TogoDocClient プラグイン開発マニュアル

目次

1. この文書の目的

本文書は、「TogoDocClient」のソフトウェアおよびプラグインを開発する手順について説明するものである。

2. TogoDocClient プラグインの開発

2.1. 開発環境の構築

2.1.1. JDK 5.0 の導入

下記の URL から Java SE Development Kit (JDK) 5.0 をダウンロードし、コンピューター上にインストールする。

- http://java.sun.com/j2se/1.5.0/ja/download.html

2.1.2. Eclipse 開発環境の導入

下記の URL から Eclipse Classic または Eclipse RCP and RAP Developers をダウンロードし、コンピューター上に展開する。

- http://www.eclipse.org/downloads/

なお、以後の説明では Eclipse Helios (3.6) Packages の Eclipse Classic を利用するものとして説明を行う。

2.1.3. ターゲットプラットフォームの導入

下記の URL から Eclipse SDK をダウンロードする。これは、TogoDocClient をビルドする際に必要なファイルが含まれている。

- http://archive.eclipse.org/eclipse/downloads/drops/R-3.4.1-200809111700/

ただし、現在の開発に利用しているオペレーティングシステムが Windows XP, Vista, 7 のいずれかであれば Platform に Windows と記載されたものを、Mac OS X 10.5 または Mac

OS X 10.6 のいずれかであれば Platform に Mac OSX (Mac/Carbon)と記載されたものをダウンロードする。ダウンロードしたらアーカイブを任意のパスに展開する。以後、同パスを${TDC_ECLIPSE}と表記する。

次に、プラグインの開発対象としたい TogoDocClient をダウンロードする。ただし、現在の開発に利用しているオペレーティングシステムが Windows XP, Vista, 7 のいずれかであれば Windows 版を、Mac OS X 10.5 または Mac OS X 10.6 のいずれかであれば Mac OSX (Mac/Carbon)版をダウンロードすること。ダウンロードしたらアーカイブを任意のパスに展開する。以後、同パスを${TDC_TARGET}と表記する。

2.1.4. ターゲット環境の指定

2.1.2 で展開した Eclipse を起動し、Preferences ダイアログを開く。同ダイアログの左部から「Plug-in Development > Target Platform」を選択し、右部の「Add...」ボタンを押下する。

「New Target Definition」ダイアログが開かれたら、「Nothing: ...」ラジオボタンが選択された状態で、「Next >」を押下する。次の画面では、「Locations」タブの「Add...」ボタンを押下し、「Add Content」ダイアログの「Installation」を選択して「Next」ボタンを押下する。ここでは「Location:」に${TDC_ECLIPSE}のパスを指定し、「Finish」ボタンを押下する。

続けて、「Locations」タブの「Add...」ボタンを押下し、「Add Content」ダイアログの「Installation」を選択して「Next」ボタンを押下する。今回は「Location:」に${TDC_TARGET}のパスを指定し、「Finish」ボタンを押下する。

「Locations」タブに${TDC_ECLIPSE}と${TDC_TARGET}が追加されたら、同ページの「Environment」を開き、以下の情報を入力する。

- Windows の場合
  - Operating System: win32
  - Windowing System: win32
- Mac OS X の場合
  - Operating System: macosx
  - Windowing System: carbon

- Windows, Mac OS X 共通
  - Architecture: x86
  - Execution Environment: J2SE-1.5

以上を入力し終えたら、上部の「Name:」テキストボックスに「TogoDocClient Plug-in」(任意)を入力し、「Finish」ボタンを押下する。Preferences ダイアログに戻るので、「TogoDocClient Plug-in」にチェックをいれて「OK」ボタンを押下する。

2.2. プラグイン開発の準備

ここでは、TogoDocClient を拡張する手順に共通する、いくつかの手順を解説する。

2.2.1. プラグインプロジェクトの作成

TogoDocClient は Eclipse のプラグイン機構を利用して拡張する。このプラグインを開発するためのプロジェクトは、メニューバーから「File > New > Project...」を選択し、表示されたダイアログの「Plug-in Development > Plug-in Project」を選択して作成する。

New Plug-in Project ウィザードの最初のページでは、次の情報を入力する。

- Project Name
  - プロジェクトの名前

入力が終わったら「Next >」ボタンを押下し、次のページに下記の情報を入力する。

- ID:
  - プロジェクトの名前と同様
- Version:
  - プラグインのバージョン
- Name:
  - プラグインの表記名
- Provider:

