日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

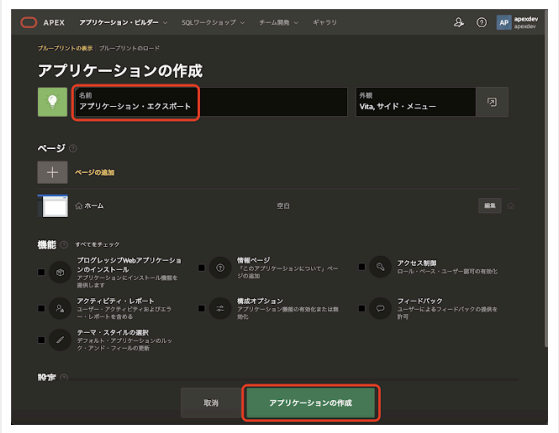
2022年12月5日月曜日

GitHubにAPEXアプリをエクスポートする(3) - アプリケーションの作成

APEXアプリケーションをGitHubにエクスポートするアプリケーションを作成します。

最初に**アプリケーション作成ウィザード**を起動し、空のアプリケーションを作成します。

名前は**アプリケーション・エクスポート**としました。



ページの作成

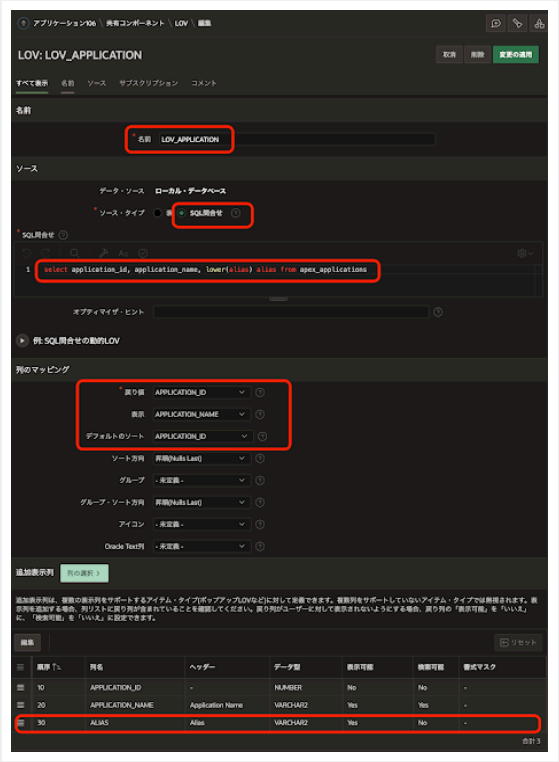
作成されたアプリケーションのホーム・ページに、ページ・アイテムを５つ、ボタンをひとつ作成します。

エクスポート対象のアプリケーションは、ポップアップLOVで選択します。選択したアプリケーションの別名をページ・アイテム**P1_ALIAS**に設定するため、**LOV**は**共有コンポーネント**として作成します。

LOVの**名前**は**LOV_APPLICATION**とします。**ソース**として**SQL問合せ**を選択し、**SQL問合せ**として以下を記述します。**戻り値**は**APPLICATION_ID**、**表示値**は**APPLICATION_NAME**です。

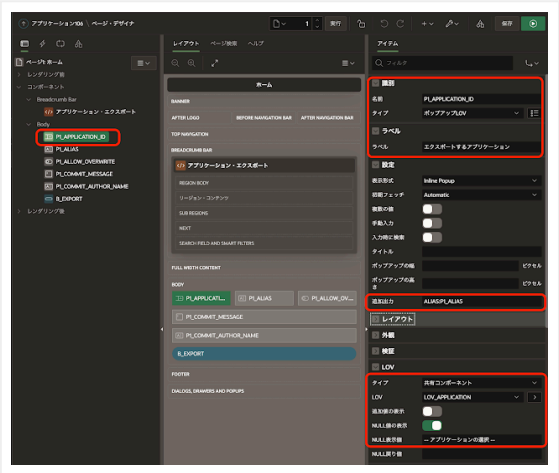
```
select application_id, application_name, lower(alias) alias from apex_applications
```

追加表示列として、列**ALIAS**を追加します。この通りでなくても構いませんが、**表示可能**は**Yes**、**検索可能**は**No**としています。



