかが一致すると、画面が一致します。それ以外の場合は画面は一致しません。値は true または false でなければなりません。この属性はオプションです。デフォルトは false です。

invertmatch
true の場合は、画面がこの記述エレメントに一致しない (ブール not 演算) 場合にのみ、認識比較を通過します。値は true または false でなければなりません。この属性はオプションです。デフォルトは false です。

<string> タグ

<string> タグは、ストリングに基づいて画面を記述します。<string> タグの属性には以下のものがあります。

value ストリング値。この値は、任意の有効な Unicode 文字を含むことができます。これは必須属性です。

row ストリングの開始行の位置 (絶対位置または長方形の中の位置)。値は数値または数値に評価される式でなければなりません。この値はオプションです。指定しない場合は、マクロ・ロジックは画面全体でストリングを検索します。指定する場合は、カラム位置も指定する必要があります。<erow> 属性および <ecol> 属性を指定して、長方形の領域内のストリングを指定することもできます。

注**:** 負の値も有効です。負の値は、画面の最下部を基準とする相対位置を示します (例えば、-1 は最終行です)。

col ストリングの開始カラムの位置 (絶対位置または長方形の中の位置)。値は数値または数値に評価される式でなければなりません。この属性はオプションです。

erow 長方形の中でのストリングの終了行位置。値は数値または数値に評価される式でなければなりません。この属性はオプションです。**erow** と **ecol** の両方を指定する場合は、ストリングは長方形の内側に位置します。

ecol 長方形の中でのストリングの終了カラム位置。値は数値または数値に評価される式でなければなりません。この属性はオプションです。**erow** と **ecol** の両方を指定する場合は、ストリングは長方形の内側に位置します。

casesense
true の場合は、ストリングの比較時に大文字と小文字を区別します。この値は、true、false、または true か false に評価される式でなければなりません。この属性はオプションです。デフォルトは false です。

optional
false の場合は、画面認識時にこの記述子は非オプションと見なされます。

記述子に複数の非オプションの記述子および複数のオプション記述子が含まれる場合は、非オプション記述子が最初に検査されます。非オプション記述子がすべて一致する場合は、画面が一致します。一致しない非オプション記述子が 1 つでも存在する場合は、オプション記述子が検査されます。オプション記述子のいずれかが一致すると、画面が一致します。それ以外の場合は画面は一致しません。この値は、true、false、または true か false に評価される式でなければなりません。この属性はオプションです。デフォルトは false です。

invertmatch

true の場合は、画面がこの記述エレメントに一致しない (ブール not 演算) 場合にのみ、認識比較を通過します。この値は、true、false、または true か false に評価される式でなければなりません。この属性はオプションです。デフォルトは false です。

<nextEvents> タグ

<nextEvents> タグは、その後に続く <event> タグを囲むタグです。<nextEvents> タグには以下の属性があります。

defaultEvent

event タグで指定された画面カスタマイズに一致するものがない場合、表示する次の画面として使用するデフォルトの画面カスタマイズ (イベント) を指定します。defaultEvent でイベントを指定しなかった場合、プロジェクト設定の通常のイベント優先順位リストが使用されます。有効な値は以下のとおりです。

- unmatchedScreen
- error
- disconnect
- stop
- (値なし)

<event> タグ

発生が予想される次の画面の画面カスタマイズをプロジェクトにもう 1 つ定義します。<event> タグには以下の属性があります。

enabled

name 属性で定義した画面カスタマイズ (イベント) が使用可能かどうかを示します。デフォルトは true です。

name 発生が予想される次の画面の画面カスタマイズの名前を指定します。

<remove> タグ

<remove> タグは、画面カスタマイズ (イベント) に以前追加されたグローバル変数を除去します。 <remove> タグには以下の属性があります。

enabled

name 属性に定義されたグローバル変数で除去操作が使用可能になっているかどうかを示します。デフォルトは true です。

name 除去するグローバル変数の名前を指定します。

remove Type
除去するグローバル変数の型を指定します。型には、oneLocal、oneShared、 allLocal、 allShared、および all などがあります。

マクロ・ファイル (.hma)

マクロ・ファイルは、*project_name*/**profiles/macros** ディレクトリーに格納されます。マクロ・ファイルのソースを表示および編集するには、「**HATS** プロジェクト表示」で目的のマクロの名前をダブルクリックして、マクロ・エディターをオープンします。そして、「ソース」タブをクリックして、ファイルのソースを表示することができます。マクロについて詳しくは、拡張マクロ・ガイドを参照してください。

マクロ・ファイルには、一組の画面を定義するタグが含まれています。タグについては、この後のセクションで説明されます。

<macro> タグ

マクロの定義を開始します。macro タグには属性はありません。

<associatedConnections> タグ

<associatedConnections> タグは、<connection> タグを囲みます。<associatedConnections> タグの属性は次のとおりです。

default
このマクロのデフォルト接続を識別します。

<connection> タグ

<connection> タグは、このマクロが関連付けられている接続を識別します。connection タグの属性は以下のとおりです。

name このマクロが関連付けられている接続の名前を識別します。

<extracts> タグ

<extracts> タグは、その後に続く extract タグを囲むタグです。<extracts> タグには属性はありません。

<extract> タグ

<extract> タグは、実行する抽出を定義します。extract タグの属性には以下のものがあります。

name 抽出の名前を指定します。

handler
抽出した情報をエンド・ユーザーに対して表示するための .jsp ファイルを選択できます。HATS には、default.jsp という名前のデフォルトのマクロ・ハンドラーが組み込まれています。このファイルを見付けるには、HATS Toolkit の「**HATS** プロジェクト表示」タブをクリックし、プロジ