- プラグイン作成者または団体の名前

- Execution Environment:
  - J2SE-1.5
- Rich Client Application
  - No

その他の項目は、必要に応じて変更すること。以上の入力が終わったら、「Finish」ボタンを押下してプロジェクトを作成する。

2.2.2. 依存関係の設定

次に、プラグインが TogoDocClient のライブラリを利用できるように、依存関係を設定する。作成したプロジェクトの META-INF フォルダから MANIFST.MF ファイルを開く。

MANIFEST.MF ファイルがエディタで開かれたら、下部「Dependencies」タブを開き、「Required Plug-ins」セクションの Add ボタンを押下する。開かれたダイアログの上部テキストボックスにそれぞれ下記の情報を入力し、「OK」ボタンを押下して依存関係を設定する。

- org.eclipse.core.expressions
- org.eclipse.core.resources
- org.eclipse.core.runtime
- org.eclipse.ui
- org.eclipse.ui.forms
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.core
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui

なお、上記は一例であり、設定によっては既に含まれているものや、実際には利用しないものなどを含む可能性がある。また、jp.ac.u_tokyo.k.cb.wiwasaki.tdc.core や jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui は TogoDocClient の中核を担うライブラリである。

2.3. Eclipse Commands を利用した拡張

TogoDocClient の拡張の多くは、Eclipse のコマンドフレームワークを利用して行うことができる。コマンドはメニューやツールバーに追加することができ、コマンドに対応するハンドラを作成することで、メニュー項目が選択された際に任意のプログラムを実行することができる。

2.3.1. コマンドの作成

コマンドはプラグインの拡張として作成する。プロジェクトの META-INF フォルダから MANIFST.MF ファイルを開き、さらに下部の「Extensions」タブを開く。「All Extensions」セクションに「org.eclipse.ui.commands」という項目が存在しない場合、「Add」ボタンを押下して org.eclipse.ui.commands を追加する。

次に、「All Extensions」セクションの「org.eclipse.ui.commands」項目を選択し、コンテキストメニューから「New > command」を選択する。自動的に子要素が作成されたら、それを選択した状態で「Extension Element Details」セクションを下記のように編集する。

- id
  - コマンドの識別子。仮に「com.example.tdc.exampleCommand」とする
- name
  - コマンドの名前。仮に「サンプルコマンド」とする

以上で、「サンプルコマンド」という名前のコマンドが作成された。

2.3.2. ハンドラの作成

ハンドラはコマンドが何らかの方法(メニューから選択されたなど)で指示された際に、それを受け取ってプログラムを起動するための仕組みである。

ハンドラに対応するプログラムは通常の Java のクラスとして作成し、基底クラスである org.eclipse.core.commands.AbstractHandler を継承させる。以下はメッセージダイアログを表示するだけの簡単なハンドラクラスである。

```
package com.example.tdc;


import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.handlers.HandlerUtil;


public class ExampleHandler extends AbstractHandler {


	public Object execute(ExecutionEvent event) throws ExecutionException {
		Shell shell = HandlerUtil.getActiveShellChecked(event);
		MessageDialog.openInformation(shell,
				"ハンドラのサンプル",
				"これはハンドラのサンプルです");
		return null;
	}
}
```

次に、ハンドラもコマンドと同様に、プラグインの拡張として登録する。プロジェクトの META-INF フォルダから MANIFST.MF ファイルを開き、さらに下部の「Extensions」タブを開く。「All Extensions」セクションに「org.eclipse.ui.handlers」という項目が存在しない場合、「Add」ボタンを押下して org.eclipse.ui. handlers を追加する。

次に、「All Extensions」セクションの「org.eclipse.ui.handlers」項目を選択し、コンテキストメニューから「New > handler」を選択する。自動的に子要素が作成されたら、そ

れを選択した状態で「Extension Element Details」セクションを下記のように編集する。

- commandId
  - 対象コマンドの識別子。2.3.1 で作成したコマンドの識別子を指定する
- class
  - ハンドラプログラム。作成したハンドラクラスの名前を指定する

