Valence

[Edit Grid]ウィジェット活用術

株式会社ミガロ.
プロダクト事業部　技術支援課
尾崎　浩司

略　歴

生年月日:1973年8月16日
最終学歴:1996年　三重大学　工学部卒業
入社年月:1999年10月　株式会社ミガロ. 入社
社内経歴:
1999年10月　システム事業部配属
2013年04月　RAD事業部(現プロダクト事業部)配属

現在の仕事内容:

ミガロ. が取り扱う3つの開発ツールのセミナー講師や技術支援を主に担当している。

1.はじめに

IBM i(AS/400)環境に特化したモダナイゼーションツールであるValenceには、ローコード開発機能である「Nitro App Builder」(以下「App Builder」と記載)が搭載されている。
「App Builder」は、[①データソースの作成]-[②ウィジェットの定義]-[③アプリケーション作成]という3ステップで容易にアプリケーションを作成できるのが特徴である。その中でも、UI(ユーザーインターフェース)の肝となるが、[②ウィジェットの定義]である。Valence6.2には、全16種類のウィジェット(部品)が用意されており、データソースと関連付けてウィジェットを定義する事で、表現力の高いGUI画面が実装できる。

参照/照会系のWebアプリ開発では、一覧形式(Grid)でデータ出力する[Grid]ウィジェットや、クロス集計表を実現する[Pivot Grid]ウィジェットを使用するのが一般的である。
また、登録/更新系のWebアプリ開発では、[Grid]ウィジェットの機能にデータの更新機能を追加した[Edit Grid]ウィジェットを使用する事ができる。
この[Edit Grid]ウィジェットについて、「App Builder」が登場した2018年当時は、シンプルなデータ編集機能しか実装されていなかった。その為、このウィジェットは、簡単なマスタメンテナンスアプリ位でしか利用できないと認識の方もいるだろう。しかし、Valenceの進化と共に[Edit Grid]ウィジェットの機能も大幅に進化しており、現在は多彩な編集処理が可能である。
ただ、これまで[Edit Grid]ウィジェットの機能を体系的に纏めた事が無かった為、今回改めて本稿を執筆する事にした。

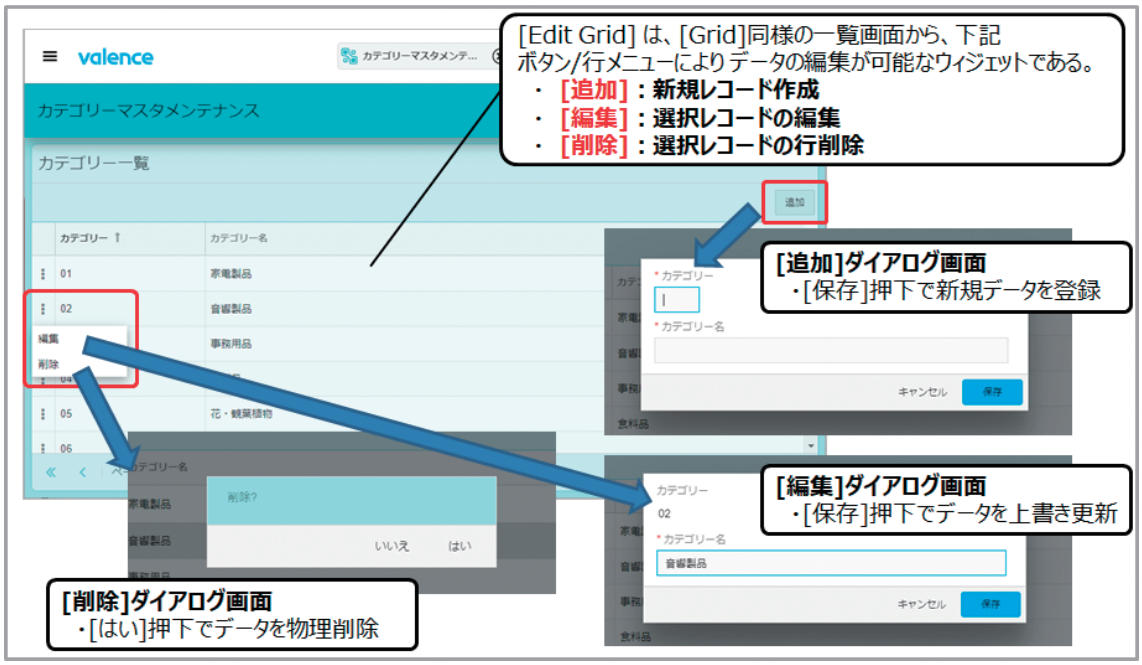
本稿では、登録/更新系アプリ開発には欠かせない[Edit Grid]ウィジェットに焦点を絞り、基本的な使用方法から、応用的な活用術までを具体例を交えて紹介したい。なお、本稿は2023年8月現在の最新版Valence6.2(Build 6.2.20230808.0)を前提とした内容となっている。旧バージョンを利用している場合、無償で製品バージョンアップが可能である為、最新版を導入の上で試してほしい。
なお、本稿で提示している各サンプルアプリは、以下より、全てダウンロードできるようにしている。ぜひサンプルアプリを導入して実際の動きを確認する事をお勧めしたい。