ェクト名を展開し、次に「マクロ」>「マクロ・イベント・ハンドラー」の順に展開します。独自のハンドラーを作成する場合は、必ず HATS ランタイムに制御を戻してください。

showHandler
抽出した情報をエンド・ユーザーに提示するかどうかを指定します。有効な値は true および false です。

shared
抽出するグローバル変数を同じリッチ・クライアント環境内で稼働するアプリケーション間で共用するかどうかを指定します。

save 抽出した情報をグローバル変数に保管するかどうかを指定します。有効な値は `true` および `false` です。

variableName
抽出した情報をグローバル変数に保管する場合に、variableName は新規または既存のグローバル変数の名前を指定します。

overwrite
抽出した情報を既存のグローバル変数に保管する場合に、overwrite は、既存のグローバル変数の現行値を抽出した情報で上書きするのか、抽出した情報を現行値に追加するのかを指定します。有効な値は `true` および `false` です。`True` は、既存のグローバル変数の値を上書きすることを指定します。

index 抽出した値を既存の索引付きグローバル変数に書き込む場合、この属性は、設定した値を書き込む索引の番号を指定します。この属性の効果は、**overwrite** 属性の値によって異なります。overwrite=true であれば、既存の変数内の指定された索引位置から始まる部分が、抽出した値で上書きされます。`overwrite=false` であれば、既存の変数内の指定された索引位置から始まる部分に、抽出した値が挿入されます。

indexed
抽出する情報が、単一のストリングか、ストリングのリストかを指定します。有効な値は `true` および `false` です。`True` は、抽出する情報がストリングのリストであることを指定します。

isBidi マクロの記録に使用する接続が双方向かどうかを指定します。有効な値は `true` および `false` です。

isRtlScreen
双方向画面が RTL であるかどうかを指定します。有効な値は `true` および `false` です。

screenorientation
抽出アクションの方向を指定します。有効な値は `ltr` および `rtl` です。

<prompts> タグ

<prompts> タグは、その後に続く prompt タグを囲むタグです。prompts タグには属性はありません。

<prompt> タグ

<prompt> タグは、表示するプロンプトを定義します。<prompt> タグの属性には以下のものがあります。

name プロンプトの名前を指定します。

handler
エンド・ユーザーに必要な情報の入力を求めるための .jsp ファイルを選択し、ユーザーが情報を送信するためのボタンを組み込むことができます。HATS には、default.jsp という名前のデフォルトのマクロ・ハンドラーが組み込まれています。このファイルを見つけるには、HATS Toolkit の「**HATS** プロジェクト表示」タブをクリックし、プロジェクト名を展開し、「マクロ」>「マクロ・イベント・ハンドラー」の順に展開します。独自のハンドラーを作成する場合は、必ず HATS ランタイムに制御を戻してください。

source
プロンプトの値をストリングに設定するのか、グローバル変数の値に設定するのかを指定します。有効な値は `string` および `variable` です。

variableName
プロンプトの値をグローバル変数に保管する場合に、variableName は新規または既存のグローバル変数の名前を指定します。

variableIndex
プロンプトの値を索引付きグローバル変数に保管する場合に、variableIndex は、値を割り当てる索引を指定します。この値は常に `0` です。

variableIndexed
プロンプトの情報を索引付きグローバル変数から取得するかどうかを指定します。有効な値は `true` および `false` です。`True` はグローバル変数が索引付きであることを示します。

value プロンプトに使用するストリングか、値がとられるグローバル変数の名前を指定します。

welApplID
WEL ログオン・マクロで使用するアプリケーション ID を指定します。

welIsPassword
これが WEL ログオン・マクロで使用されるパスワード・フィールドであるかどうかを指定します。

LTRImpicitOrient
暗黙の双方向画面方向が LTR であるかどうかを指定します。有効な値は `true` および `false` です。

isBidi マクロの記録に使用する接続が双方向かどうかを指定します。有効な値は `true` および `false` です。

isRtlField
双方向フィールドが RTL であるかどうかを指定します。有効な値は `true` および `false` です。

isRtlScreen

双方向画面が RTL であるかどうかを指定します。有効な値は `true` および `false` です。

screenorientation

プロンプト・アクションの方向を指定します。有効な値は `ltr` および `rtl` です。

<HAScript> タグ

<HAScript> タグは、他のマクロ・タグおよび属性を囲むメイン・タグです。マクロ・タグの詳細については、「*HATS* 拡張マクロ・ガイド」を参照してください。

画面キャプチャー・ファイル (.hsc)

画面キャプチャー・ファイルはホスト画面の XML 表現で、画面カスタマイズ、画面組み合わせ、変換、グローバル規則、またはマクロを作成またはカスタマイズするために使用されます。

画面キャプチャー・ファイルは、*project_name*/Screen Captures ディレクトリーに格納されます。これらのファイルを表示するには、「**HATS** プロジェクト表示」で目的の画面キャプチャーの名前をダブルクリックします。画面キャプチャー・ファイルは編集できません。

注: ビデオ端末 (VT) ホスト画面の画面キャプチャーは、Visual Macro Editor を使用して、およびプール構成時のチェックイン画面として、マクロを作成またはカスタマイズするために使用できます。これらの画面キャプチャーは、画面カスタマイズ、画面組み合わせ、変換、デフォルト・レンダリング、またはグローバル規則の作成には使用できません。

BMS マップ・ファイル (.bms および .bmc)

基本マッピング・サポート (BMS) のマップは、顧客情報管理システム (CICS®) の画面定義ファイルです。各マップが 1 つの画面の全体または一部を定義しており、通常 CICS アプリケーションは 1 つ以上のマップを表示して、完全な画面イメージを作成します。BMS マップのソースは、マップ・セットというグループに編成されます。1 つのマップ・セットには、1 つ以上のマップが含まれています。マップ・セットは、1 つのソース・ファイルに対して 1 つのマップ・セットというようにソース形式で存在します。

BMS マップ・セット・ファイルは HATS Toolkit のプロジェクトにインポートできます。HATS が BMS マップをインポートする際、インポートはマップ・セットのレベルで行われます。マップ単位で個別にインポートすることはできません。HATS Toolkit では、インポートされた BMS マップ・セット・ファイルのファイル拡張子は .bms で、個々のマップのファイル拡張子は .bmc になります。

マップ・セット・ファイル (.bms) とマップ・ファイル (.bmc) は両方とも、HATS プロジェクト内の 1 つの独立した **Maps** フォルダーに格納されます。デフォルトでは、マップがインポートされるまで、「**HATS** プロジェクト表示」では **Maps** フォルダーは見えません。

HATS を使用すれば、マップ・ファイルから画面キャプチャーを生成することができます。画面キャプチャーは、マップ・セットをインポートする際に生成することもでき、**Maps** フォルダーに入れた後、マップから生成することもできます。マップから画面キャプチャーを生成するには、マップ・ファイル上で右マウス・ボタンをクリックしてポップアップ・メニューを表示し、「画面キャプチャーの生成」を選択します。選択した BMS マップごとに個別の画面キャプチャーを作成することもでき、選択した複数の BMS マップを 1 つの画面キャプチャーにマージすることもできます。フィールドがオーバーラップしている場合は、マップをマージすることはできません。画面キャプチャーを作成すれば、HATS 画面カスタマイズの作成を開始できます。