以上で「サンプルコマンド」に対するハンドラを登録した。以後、「サンプルコマンド」が指示された際にはメッセージダイアログが表示されることになる。

2.3.3. メニュー項目の作成

最後に、作成したコマンドをメニューに登録する。プロジェクトの META-INF フォルダから MANIFST.MF ファイルを開き、さらに下部の「Extensions」タブを開く。「All Extensions」セクションに「org.eclipse.ui.menus」という項目が存在しない場合、「Add」ボタンを押下して org.eclipse.ui. menus を追加する。

次に、「All Extensions」セクションの「org.eclipse.ui.menus」項目を選択し、コンテキストメニューから「New > menuContribution」を選択する。自動的に子要素が作成されたら、それを選択した状態で「Extension Element Details」セクションを下記のように編集する。

- locationURI
  - メニュー識別子。仮に「menu:tdc.ui.menu.help?after=additions」と入力する

この要素は、コマンドを追加する先のメニューを指定する。上記の例では、ヘルプメニューの末尾に追加される。詳しくは TODO を参照すること。

次に、実際のコマンドを指定する。先ほど作成した要素を選択し、コンテキストメニューから「New > command」を指定する。自動的に子要素が作成されたら、それを選択した状態で「Extension Element Details」セクションを下記のように編集する。

- commandId
  - 対象のコマンド識別子。2.3.1 で作成したコマンドの識別子を指定する

その他、メニュー項目として表示される際のラベルを書き換えたり(通常はコマンドと同じ名称を利用する)、アイコンを指定したりすることもできる。

以上の作業が完了したら、現在の作業を保存する。

2.3.4. プラグインの作成

最後に、プラグインを配布可能な形式に変換する。プロジェクトの META-INF フォルダから MANIFST.MF ファイルを開き、さらに下部の「Overview」タブを開く。同タブの「Exporting」セクションから「Export Wizard」という名前のリンクを選択する。

ダイアログが表示されたら、「Available Plug-ins...」に作成したプラグインプロジェクトが選択されていることを確認した上で、「Destination」タブの「Directory」に出力先のパスを指定する。入力が終わったら「Finish」ボタンを押下すると、指定したパスに「plugins」という名前のフォルダが作成され、その中に「プラグイン ID_バージョン番号.jar」という名前のファイルが作成される。

作成したプラグインを組み込むには、上記のファイルを TogoDocClient の直下にある「plugins」ディレクトリにコピーしたのちに、TogoDocClient を起動すればよい。

2.4. TogoDocClient の拡張

ここでは、現在 TogoDocClient で利用可能な拡張を紹介する。

2.4.1. メニューバー項目の追加

メニューバー上の各メニューに項目を追加するには、2.3 で紹介したコマンドフレームワークを利用する。追加先のメニューは「org.eclipse.ui.menus」の「menuContribution」で指定することができる。

以下、同項目に指定可能な locationURI の一覧である。

対象のメニュー	locationURI
ファイル	menu:tdc.ui.menu.file?after=additions
TogoDoc	menu:tdc.ui.menu.dbcls?after=additions

対象のメニュー	locationURI
検索	menu:tdc.ui.menu.search?after=additions
統計情報	menu:tdc.ui.menu.statistics?after=additions
ウィンドウ	menu:tdc.ui.menu.window?after=additions
ヘルプ	menu:tdc.ui.menu.help?after=additions

なお、Mac OS X 固有のアプリケーションメニューに項目を追加することはできない。

2.4.2. ツールバーやアクションバーボタンの追加

ツールバーやアクションバーにボタンを追加する場合も、2.3 で紹介したコマンドフレームワークを利用する。追加先は「org.eclipse.ui.menus」の「menuContribution」で指定することができる。

以下、同項目に指定可能な locationURI の一覧である。

対象のメニュー	locationURI
ツールバー	toolbar:tdc.ui.toolbar?after=additions
Article Explorer	toolbar:tdc.ui.articleexplorer?after=additions
Tag Explorer	toolbar:tdc.ui.tagexplorer?after=additions
Article Search	toolbar:tdc.ui.articlesearch?after=additions

2.4.3. コンテキストメニュー項目の追加

各種コンテキストメニューに項目を追加する場合も、2.3 で紹介したコマンドフレームワークを利用する。追加先のメニューは「org.eclipse.ui.menus」の「menuContribution」