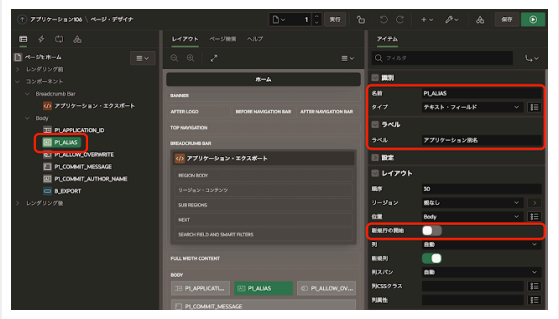
エクスポートするアプリケーションを選択するページ・アイテム**P1_APPLICATION_ID**として、**タイプ**に**ポップアップLOV**、**設定**の**追加出力**として**ALIAS:P1_ALIAS**を指定します。ポップアップLOVで値を選択すると、列**APPLICATION_ID**の値がページ・アイテム**P1_APPLICATION_ID**の値になり、列**ALIAS**の値がページ・アイテム**P1_ALIAS**の値になります。

LOVの**タイプ**として**共有コンポーネント**を選択し、**LOV**に**LOV_APPLICATION**を選びます。**追加値の表示**は**OFF**、**NULL値の表示**は**ON**、**NULL表示値**として**-- アプリケーションの選択 --**を入力します。



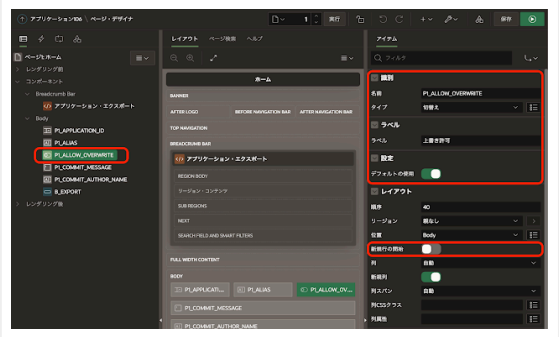
ページ・アイテム**P1_ALIAS**の値は、アプリケーションを選択すると自動的に設定されます。値の設定後に変更ができるよう、**タイプ**は**テキスト・フィールド**にします。ページ・アイテム

P1_APPLICATION_IDの右隣に配置されるよう、**レイアウト**の**新規行の開始**は**OFF**にします。

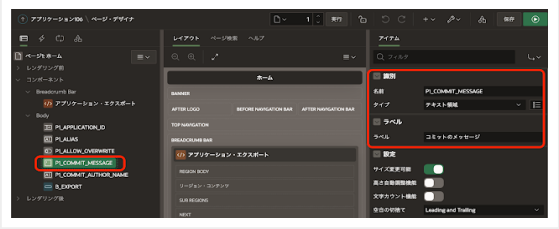


上書き許可のページ・アイテム**P1_ALLOW_OVERWRITE**の**タイプ**に**切替え**（英語は**Switch**）を選択します。このページ・アイテムの値はそのまま、PL/SQLのboolean値として扱うことができます。

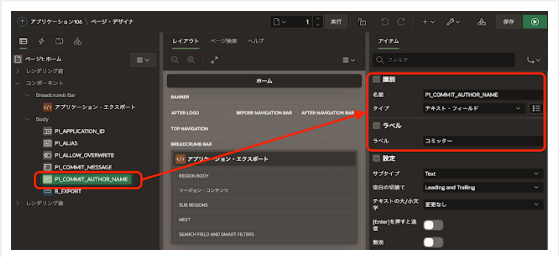
こちらも**レイアウト**の**新規行の開始**は**OFF**にします。



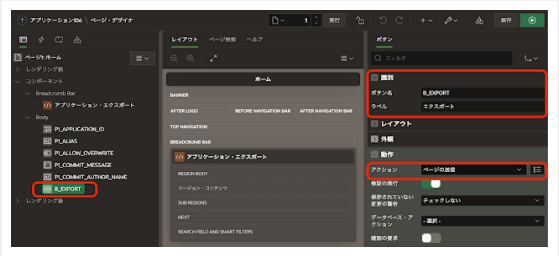
コミットのメッセージを保持するページ・アイテム**P1_COMMIT_MESSAGE**の**タイプ**として**テキスト領域**を選択します。