「**HATS** プロジェクト表示」内のファイルをダブルクリックすることで、HATS Toolkit エディターでマップ・セット・ファイルを開くことができます。ファイルの内容について詳しくは、「*CICS* アプリケーション・プログラミング・リファレンス」を参照してください。マップ・セット・ファイルをテキスト・エディターで変更し保管した場合、そのファイルを構成するマップは再生成されますが、1 つ例外があります。マップ・セット・ファイル内でラベル定義されたフィールドの内容が変更されたマップ・ファイルの場合です。このようなマップを再生成するには、「BMS マップ・セットのインポート (BMS map set import)」ウィザードを使用してソース・ファイルを再度インポートする必要があります。

CICS アプリケーションの実行時に、ラベル定義されたフィールドの内容が変更される場合があります。CICS アプリケーションの実行時に表示されるとおりのフィールドを持つ画面キャプチャーを作成する必要がある可能性があります。ラベル定義されたフィールドはアプリケーション実行時に変更可能なため、マップ・セット・ファイル (およびマップ・セット内にあるマップ・ファイル) に、実際の画面キャプチャーの生成に必要な情報のすべてが含まれているとは限りません。マップ・ファイルの編集はできませんが、ファイルをダブルクリックすることで、ファイル・プレビューが開きます。HATS Toolkit のプロパティー・シート表示によって、欠落情報を追加し、フィールドの内容を手動で設定できます。フィールドの内容を変更することによって、単一のマップを複数の画面キャプチャーに使用することができます。

注**:**

1. HATS Toolkit でマップ・ファイルをプレビューする際、プロパティー・シート表示に表示されているフィールドは、「**HATS** プロジェクト表示」で強調表示されているマップ・ファイルのフィールドであり、プレビュー・ウィンドウに表示されているマップ・ファイルのフィールドではありません。
2. 画面キャプチャー・ファイルが BMS マップ・ファイルから生成されている場合は、HATS Toolkit のプロパティー・シート表示を使用して画面キャプチャー・ファイルを表示することも可能です。

イメージ・ファイル (.gif、.jpg、または .png)

イメージ・ファイルは、HATS Toolkit でプロジェクトのユーザーに表示するページを作成するために、テンプレート・ファイルの中で使用されます。

イメージ・ファイルは、*project_name*/**images** ディレクトリーに格納されます。イメージ・ファイルを表示するには、イメージの名前をダブルクリックします。

スプレッドシート・ファイル (.csv または .xls)

.csv (コンマ区切りの値) または .xls (Microsoft Excel) フォーマットのスプレッドシート・ファイルは、TableWidget 内で定義されたホスト画面データから自動的に生成することができます。スプレッドシート・ファイルは HATS SpreadsheetGeneratorServlet によって作成され、ユーザーが定義済みのボタンまたはリンクをクリックすると、実行時に表示できます。ダイアログ・ポップアップが表示され、ユーザーはスプレッドシート・ファイルを格納するディレクトリーおよびファイル名を入力できます。

スプレッドシート・ファイルの作成の詳細については、「*HATS* ユーザーと管理者のガイド」の『「テーブル」ウィジェット』を参照してください。

ホスト・シミュレーション・トレース・ファイル (.hhs)

ホスト・シミュレーション・トレース・ファイルはシミュレートされたホスト接続環境に保管され、ライブ・ホスト接続を使用するのではなく、このファイルを使用して HATS を実行します。シミュレーションは Host Simulator Recorder によって作成されます。これは、実際のホストと HATS 端末の間のプロキシーとして動作します。ホスト・シミュレーション・トレース・ファイルは、ファイル拡張子 .hhs を付けて XML フォーマットで作成され、**HATS Projects** 表示からアクセスする「ホスト・シミュレーション」 フォルダーの以下のに格納されます。

- RCP プロジェクト - *Project_name*/profiles/hostsimulations

ホスト・シミュレーション・トレース・ファイルの作成の詳細については、「*HATS* ユーザーと管理者のガイド」を参照してください。

ComponentWidget.xml

ComponentWidget.xml ファイルには、HATS で提供されるすべてのホスト・コンポーネントおよびウィジェットの定義が格納されています。独自のホスト・コンポーネントまたはウィジェットを追加したときは、このファイルを更新する必要があります。このファイルの説明と簡単なサンプルについては、77 ページの『コンポーネントまたはウィジェットの登録』を参照してください。ComponentWidget.xml ファイルは、「ナビゲーター」ビューでプロジェクトの最後の項目として表示されます。このファイルを編集するには、「ナビゲーター」ビューでファイル名をダブルクリックして、「ソース」タブを選択します。

このファイルの内容の説明と使用方法については、77 ページの『コンポーネントまたはウィジェットの登録』を参照してください。

付録 B. 特記事項

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

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

〒103-8510
東京都中央区日本橋箱崎町19番21号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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

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

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

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

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

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

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

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

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

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

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

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

プログラミング・インターフェース情報

この「リッチ・クライアント・プラットフォーム・アプリケーション・プログラマーズ・ガイド」には、プログラムを作成するユーザーが HATS のサービスを使用するためのプログラミング・インターフェースに関する情報があります。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Linux は、Linus Torvalds の米国およびその他の国における登録商標です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社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。

用語集

アクション **(action).** 画面イベントに指定された画面認識基準と一致するホスト画面などの、イベントの結果として管理対象オブジェクトに対してアプリケーションが実行する既定のタスク。アクションのリストは、各イベントの定義の一部である。

ADB. 「アプリケーション・データ・バッファー **(application data buffer)**」を参照。

管理コンソール **(administrative console).** HATS 管理コンソールは、Web ベースのユーティリティーである。管理コンソールで提供されるビューおよび機能を使用すると、HATS Web アプリケーションのためのライセンスと接続の管理、ログ設定とトレース設定、メッセージとトレースの表示、および問題判別を実行できる。

アプリケーション **(application).** 「**HATS** アプリケーション **(HATS application)**」を参照。

アプリケーション・データ・バッファー **(application data buffer).** WebFacing アプリケーションが消費するために WebFacing サーバーによって戻されるデータの形式。

アプリケーション・イベント **(application event).** アプリケーションのライフ・サイクル内での状態変更により起動される HATS イベント。アプリケーション・イベントの例には、ユーザーによる HATS アプリケーションへの最初のアクセス (開始イベント)、またはアプリケーションによる認識できない画面の検出 (一致しない画面イベント) などがある。

アプリケーション・キーパッド **(application keypad).** HATS アプリケーション・レベルの機能を表す一連のボタンまたはリンク。 (「ホスト・キーパッド **(host keypad)**」と対比。)