で指定することができる。

以下、同項目に指定可能な locationURI の一覧である。

対象のメニュー	locationURI
Article Explorer	popup:tdc.ui.articleexplorer?after=additions
Tag Explorer	popup:tdc.ui.tagexplorer?after=additions
タグ作成ダイアログ	popup:tdc.ui.articleeditor.tagdialog?after=additions
Article Search	popup:tdc.ui.articlesearch?after=additions
List View	popup:tdc.ui.articleplaylist?after=additions

2.4.4. 文献タブのセクションの追加

文献タブにセクションを追加する際には、追加するセクションを作成・更新するクラスと、そのクラスのインスタンスを生成する Factory クラスの 2 つを作成し、Factory クラスの名前を拡張ポイントの creatorFactory 要素の class に設定する。

セクションの追加をする plugin.xml を記述するために必要な情報は下記の通りである。

項目	内容
extension point	jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui. articleSectionCreatorFactory
creatorFactory	jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.control. IArticleSectionCreatorFactory を実装したクラス

2.4.4.1. Factory クラス

拡張の基盤となるのは実際のセクションを作成するクラスではなく、そのクラスを生成するための Factory クラスであり、下記のインターフェースを実装している必要がある。

jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.control.IArticleSectionCreatorFactory

Factory クラスのインスタンスは TogoDocClient 内でひとつのみ生成される。複数の Factory クラスを拡張として登録した場合、拡張ごとにひとつずつインスタンスが生成される。

新しい Article Tab を開いた時に createSection メソッドが呼ばれ、セクション生成クラスのインスタンスが生成される。

2.4.4.2. セクション生成クラス

文献タブ上にセクションを生成し、そのセクションに文献情報を表示する処理を行うクラス。新しい Article Tab を開いた時にインスタンスが生成され、Article Tab を閉じる時にそのインスタンスが解放される。セクション生成クラスは下記の interface を実装している必要がある。

jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.control.IArticleSectionCreator

インスタンス生成時には表示対象文献がない状態であり、setArticle メソッドが呼ばれたときにパラメータとして渡された文献情報の内容を表示するよう実装する。setArticle メソッドが呼ばれるのは次のタイミングである。

- 文献タブを開いた直後
- 表示対象の文献を変更した時
- 文献タブがフォーカスされていない状態からフォーカスされた時

文献情報の項目には必ずしも値が入っているとは限らないため、値が入っていない場合についても考慮して実装する必要がある。

2.4.5. Article Search および List View のカラムの追加

Article Search および List View のテーブルにカラムを追加する際には、追加するカラムを定義するクラスと、そのクラスのインスタンスを生成する Factory クラスの 2 つを作成し、Factory クラスの名前を拡張ポイントの columnFactory 要素の class に設定する。

カラムの追加をする plugin.xml を記述するために必要な情報は下記の通り。

項目	内容
extension point	jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui. articleTableColumnFactory
columnFactory	jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.control. IArticleTableColumnFieldFactory を実装したクラス

2.4.5.1. Factory クラス

拡張の基盤となるのはカラムを定義するクラスではなく、そのクラスを生成するための Factory クラスであり、下記の interface を実装している必要がある。

jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.control.IArticleTableColumnFieldFactory

Factory クラスのインスタンスは TogoDocClient 内でひとつのみ生成され、すぐに createColumnFields メソッドが呼ばれたあと、Factory クラスのインスタンスは参照されなくなる。複数の Factory クラスを拡張として登録した場合、TogoDocClient が Factory クラスを認識した順番に createColumnFields メソッドを呼び、戻り値のリストをひとつにまとめて利用する。

2.4.5.2. カラム定義クラス

Article Search および List View のテーブルビューに追加するカラムの定義をするクラス。定義の内容としては下記のものがある。

- カラム名 (表示上)
- カラム幅の初期値
- カラムの内容
- カラムの値の比較 (並べ替えで利用)

カラム生成クラスのインスタンスはテーブルごとに生成される。Article Search のテーブルビューと List View のテーブルビューとで、合計二回ずつ生成される形となる。カラム生成クラスは下記の interface を実装している必要がある。

jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.control.IArticleTableColumnField