コミッターを保持するページ・アイテム**P1_COMMIT_AUTHOR_NAME**の**タイプ**は**テキスト・フィールド**です。



最後に送信ボタン**B_EXPORT**を作成します。**動作**の**アクション**は**ページの送信**です。



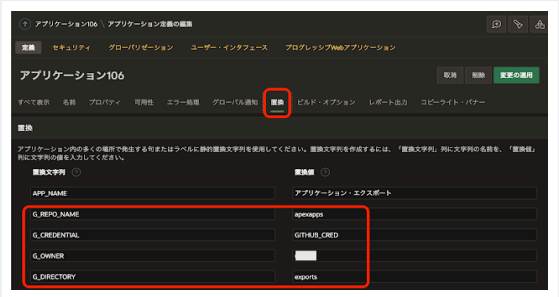
以上でページは完成です。

置換文字列の設定

プロシージャEXPORT_APEX_APP_TO_GITHUBを呼び出す際に、APEX側で決定する固定値を**アプリケーション定義**の**置換文字列**として設定します。

アプリケーション定義を開き、**置換**のセクションを選択します。以下の**置換文字列**を設定します。

G_REPO_NAE = GitHubのリポジトリ名
G_CREDENTIAL = DBMS_CLOUD.CREATE_CREDENTIALで作成したクリデンシャル名 - 今回の例通りに作成していると**GITHUB_CRED**
G_OWNER = GitHubのアカウント名
G_DIRECTORY = リポジトリのディレクトリ名。指定されたディレクトリ以下にAPEXアプリケーションをエクスポートする。



プロセスの作成

左ペインで**プロセス・ビュー**を開き、**プロセスの作成**を実行します。

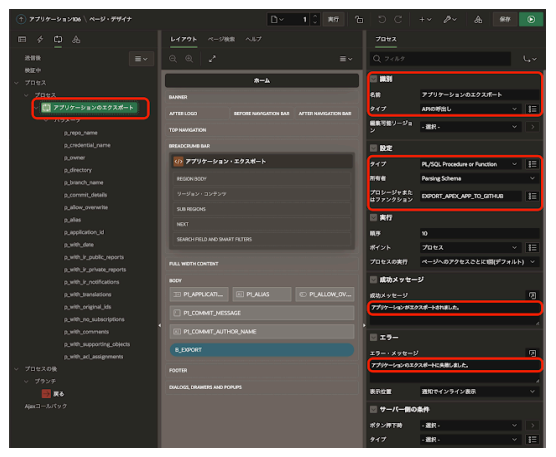
識別の**名前**を**アプリケーションのエクスポート**とします。**タイプ**としてAPEX 22.2より追加された**APIの呼び出し**を選択します。

設定の**タイプ**として**PL/SQL Procedure or Function**を選択します。今回はサンプルなのでプロシージャとして機能を実装していますが、製品レベルでの実装ではパッケージとして実装する（その場合はPL/SQL Packageを選択する）ことをお勧めします。

所有者は**Parsing Schema**、**プロシージャまたはファンクション**として
EXPORT_APEX_APP_TO_GITHUBを選択します。プロシージャが選択されると、そのパラメータが

ノード**パラメータ**に一覧されます。

成功メッセージとして**アプリケーションがエクスポートされました。エラー・メッセージ**として**アプリケーションのエクスポートに失敗しました。**を設定します。エラーメッセージはプログラムからも設定できるので、もっと凝ったメッセージを出力することもできます。



プロシージャのパラメータに与える値を設定します。

p_repo_nameを選択します。**パラメータ**の**方向**、**データ型**、**デフォルトあり**はプロシージャ定義から決まります。APEXでサポートしているデータ型であれば、設定の変更が必要なケースは少ないでしょう。

値の**タイプ**に**アイテム**、**アイテム**として**G_REPO_NAME**を指定します。G_REPO_NAMEは実際は置換文字列なので、アイテムとして選ぶことはできません。直接、キーボードから入力します。

以上で置換文字列**G_REPO_NAME**として設定した値が、引数**p_repo_name**に渡されます。

同様に、パラメータ**p_credential_name**には**G_CREDENTIAL**、**p_owner**には**G_OWNER**、**p_directory**には**G_DIRECTORY**、**p_allow_overwrite**には**P1_ALLOW_OVERWRITE**、**p_application_id**には**P1_APPLICATION_ID**、**p_alias**には**P1_ALIAS**を設定します。

パラメータの**データ型**が**BOOLEAN**（今回はp_allow_overwrite）の際に、真偽値として使用する文字がデフォルトであると、**True値**と**False値**の指定を省略できます。

p_branch_nameを選択します。**値のタイプ**として**APIデフォルト**を選択します。プロシージャの引数に設定されているデフォルト値がp_branch_nameの値になります。