成果物 **(artifact).** 「リソース **(resource)**」を参照。

バックグラウンド接続 **(background connection).** デフォルト接続以外の、HATS アプリケーションで定義されている接続。 HATS はバックグラウンド接続からの画面は変換しない。 (「デフォルト接続 **(default connection)**」と対比。)

双方向 **(bidi) (bidirectional (bidi)).** 一般に右から左に読み、数字だけは左から右に読むアラビア語やヘブライ語などのスクリプトに関係した表現。

BMS マップ **(BMS map).** CICS の基本マッピング・サポートで使用する画面定義ファイル。 BMS マップは、CICS アプリケーションによりグループとして表示される一連のフィールドを定義する。

ビジネス・ロジック **(business logic).** 高度な機能を実行する Java コード。例えば、Java API 経由でアクセス可能な別のアプリケーション、データベース、またはその他のシステムと対話する機能を実行する。ビジネス・ロジックは、アプリケーションまたは画面イベント内のアクションとして呼び出される。

チェックイン画面 **(checkin screen).** ホスト画面を識別する画面。これは接続プールに戻せると見なされる接続に対してアクティブである必要がある。アプリケーションがチェックイン画面で指定された画面上にない場合、接続は廃棄されるか、またはチェックイン画面で指定されたホスト画面への接続を戻そうとしてリサイクルされる。チェックイン画面は、接続に対して接続プールが指定されている場合にのみ意味がある。

コンポーネント **(component).** ホスト画面のビジュアル要素。コマンド行、ファンクション・キー、または選択リストなど。 HATS アプリケーションは、ホスト・コンポーネントをウィジェットに変換する。

接続 **(connection).** HATS により使用される一連のパラメーター。.hco ファイルに保管され、ホスト・アプリケーションに接続する。 (「デフォルト接続 **(default connection)**」および「バックグラウンド接続 **(background connection)**」も参照。)

接続プール **(connection pool).** 初期状態で維持されるホスト接続のグループ。作成および初期化せずに使用できる。

信任状マッパー **(credential mapper).** Web 高速ログオンのコンポーネント。ネットワーク・セキュリティー層により事前に認証されている、ホスト信任状に対する要求を処理する。 (「ネットワーク・セキュリティー層 **(network security layer)**」を参照。)

DDS マップ **(DDS map).** データ記述仕様 (Data Description Specification) マップの略。これらのマップでは、IBM i 端末アプリケーションの表示スペースのレイアウトおよび動作を定義する。

デバッグ **(Debug).** リッチ・クライアント・プロジェクトの場合、「実行」と同じである。加えて以下の操作を実行できる。

- 端末の表示を使用して、プロジェクトのテスト中にホスト画面ナビゲーションを表示する。
- Rational SDP コンソールにデバッグ・メッセージを表示する。
- アプリケーションを再始動せずに、プロジェクトへの変更 (テンプレートまたは変換の変更など) を表示する。
- ランタイム環境に展開された設定 (runtime.properties ファイル内で定義) を変更せずに、ランタイム設定 (runtime-debug.properties ファイル内で定義) を変更およびテストする。
- HATS ビジネス・ロジックなどの Java コードをステップスルーする。

サーバーでデバッグ **(Debug on Server).** Web プロジェクトの場合、「サーバーで実行」と同じである。加えて以下の操作を実行できる。

- 端末の表示を使用して、プロジェクトのテスト中にホスト画面ナビゲーションを表示する。
- Rational SDP コンソールにデバッグ・メッセージを表示する。
- テスト・サーバー上でアプリケーションを再始動せずに、プロジェクトへの変更 (テンプレートまたは変換の変更など) を表示する。
- ランタイム環境に展開された設定 (runtime.properties ファイル内で定義) を変更せずに、ランタイム設定 (runtime-debug.properties ファイル内で定義) を変更およびテストする。
- HATS ビジネス・ロジックなどの Java コードをステップスルーする。

デフォルト接続 **(default connection).** HATS がホスト・アプリケーション画面を変換してユーザーに提示する接続。「変換接続 **(transformation connection)**」とも言う。 (「バックグラウンド接続 **(background connection)**」と対比。)

デフォルト・レンダリング **(default rendering).** 特定の変換が指定されていないホスト画面のパーツのレンダリングに HATS が使用するメソッド。

展開 **(deploy).** HATS アプリケーションをランタイム環境内で使用できるようにすること。 HATS Web アプリケーションの場合、これには、HATS プロジェクトの Java EE アプリケーション (つまり .ear ファイル) としてのエクスポート、および WebSphere Application Server へのそのインストールが含まれる。HATS リッチ・クライアント・アプリケーションの場合、これには、HATS プロジェクトの Eclipse フィーチャーとしてのエクスポート、および個々のクライアント・システムへのそのインストールが含まれる。インストールは、独立の Eclipse アプリケーションとして行う場合と、更新サイトから既存の Eclipse ランタイム環境に対して行う場合がある。

記述子 **(descriptor).** 「画面認識基準 **(screen recognition criteria)**」を参照。

開発者 **(developer).** HATS Toolkit を使用してアプリケーションを開発する人。アプリケーション開発者または Web 開発者とも呼ばれる。 (「ユーザー **(user)**」と対比。)

Device Runtime Environment (DRE). Lotus Expeditor Client V6.2.0 以前で HATS リッチ・クライアント・アプリケーションを実行するために必要な他のランタイム環境 (J2SE ランタイムなど) を含むパッケージ。DRE は、Lotus Expeditor Client のランタイム環境にインストールされる。

端末の表示 **(display terminal).** 実行時の HATS アプリケーションとホスト・アプリケーションとの対話を監視するために、テストおよびデバッグ中に使用できるホスト画面を表示する端末ウィンドウ。端末ウィンドウ内のホスト画面を使用して、ホスト・アプリケーションと対話することもできる。

Eclipse. ISV および他のツール開発者に、プラグ・コンパチブル・アプリケーション開発ツール用の標準プラットフォームを提供する、オープン・ソース・イニシアチブ。 Eclipse は、http://www.eclipse.org からダウンロードできる。

エディター **(editor).** 既存のデータを変更できるアプリケーション。 HATS Toolkit では、エディターはウィザードで作成されたリソースのカスタマイズに使用できる。

拡張非プログラマブル端末ユーザー・インターフェース **(ENPTUI: Enhanced Non-Programmable Terminal User Interface).** 非プログラマブル端末 (NPT) およびプログラマブル・ワークステーション (PWS) 上で拡張インターフェースを使用可能にし、5250 フルスクリーン・メニュー方式インターフェース上で、5250 表示データ・ストリーム拡張機能を使用する。