文献情報の項目には必ずしも値が入っているとは限らないため、値が入っていない場合についても考慮して実装する必要がある。

3. TogoDocClient の開発

3.1. 開発環境の構築

3.1.1. JDK 5.0 の導入

下記の URL から Java SE Development Kit (JDK) 5.0 をダウンロードし、コンピューター上にインストールする。

- http://java.sun.com/j2se/1.5.0/ja/download.html

3.1.2. Eclipse 開発環境の導入

下記の URL から Eclipse Classic をダウンロードし、コンピューター上に展開する。この展開先を以後、${ECLIPSE}と表記する。

- http://www.eclipse.org/downloads/

なお、以後の説明では Eclipse Helios (3.6) Packages の Eclipse Classic を利用するものとして説明を行う。また、Eclipse 起動時のワークスペースへのパスは${WORKSPACE}と表記するものとする。

3.1.3. SVN クライアントの導入

3.1.2 で展開した Eclipse を起動し、メニューバーから「Help > Install New Software...」を選択する。Install ダイアログが表示されたら、「Work with:」ドロップダウンリストから「Helios - ...」を選択する。

しばらくするとインストール可能なソフトウェアの一覧が表示されるので、「Collaboration」の項目から以下の 2 つを選択する。

- Subversive SVN Team Provider
- Subversive SVN JDT Ignore Extensions

以上 2 点を選択したら、「Next >」ボタンを押下し、ウィザードの流れに従ってソフトウェアをインストールする。インストールが完了すると再起動を要求されるので、説明に従って Eclipse を再起動する。

再起動が完了したら、メニューバーから「Window > Open Perspective > Other...」を選択し、さらに「SVN Repository Exploring」を選択する。

ここで、「Install Connectors」というダイアログがポップアップするので、「SVN Kit」から始まるものだけを選択して「Finish」ボタンを押下する。以後、ウィザードの流れに従ってソフトウェアをインストール後、指示に従って Eclipse を再起動する。

再起動が完了したら、以下の手順で Preferences ダイアログを表示する。

- Windows
  - ➢ メニューバーから「Window > Preferences」を選択する
- Mac OS X
  - ➢ メニューバーから「Eclipse > Preferences」を選択する

Preferences ダイアログの左部から「Team > SVN」を選択し、右側の「SVN Connector」タブを開く。「SVN Connector:」ドロップダウンボックスから最新のバージョンを選択し、「OK」ボタンを押下する。

3.1.4. ソースコードのチェックアウト

メニューバーから「Window > Open Perspective > Other...」を選択し、さらに「SVN Repository Exploring」を選択する。左部の「SVN Repositories」ビューのアクションバー(タブの右または下)から「New Repository Location」ボタンを押下し、ダイアログの「General」タブに次の情報を入力する。

- URL: http://svn.sourceforge.jp/svnroot/togodoc/client/tags/[Version Number]
- User: (SVN のユーザー名)

- Password: (SVN のパスワード)

リポジトリが追加されたら、下記のフォルダを選択し、コンテキストメニューから「Check Out」を選択する。

- jp.ac.u_tokyo.k.cb.wiwasaki.tdc
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc-feature
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc-site
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.analyzer.jpedal
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.analyzer.jpedal-feature
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.analyzer.jpedal.nl_ja
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.base
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.base-feature
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.build
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.core
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.core.nl_ja
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.core.test
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.carbon
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.nl_ja
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.test
- tdc-documents
- tdc-releases

ただし、jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui.carbon は Windows 環境ではチェックアウトしないこと。これは Mac OS X の環境でのみビルド可能なプロジェクトである。

チェックアウトが完了した状態では、いくつかのプロジェクトがエラーとして表示されているはずである。

3.1.5. ターゲットプラットフォームの導入

下記の URL から Eclipse SDK をダウンロードする。これは、TogoDocClient をビルドする際に必要なファイルが含まれている。

- http://archive.eclipse.org/eclipse/downloads/drops/R-3.4.1-200809111700/

ただし、現在の開発に利用しているオペレーティングシステムが Windows XP, Vista, 7 のいずれかであれば Platform に Windows と記載されたものを、Mac OS X 10.5 または Mac OS X 10.6 のいずれかであれば Platform に Mac OSX (Mac/Carbon)と記載されたものをダウンロードする。ダウンロードしたらアーカイブを任意のパスに展開する。以後、同パスを${TARGET }と表記する。

次に、同ページから Delta Pack をダウンロードする。これは、TogoDocClient を様々なプラットフォーム向けにビルドするために必要である。ダウンロードが完了したらアーカイブを展開し、それに含まれる plugins フォルダ以下に含まれる、全ての.jar ファイルを${TARGET}/plugins 以下にコピーする。