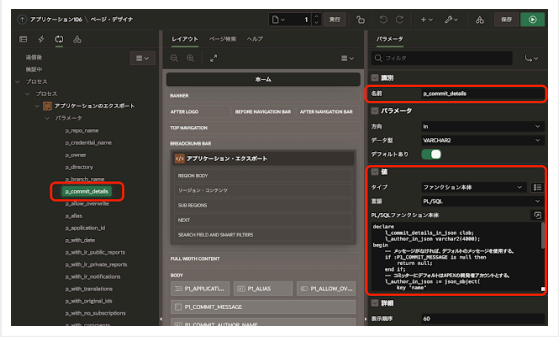
p_commit_detailsを除いて、他のパラメータはすべて**APIデフォルト**を選択します。引数にデフォルト値が定義されていると、値のタイプのデフォルトはAPIデフォルトになります。

p_commit_detailsはDBMS_CLOUD_REPO.PUT_FILE等の引数p_commit_detailsとして与えられる値ですが、これはJSON文字列です。そのためページ・アイテム**P1_COMMIT_MESSAGE**と**P1_COMMIT_AUTHOR_NAME**よりJSON文字列を作成し、その値をp_comitt_detailsに与えます。

p_commit_detailsの**値のタイプ**として**ファンクション本体**を選択し、**言語**は**PL/SQL**を選びます。**PL/SQLファンクション本体**として以下を記述します。

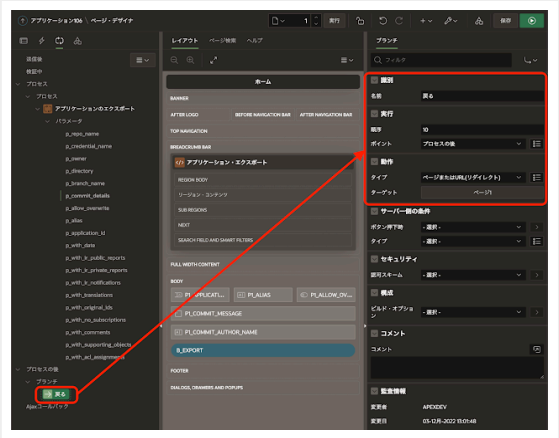
```
declare
    l_commit_details_in_json clob;
    l_author_in_json varchar2(4000);
begin
    -- メッセージがなければ、デフォルトのメッセージを使用する。
    if :P1_COMMIT_MESSAGE is null then
        return null;
    end if;
    -- コミッターにデフォルトはAPEXの開発者アカウントとする。
    l_author_in_json := json_object(
        key 'name'
        value
            case
            when :P1_COMMIT_AUTHOR_NAME is not null then
                :P1_COMMIT_AUTHOR_NAME
            else
                :APP_USER
            end
    );
    l_commit_details_in_json := json_object(
        key 'message' value :P1_COMMIT_MESSAGE,
        key 'author' value l_author_in_json format json
    );
    return l_commit_details_in_json;
end;
```

p_commit_details.sql hosted with ❤️ by GitHub　　view raw



以上でAPI呼び出しを行うプロセスの設定は完了です。

プロセスの実行後、ホーム・ページを再表示するために**ブランチを作成**します。**タイプ**は**ページまたはURL(リダイレクト)**、**ターゲット**として**ページ 1** を選択します。



以上でアプリケーションは完成です。

YAMLのGitHubへの保存が遅いのが気にはなります。とはいえYAMLファイルは非圧縮であるためファイル・サイズが大きい、またファイル数がそれなりにあることが原因で、APEX側で対応できないように思います。

プロシージャEXPORT_APEX_APP_TO_GITHUBの引数としてb_branch_nameがあり、コミットするブランチを指定できます。ただし、今回作成したAPEXアプリケーションでは常にmainを指定しています。

今のところAPEXではYAML形式のファイルをインポートすることはできません（**p_type**の指定は**READABLE_YAML**です）。そのため、ブランチごとの変更をマージするのは、アプリケーション・デザイナーおよびページ・デザイナを使用して手動で行う必要があります。とはいえ、YAMLファイルを比較することでアプリケーションの違いを確認することはできます。アプリケーションがインストールされているインスタンスが異なるという場合など、YAMLファイルで差分を確認できると便利なことも多いでしょう。

今回の記事は以上になります。

Oracle APEXのアプリケーション作成の参考になれば幸いです。

完

Yuji N. 時刻: 18:50

共有

‹ ホーム ›

ウェブ バージョンを表示

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示