エンタープライズ・アーカイブ **(EAR: enterprise archive).** 特殊な Java アーカイブ (JAR)ファイル。Java EE 標準により定義され、Java EE アプリケーションを Java EE アプリケーション・サーバーに展開するために使用される。EAR ファイルには、エンタープライズ Bean、配置記述子、および個々の Web アプリケーション用の Web アーカイブ (WAR) ファイルが含まれる。 (Sun)

Enterprise JavaBeans (EJB). オブジェクト指向の分散エンタープライズ・レベル・アプリケーションの開発およびデプロイメントのために Oracle によって定義されたコンポーネント・アーキテクチャー。 (Oracle)

イベント **(event).** 特定の状態への到達に基づいて一連のアクションを実行する HATS リソース。アプリケーション・イベントと画面イベントの、2 つのタイプの HATS イベントがある。

エクスポート **(export).** アプリケーションの展開に備えて、HATS プロジェクトのリソースを収集すること。リソースは、必要な実行可能コードとともに、(Web アプリケーションの場合は) アプリケーション EAR ファイル、(リッチ・クライアント・アプリケーションの場合は) Eclipse フィーチャーに収集される。

Extensible Markup Language (XML). SGML から派生し、そのサブセットであるマークアップ言語を定義するための標準メタ言語。

GB18030. GB18030 は、新規の中国語文字エンコード規格である。 GB18030 には 160 万の有効なバイト・シーケンスがあり、1、2、または 4 バイトの文字シーケンスをエンコードする。

グローバル規則 **(global rule).** 特定の入力フィールドのレンダリングを特定の基準に基づいて変更する方法を定義する規則。グローバル規則は、カスタマイズ画面およびデフォルトを使ってレンダリングされる画面に使用される。グローバル規則は、プロジェクト・レベルまたは画面イベント・レベルで定義できる。

グローバル変数 **(global variable).** アクション用の情報を含めるために使用する変数。グローバル変数の値は、ホスト画面などから抽出して、テンプレート、変換、マクロ、統合オブジェクト、またはビジネス・ロジックで使用することができる。グローバル変数は、単一値にも配列にもでき、同じブラウザー・セッションを共用する他の HATS アプリケーションと共用できる。

HATS. 「**Host Access Transformation Services**」を参照。

HATS アプリケーション **(HATS application).** ホスト・アプリケーションの特定のバージョンをユーザーに提示するアプリケーション。WebSphere Application Server に展開された Web 対応アプリケーション、WebSphere Portal に展開されたポートレット、または Lotus Notes、Lotus Expeditor Client などの Eclipse リッチ・クライアント・プラットフォームに展開された Eclipse クライアント・サイド処理プラグインの形を取る。 HATS アプリケーションは、HATS プロジェクトから HATS Toolkit で作成され、適用可能な環境に展開される。展開されたアプリケーションは、他のホストまたは e-ビジネス・アプリケーションと対話して、組み合わせた情報をユーザーに提示できる。

HATS EJB プロジェクト **(HATS EJB project).** 他のアプリケーションがホスト・データを入手するために使用できる、HATS EJB および統合オブジェクトを含むプロジェクト。 HATS EJB プロジェクトは、ホスト・アプリケーションからの変換済み画面は示さない。

HATS エントリー・サーブレット (HATS entry servlet). HATS Web アプリケーションをブラウザーで開始するときに処理されるサーブレット。

HATS プロジェクト (HATS project). HATS リソースの集合 (成果物とも呼ばれる)。HATS Toolkit ウィザードを使用して作成され、HATS Toolkit エディターを使用してカスタマイズされ、HATS アプリケーションにエクスポートできる。

HATS Toolkit. Rational SDP 上で稼働する HATS のコンポーネント。これにより、HATS プロジェクトの作業を行って HATS アプリケーションを作成できるようになる。

Host Access Transformation Services (HATS). IBM ソフトウェアのツール・セットであり、ホスト・ベース・アプリケーションおよびデータ・ソースへの Web ベースのアクセスを可能にする。

ホスト・コンポーネント **(host component).** 「コンポーネント **(component)**」を参照。

ホスト・キーパッド **(host keypad).** ファンクション・キーまたは Enter キーなどの、一般にホスト・キーボードから使用できる機能を表す一連のボタンまたはリンク。 (「アプリケーション・キーパッド **(application keypad)**」と対比。)

ホスト・シミュレーション **(host simulation).** ホスト・シミュレーションを利用すると、ホスト・シミュレーション・トレース・ファイルを記録および保管できる。これらのファイルは、ライブ・ホスト接続の代わりに後から使用できる。記録されたトレース・ファイルを再生することにより、以下の操作を実行できる。ホスト端末機能を使用した、画面キャプチャー、画面イベント、および画面変換の作成。ホスト端末機能を使用した、マクロの作成およびテスト。Rational SDP ローカル・テスト環境を使用した、HATS アプリケーションのテスト。他のトレースおよびログと併せて行われる、ランタイム環境内で失敗したシナリオのトラブルシューティングの支援。

ホスト・シミュレーション・トレース **(host simulation trace).** ホスト・シミュレーション・トレース・ファイルは、ホスト画面とトランザクションを記録および保管する。これらのホスト画面とトランザクションは、ライブ・ホスト接続を使用する代わりに後から再生できる。トレース・ファイルは、ホスト端末機能を使用するかまたはランタイム環境内での実行中に記録できる。

ホスト端末 **(host terminal).** HATS Toolkit ツール。特定の HATS 接続と結合されたセッションであり、HATS 開発者は、画面の取り込み、画面カスタマイズの作成、およびマクロの記録に使用できる。

HTML. ハイパーテキスト・マークアップ言語 (Hypertext Markup Language)。

HTML ウィジェット (HTML widget). 「ウィジェット **(widget)**」を参照。

統合オブジェクト **(Integration Object).** 単一または一連のホスト画面との対話をカプセル化する Java Bean。統合オブジェクトはマクロで構成され、従来の (WSDL ベースの) Web サービス、RESTful Web サービス、または HATS EJB プロジェクトに組み込むことができる。統合オブジェクトは、リッチ・クライアント・プラットフォーム・アプリケーションでは使用できない。

相互運用性 **(interoperability).** コンピューターまたはプログラムが他のコンピューターまたはプログラムと連携する能力。

相互運用性ランタイム **(interoperability runtime).** バックエンド・ホストへの共通接続を管理するために、結合 HATS/WebFacing アプリケーションが使用する共通ランタイム。このランタイムは、WebFacing サーバーが戻すデータをアプリケーションの HATS 部分または WebFacing 部分のどちらで処理するかを決定する。