最後に、先ほど入手した${WORKSPACE}/tdc-releases/rcp.nl_ja 以下に含まれる、全ての.jar ファイル${TARGET}/plugins 以下にコピーする。

3.1.6. ターゲット環境の指定

3.1.2 で展開した Eclipse を起動し、Preferences ダイアログを開く。同ダイアログの左部から「Plug-in Development > Target Platform」を選択し、右部の「Add...」ボタンを押下する。

「New Target Definition」ダイアログが開かれたら、「Nothing: ...」ラジオボタンが選択された状態で、「Next >」を押下する。次の画面では、「Locations」タブの「Add...」ボタンを押下し、「Add Content」ダイアログの「Installation」を選択して「Next」ボタンを押下する。ここでは「Location:」に${TARGET}のパスを指定し、「Finish」ボタンを押下する。

「Locations」タブに${TARGET}が追加されたら、同ページの「Environment」を開き、以下の情報を入力する。

- Windows の場合
  - Operating System: win32

- Windowing System: win32

- Mac OS X の場合
  - Operating System: macosx
  - Windowing System: carbon

- Windows, Mac OS X 共通
  - Architecture: x86
  - Execution Environment: J2SE-1.5

以上を入力し終えたら、上部の「Name:」テキストボックスに「TogoDocClient」(任意)を入力し、「Finish」ボタンを押下する。Preferences ダイアログに戻るので、「TogoDocClient」にチェックをいれて「OK」ボタンを押下する。

上記の指定が終わると、全てのプロジェクトのクリーンビルドが行われる。

3.2. TogoDocClient のビルド

3.1.2 で展開した Eclipse を起動し、jp.ac.u_tokyo.k.cb.wiwasaki.tdc.build プロジェクトの scripts フォルダを展開する。同フォルダに含まれる build.sample.properties というファイルを同じディレクトリに「build.properties」という名前でコピーする。

コピーしたファイルを開くと、ビルドのための設定が「名前 = 設定する値」の形式で列挙してある。以下のペアを環境に合わせて修正する。

名前	設定する値
version	ビルドするバージョン
buildDirectory	ビルドの作業ディレクトリ
base	${TARGET}の親ディレクトリ
baseLocation	${TARGET}
baseos	win32 または macosx

名前	設定する値
basews	win32 または carbon

以上の設定が終わったら内容をファイルに保存し、ファイルのコンテキストメニューから「Run As > Ant Build」を選択する。Ant によるビルドスクリプトの実行が完了したら、tdc-releases プロジェクトをリフレッシュすると release フォルダに指定したバージョンのバイナリが生成されているはずである。

同様に、jp.ac.u_tokyo.k.cb.wiwasaki.tdc-site プロジェクトをリフレッシュすると、features と plugins というディレクトリに更新サイト用のファイルが生成されているはずである。更新サイトを構築する場合、site.xml と併せて features, plugins フォルダをオンラインに配置すればよい。

3.3. TogoDocClient のデバッグ

3.3.1. 開発環境から TogoDocClient を起動する

jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui プロジェクト直下の tdc.product ファイルを選択し、コンテキストメニューから「Debug As > Eclipse Application」を選択する。すると、TogoDocClient が開発環境の管理下で起動する。

さらに詳細なログが必要である場合、メニューバーの「Run > Debug Configurations...」を選択する。ダイアログが開いたら、左部の「Eclipse Application > tdc.product」を選択し、Tracing タブの内容を次のように変更する。

- Enable tracing にチェック
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.core を選択し、debug にチェック
- jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui を選択し、debug にチェック

以上の設定を終えたら、Debug ボタンを押下すると TogoDocClient が起動する。

3.3.2. TogoDocClient にデバッガでアタッチする

開発環境の外で起動した TogoDocClient にデバッガでアタッチする場合、設定ファイル

にデバッガ用の指定を行った後に TogoDocClient を起動する必要がある。TogoDocClient の設定ファイルは次の場所に配置される。

- Windows の場合
  - <インストール先>/TogoDocClient.ini
- Mac OS X の場合
  - <インストール先>/TogoDocClient.app/Contents/MacOS/TogoDocClient.ini

ファイルを開き、ファイルの末尾 (または-vmargs の行よりも後) に次の行を追加する。

```
-Xdebug
-Xnoagent
-Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=8000
```

上記の設定が完了したらファイルを保存し、TogoDocClient を起動する。この設定では、TCP のポート 8000 番でデバッガのアタッチを受け付け、アタッチされるまでプログラムを起動しない。

開発環境側では、上記のプロセスにアタッチしてデバッグを行う。メニューバーの「Run As > Debug Configurations...」を開き、左部から「Remote Java Application」を選択した状態で、コンテキストメニューの「New」を選択する。生成された画面では、次のような設定を行う。

- Name: (自由)
- Project: jp.ac.u_tokyo.k.cb.wiwasaki.tdc.ui
- Host: localhost
- Port: 8000

以上の設定で「Debug」ボタンを押下すると、デバッガのアタッチを待機していた TogoDocClient の起動が始まる。

3.3.3. 詳細なログを生成する

TogoDocClient の利用時に詳細なログを生成するようにするには、3.3.2 に記載するファイルを開き、ファイルの先頭 (または-vmargs の行よりも前) に次の行を追加する。

```
-debug
```

設定ファイルを保存後に TogoDocClient を起動すると、裏側では様々なログが生成される。このログは、それぞれ以下のディレクトリ上に生成される。

- Windows
  - <インストール先>/workspace/.metadata
- Mac OS X
  - <インストール先>/TogoDocClient.app/Contents/MacOS/workspace/.metadata

3.3.4. 更新サイトを指定する

TogoDocClient の更新サイトを構築する前にソフトウェアアップデートのテストを行うには、3.3.2 に記載するファイルを開き、ファイルの末尾 (または-vmargs の行よりも後) に次の行を追加する。

```
-Dtdc.update.site=<更新サイトの URL>
```

更新サイトの URL は"file:"から始まるものでもかまわない。たとえばローカルの jp.ac.u_tokyo.k.cb.wiwasaki.tdc-site プロジェクトへのパスを指定すれば、最終ビルド結果が常にアップデートサイトから利用できるようになる。

なお、更新サイトのテストは 3.3.1 のように開発環境から TogoDocClient を起動した場合にうまくいかない可能性がある。

3.3.5. 英語メッセージリソースを表示する

TogoDocClient を英語版として起動する場合、3.3.2 に記載するファイルを開き、ファイルの先頭 (または-vmargs の行よりも前) に次の行を追加する。

```
-nl
en
```

TogoDocClient はオペレーティングシステムの言語設定を元に自動的に言語を切り替える。日本語以外の環境から日本語版を起動する場合、上記の「en」を「ja」に変えて起動すること。

3.4. TogoDocClient のリリース

3.4.1. バージョン番号の調整

リリースを行うには、全てのプラグイン、フィーチャー、およびプロダクトに関するバージョンの変更をあらかじめ行う必要がある。バージョンが衝突した場合、更新サイトを利用した更新に不具合が起きる可能性がある。

また、更新サイトを構築するため、jp.ac.u_tokyo.k.cb.wiwasaki.tdc-site プロジェクトの site.xml ファイルに、リリースするフィーチャーを加えてやること。

3.4.2. 実行可能パッケージ

実行可能パッケージは、3.2 で生成したアーカイブファイルをそのまま利用する。

3.4.3. 更新サイト

更新サイトは、3.2 で jp.ac.u_tokyo.k.cb.wiwasaki.tdc-site プロジェクトに生成された plugins フォルダの内容、features フォルダの内容、および site.xml ファイルを利用する。

これらの内容を現在の更新サイトに上書きすればよい。また、更新サイトを最初から構築する場合には、site.xml に含まれるフィーチャーのうち、存在しないバージョンに関するエントリを除去しておくこと。

3.4.4. DTD リポジトリ

現在の TogoDocClient はサーバとの通信で XML DTD を利用しているが、いくつかのサ

イトの DTD の配布が正しくない場合が確認されている。このような問題を回避するため、現在は利用する DTD を別のサーバからも提供するようにしている。

- http://togodoc.sourceforge.jp/client/dtd/

新たな DTD が追加された場合、その DTD ファイルの名前を変えずに、上記の URL の直下に配置すること。

3.4.5. SVN の設定

全ての作業が完了したら、リリースタグを作成する。リリースタグは trunk の内容丸ごとで、名前はリリースバージョンとする。