Java Platform, Enterprise Edition (Java EE). エンタープライズ・アプリケーションの開発およびデプロイのために Oracle によって定義された環境。 Java EE プラットフォームは、複数層の Web ベース・アプリケーションを開発するための機能を提供するサービス、アプリケーション・プログラミング・インターフェース (API)、およびプロトコルのセットで構成される。 (Oracle)

JavaServer Faces (JSF). Java で Web ベースのユーザー・インターフェースを作成するためのフレームワーク。Web 開発者は、ページ上に再利用可能な UI コンポーネントを配置し、コンポーネントをアプリケーション・データ・ソースに接続し、クライアント・イベントをサーバー・イベント・ハンドラーに接続することで、アプリケーションを構築できる。 (Oracle)

JavaServer Pages (JSP). サーバー・サイドのスクリプト記述テクノロジー。Web ページ (HTML ファイル) が提供されるときに、Java コードを動的にそのページに組み込んで実行し、クライアントに動的コンテンツを返すことができる。 (Oracle)

JavaServer Pages Standard Tag Library (JSTL). 共通構造化タスクのサポートを提供する標準タグ・ライブラリー。共通構造化タスクには、反復と条件、XML 文書の処理、国際化対応、構造化照会言語 (SQL) を使用したデータベース・アクセスなどがある。(Oracle)

JSF. 「**JavaServer Faces (JSF)**」を参照。

JSP. 「**JavaServer Pages (JSP)**」を参照。

JSR 168. Java ポートレット仕様は、ポータル環境で実行されるポートレットの集約、パーソナライゼーション、プレゼンテーション、およびセキュリティーの要件に対応する。Java ポートレット仕様のバージョン 1.0、Java Specification Request 168 (JSR 168) は、各種ベンダーが提供するポータル・サーバー間でのポートレットの互換性を有効にする標準を定義している。「**JSR 286**」を参照。

JSR 286. Java ポートレット仕様は、ポータル環境で実行されるポートレットの集約、パーソナライゼーション、プレゼンテーション、およびセキュリティーの要件に対応する。Java ポートレット仕様のバージョン 2.0、Java Specification Request 286 (JSR 286) は、ポートレット、リソース・サービス提供、およびその他の拡張機能間の調整など、バージョン 1.0 (JSR 168) の機能を拡張する標準を定義している。「**JSR 186**」を参照。

JSTL. 「**JavaServer Pages Standard Tag Library**」を参照。

キーボード・サポート **(keyboard support).** Web ブラウザーまたはリッチ・クライアント環境内でのアプリケーションの実行中にユーザーが物理キーボードを使用してホストと対話できるようにするために、開発者が利用する機能。開発者は、プロジェクトにホスト・キーパッドまたはアプリケーション・キーパッド (あるいはその両方) を含めるかどうかも決定する。キーパッドが含まれる場合、開発者は、クライアント・インターフェースに含めるキーおよびそれらのキーとキーパッドの表示方法も決定する。

キーパッド・サポート **(keypad support).** ユーザーがキーボード上の物理キーを押した場合と同様にホストと対話したり、アプリケーションに関連するタスク (印刷ジョブの表示や画面の最新表示など) を実行したりできるようにするために、開発者が利用する機能。「アプリケーション・キーパッド **(application keypad)**」および「ホスト・キーパッド **(host keypad)**」も参照。

リンク **HATS/WebFacing** プロジェクト **(linked HATS/WebFacing project).** 単一の HATS Web プロジェクトと単一の WebFacing プロジェクトをリンクさせて作成されたプロジェクト。HATS Web アプリケーションと WebFacing アプリケーションが相互運用されるエンタープライズ・アプリケーションを作成し、5250 バックエンド・ホストへの接続を共用することを目的とする。

Lotus Expeditor Client. Lotus Expeditor 製品のスタンドアロン・クライアント。ユーザーのマシンまたは開発マシンにインストールされる。

Lotus Notes クライアント **(Lotus Notes Client).** Lotus Notes 製品のスタンドアロン・クライアント。ユーザーのマシンまたは開発マシンにインストールされる。

マクロ **(macro).** マクロは、.hma ファイルに保管され、ホストとの対話を自動化する。これはコマンドをホストに送信し、入力フィールドにデータを入力し、ホストからデータを抽出し、ユーザーに代わって画面をナビゲートするために使用できる。

モデル **1 Web** ページ **(Model 1 Web pages).** ユーザーに提示する情報、情報の表示方法を指定する書式設定タグ、およびページの表示順序を制御する論理を含む単一の JSP。(「**Struts Web** ページ **(Struts Web pages)**」と対比。)

ネットワーク・セキュリティー層 **(network security layer).** ユーザーを認証し、IBM Tivoli Access Manager などのネットワーク・リソースへのアクセスを許可するためのソフトウェア。

オペレーター情報域 **(OIA) (Operator Information Area (OIA)).** OIA は、ホスト・セッション画面下部の領域であり、セッション標識とメッセージが表示される。セッション標識は、ワークステーション、ホスト・システム、および接続に関する情報を示す。

パースペクティブ **(perspective).** Rational SDP ワークベンチで、そのワークベンチ内のリソースのさまざまな側面を表示するビューのグループ。 HATS パースペクティブはビューおよびエディターの集合であり、これにより開発者は HATS アプリケーションに属するリソースを作成、編集、表示、および実行できる。

プール **(pooling).** 「接続プール **(connection pool)**」を参照。

ポータル **(portal).** 統合 Web サイトのこと。リンク、コンテンツ、またはサービスなどの Web リソースのカスタマイズ・リストを動的に作成し、特定のユーザーのアクセス許可に基づいて、特定のユーザーが使用できる。

印刷サポート **(print support).** ホスト・セッションと関連付けるプリンター・セッションを指定し、ユーザーがホスト・アプリケーション印刷ジョブを表示、プリンターに送信、またはディスクに保管できるようにするために、開発者が利用する機能。印刷サポートは、デフォルト接続でのみ使用可能である。

プロファイル **(Profile).** リッチ・クライアント・プロジェクトの場合、「実行」と同じである。加えて最も多くの時間を必要とする操作を見つけ、繰り返されるアクションを識別し、冗長さを除去できる。この機能をパフォーマンス分析に使用して、アプリケーションの理解を向上させるために役立てることができる。

サーバーでプロファイル作成 **(Profile on Server).** Web プロジェクトの場合、「サーバーで実行」と同じである。加えて最も多くの時間を必要とする操作を見つけ、繰り返されるアクションを識別し、冗長さを除去できる。この機能をパフォーマンス分析に使用して、アプリケーションの理解を向上させるために役立てることができる。

プロジェクト **(project).** HATS リソースの集合 (成果物とも呼ばれる)。HATS Toolkit ウィザードを使用して作成され、HATS Toolkit エディターを使用してカスタマイズされる。これらのリソースは HATS アプリケーションとしてエクスポートされる。 HATS プロジェクトのタイプとして、Web、ポートレット、EJB、リッチ・クライアントの各プロジェクトと、HATS Web (ポートレットおよび EJB を含む) アプリケーションの管理を目的とする HATS 管理コンソール・プロジェクトがある。「**HATS** プロジェクト **(HATS project)**」または「**HATS EJB** プロジェクト **(HATS EJB project)**」を参照。

Rational Software Delivery Platform (Rational SDP). Eclipse オープン・ソース・プラットフォームをベースにした、e-business アプリケーションを開発するための一貫したツール・セットを提供する IBM ソフトウェア・プロダクト・ファミリー。

レンダリング・セット **(rendering set).** レンダリング・セットは、レンダリング項目の優先順位付きのリストを作成することにより構成される。各レンダリング項目は、1 つの特定の領域を定義する。その領域の内部で、指定されたホスト・コンポーネントが認識され、指定されたウィジェットを使用してレンダリングされる。

リソース **(resource).** HATS プロジェクトに組み込まれる複数のデータ構造のすべて。 HATS リソースには、テンプレート、画面イベント、変換、画面キャプチャー、接続、およびマクロが含まれる。他の Rational SDP プラグインでは、これらを「成果物」と呼ぶこともある。

RESTful Web サービス **(RESTful Web service).** 「**Web** サービス、**RESTful (Web service, RESTful)**」を参照。

リッチ・クライアント **(rich client).** クライアント環境内の Eclipse リッチ・クライアント・プラットフォーム上で実行するために設計されたプラグイン。展開先のプラットフォームのネイティブの外観および振る舞いを利用することにより、ユーザー・エクスペリエンスが向上するように設計されている。

実行 **(Run).** リッチ・クライアント・プロジェクトにおいて、Eclipse、Lotus Notes、または Lotus Expeditor Client インスタンスで HATS リッチ・クライアント・プロジェクトをテストできるようにする Rational SDP の一機能。このモードでは、ランタイム環境に展開されたランタイム設定 (runtime.properties ファイル内で定義) を変更およびテストできる。このモードでのテスト中にランタイム設定に加えた変更はすべて保持され、ランタイム環境に HATS アプリケーションを展開すると有効になることに注意する必要がある。

サーバーで実行 **(Run on Server).** Web プロジェクトの場合、必要に応じて、WebSphere Application Server で HATS Web およびポートレット・プロジェクトをテストできるようにする Rational SDP の機能。このモードでは、ランタイム環境に展開されたランタイム設定 (runtime.properties ファイル内で定義) を変更およびテストできる。このモードでのテスト中にランタイム設定に加えた変更はすべて保持され、ランタイム環境に HATS アプリケーションを展開すると有効になることに注意する必要がある。

ランタイム設定 **(runtime settings).** ランタイム環境に展開されたログ、トレース、および問題判別の設定 (runtime.properties ファイル内で定義)。

画面キャプチャー **(screen capture).** ホスト画面の XML 表現。.hsc ファイルに保管され、画面カスタマイズ、画面組み合わせ、変換、グローバル規則、またはマクロの作成またはカスタマイズに使用される。画面キャプチャーは、ホストに接続していない場合でも HATS プロジェクトの開発に使用できるため、有用である。 HATS 統合オブジェクトと Web サービス・サポートのコアであるマクロの作成にも有用である。

ビデオ端末 (VT) ホスト画面の画面キャプチャーは、Visual Macro Editor を使用して、およびプール構成時のチェックイン画面として、マクロを作成またはカスタマイズするために使用できます。これらの画面キャプチャーは、画面カスタマイズ、画面組み合わせ、変換、デフォルト・レンダリング、またはグローバル規則の作成には使用できない。

画面組み合わせ **(screen combination).** HATS 画面イベントのタイプの 1 つであり、連続する類似のホスト画面から出力データを収集し、それらを組み合わせ、単一の出力ページで表示するように設計されている。 .evnt ファイルに保管される画面組み合わせの定義内容には、組み合わされる開始および終了画面両方の一連の画面認識基準、画面間のナビゲート方法、各画面から収集されるデータの認識およびレンダリングに使用するコンポーネントとウィジェットが含まれる。

画面カスタマイズ **(screen customization).** ホスト画面の認識時に一連のアクションを実行するように設計された、画面イベントのタイプの 1 つ。 .evnt ファイルに保管される画面カスタマイズの定義内容には、ホスト画面を突き合わせるための一連の基準、およびホスト画面がそれらの基準と一致するときに実行されるアクションが含まれる。

画面イベント **(screen event).** 特定の画面認識基準と突き合わせることでホスト画面が認識されるときに起動される HATS イベント。画面カスタマイズと画面組み合わせの、2 つのタイプの画面イベントがある。

画面認識基準 **(screen recognition criteria).** HATS が 1 つ以上の画面と突き合わせるときに使用する一連の基準。ホストが画面を表示するときに、HATS は検索を実行し、現在のホスト画面が、プロジェクト内の画面イベントに対して定義された画面認識基準と一致するかを判別する。 HATS が一致するものを検出した場合、その画面イベントの既定のアクションが実行される。

画面認識基準はマクロの記録プロセスにも使用され、そのコンテキストでは「記述子 **(descriptors)**」と呼ばれることがある。

Secure Sockets Layer (SSL). 通信プライバシーを提供するセキュリティー・プロトコル。 SSL により、クライアント/サーバー・アプリケーションは、盗聴、改ざん、およびメッセージ偽造ができないように設計された方法で通信できる。 SSL は、Netscape Communications Corp. および RSA Data Security, Inc. により開発された。

ソース **(source).** HATS プロジェクトまたはそのリソースの 1 つを定義するマークアップ言語を含むファイル。さらに、各 HATS プロジェクトに含まれるフォルダーの名前。

SSL. 「**Secure Sockets Layer**」を参照。

標準ポートレット **(standard portlets).** Java ポートレット仕様 JSR 168 または JSR 286 で定義されている標準ポートレット API に準拠するポートレット。「**JSR 168**」および「**JSR 286**」を参照。 .

Standard Widget Toolkit (SWT). Java 開発者向けの Eclipse ツールキット。基盤となるオペレーティング・システムのネイティブ・ウィジェットを使用する、移植可能な共通のユーザー・インターフェース API を定義する。

Struts Web ページ (Struts Web pages). Java Web アプリケーションを作成するための The Apache Software Foundation の Struts オープン・ソース・フレームワークを使用して構築された Web ページ。この Web ページ構築方法では、クラス・ファイル (値を設定し、getter と setter を格納する)、入出力 JSP、および Web ダイアグラム (Web ページの流れと論理を表示する) を作成する。(「モデル **1 Web** ページ **(Model 1 Web pages)**」と対比。)

SWT. 「**Standard Widget Toolkit**」を参照。

システム画面 **(system screen).** データ記述仕様 (DDS) ディスプレイ・ファイル・ソース・メンバーを使用できない IBM i 画面。システム画面は、WebFacing 化された IBM i 上のアプリケーションに固有の画面である。

テンプレート **(template).** .jsp ファイル (Web プロジェクトの場合) または .java ファイル (リッチ・クライアント・プロジェクトの場合) に保管されるテンプレート。アプリケーションの基本レイアウト、および色やフォントなどのスタイルを制御する。さらに、バナーやナビゲーション域などの、GUI に共通の領域の外観も定義する。

テキスト置換 **(text replacement).** HATS 画面変換時にホスト・システム上のテキストをイメージ、HTML コード、または他のテキストに変換するために使用される HATS 機能。

テーマ **(theme).** テーマでは、プロジェクトの共通の外観および振る舞いの特性一式がグループ化される。これらの属性は後から個別に変更できる。

転送 **(transfer).** アプリケーション EAR ファイルをサーバーにコピーすること。一般には FTP を使用する。

変換 **(transformation).** .jsp ファイル (Web プロジェクトの場合) または .java ファイル (リッチ・クライアント・プロジェクトの場合) に保管される変換。ホスト・コンポーネントを、GUI のウィジェットを使用して抽出および表示する方法を定義する。

変換接続 **(transformation connection).** 「デフォルト接続 **(default connection)**」を参照。

変換フラグメント **(transformation fragment).** 任意の変換において特定のパターンの出現個所をすべて置換するためのコンテンツを含む HATS リソース。

Unicode. 現代世界のいずれかの言語で書き記されたテキストの交換、処理、および表示をサポートする、汎用文字エンコード標準。多くの古典言語や、いくつかの言語の過去のテキストもサポートする。 Unicode 標準には、ISO 10646 で定義された 16 ビットの国際文字セットがある。

ユーザー **(user).** コンピューター・システムのサービスを使用する、人物、組織、プロセス、デバイス、プログラム、プロトコル、またはシステムのこと。

ユーザー・リスト **(user list).** HATS アプリケーションがホストまたはデータベースにアクセスするために使用できる、アカウント (ユーザー ID) に関する情報を記載したリスト。ユーザー・リストには、アカウントのユーザー ID、パスワード、および説明が記載されている。

UTF-8. Unicode 変換形式の 8 ビットのエンコード形式。既存の ASCII ベース・システムで使いやすくするために設計された。

Web アーカイブ **(WAR) (Web archive (WAR)).** Java EE 標準で定義された圧縮ファイル・フォーマット。Web アプリケーションのインストールおよび実行に必要なすべてのリソースを単一のファイルに保管する。

Web 高速ログオン **(WEL) (Web Express Logon (WEL)).** HATS の機能の 1 つであり、これによりユーザーは、ネットワーク・セキュリティー層で認証された一連の信任状を使用して複数のホストにログオンできる。 (「ネットワーク・セキュリティー層 **(network security layer)**」を参照。)

Web サービス **(Web service).** 標準ネットワーク・プロトコルを使用して、ネットワークを介して公開および呼び出すことができる、必要なものを完備した自己記述型のモジュラー・アプリケーション。

Web サービス、RESTful (Web service, RESTful). ステートレス・アーキテクチャーを使用し、関数呼び出しではなくリソースとして表示される Web サービス。Web サービス・リソースを識別するのに適切な書式の URI が使用され、アクティビティーを作成、取得、更新、および削除 (CRUD) するのに HTTP メソッド・プロトコルが使用され、メッセージ・フォーマットを定義するのに HTTP ヘッダー情報が使用される。

Web サービス、従来、WSDL ベース (Web service, traditional, WSDL-based). 通常、データにタグを付けるのに XML が使用され、データの転送に SOAP が使用され、使用可能なサービスの記述に WSDL が使用され、どのサービスが使用可能かリストするのに UDDI が使用される Web サービス。

WebFacing フィーチャー (WebFacing feature). HATS Toolkit の IBM i 用 IBM WebFacing ツールのフィーチャー。 WebFacing フィーチャーを使用すると、IBM i データ記述仕様 (DDS) ディスプレイ・ファイル・ソース・メンバーを、既存の 5250 プログラム用 Web ベース・ユーザー・インターフェースに変換できる。

WebFacing アプリケーション (WebFaced application). HATS Toolkit の WebFacing フィーチャーによって生成される Web アプリケーション。

WebSphere. Web アプリケーションを実行するための e-business アプリケーションおよびミドルウェアを開発するためのツールを包含する IBM の商標名。 WebSphere Application Server の短縮名として使用される場合もある。これは、WebSphere プロダクト・ファミリーのランタイム部分を表す。

WebSphere Application Server. Web サーバー上で実行し、e-ビジネス・アプリケーションの展開、統合、実行、および管理に使用できる、Web アプリケーション・サーバー・ソフトウェア。 HATS アプリケーションは、エクスポートされてサーバーに転送されると、WebSphere Application Server のアプリケーションとして実行される。

WEL. 「**Web 高速ログオン (Web Express Logon (WEL))**」を参照。

ウィジェット **(widget).** 再使用可能なユーザー・インターフェース・コンポーネント。例えば、ボタン、スクロール・バー、制御域、またはテキスト編集域などで、キーボードまたはマウスから入力を受け取り、アプリケーションまたは他のウィジェットと通信できる。 HATS アプリケーションは、ホスト・コンポーネントをウィジェットに変換する。

ウィザード **(wizard).** アクティブな形式のヘルプであり、ユーザーに特定のタスクの各ステップを段階ごとに手引きする。

ワークベンチ **(workbench).** Rational SDP などの Eclipse ベースの製品におけるユーザー・インターフェースおよび統合開発環境 (IDE)。

XML. 「**Extensible Markup Language (XML)**」を参照。

Java に関する各種の定義は、Oracle からの許可を得て転載しました。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ラ行]

A

B

C

D

E

F

P

R

S

T

U

V

W

X

［特殊文字］

Printed in Japan

SA88-5384-02

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21