【サンプルアプリダウンロードURL】
https://www.migaro.co.jp/valsample/kozaki2023.zip
※サンプルダウンロードには、Valenceメンテナンスサイトへのログインユーザー・パスワードが必要

2.[Edit Grid]ウィジェットの基本

「App Builder」における[Edit Grid]ウィジェットは、データを一覧形式で表示可能な[Grid]ウィジェットの機能に、選択行のデータに対する編集/削除の機能や新規レコード追加の機能が付加されたデータ編集用ウィジェットである。例えば、**【図1】**は、シンプルにコードと名称のみを持つ『カテゴリーマスタ』を元に[Edit Grid]ウィジェットを使用して作成したカテゴリーマスタメンテナンスアプリの実行画面である。

図 1　カテゴリーマスタメンテナンスアプリ実装例



ウィジェットの外観は、[Grid]と同様だが、画面右上部には[追加]ボタンが、各行の左側には、行メニュー([⋮])が追加されている事が分かる。[追加]をクリックすると、ポップアップで空のダイアログ画面が開き新規レコードを登録できる。また、行メニューから[編集]をクリックすると、選択行の編集用ダイアログ画面からデータの編集が行える。[削除]をクリックすると、確認ダイアログが表示され、[はい]を選択すると、対象レコードを削除できる。これが[Edit Grid]ウィジェットの基本動作である。

このサンプルアプリの[Edit Grid]ウィジェット設計画面を開くと、[Grid]ウィジェット同様の[カラム]/[設定]/[フィルタ]タブに加え、[編集]タブが用意されている事がわかる。**【図2】**

図 2　Edit Grid 設計画面 – [編集]タブ

[編集]タブには、[追加時]及び[編集時]の入力対象フィールドが選択できるようになっている。**【図2】**の場合、[追加時]は、[カテゴリーコード]と[カテゴリー名]の2つが入力項目、[編集時]は、[カテゴリーコード]は読取り専用(表示のみ)とし、[カテゴリー名]のみ入力項目にするという定義となっている。なお、[削除時]の設定は、画面右側にある[設定]欄に含まれる[削除を許可]にチェックを付ければよい。

次に[Edit Grid]ウィジェットを使用した商品マスタメンテナンスアプリを紹介する。ここで使用するのは、**【図3】**のようなファイルレイアウトを持つ商品マスタである

図 3　商品マスタ　ファイルレイアウト

DSPFMT　　レ コ ー ド 設 計 書　　日付 23/08/28　時刻 19:45:21

物理ファイル TECREPLIB/MPRODP　様式名 MPRODR　レコード長 111
様式記述 商品マスタ
5= 詳細

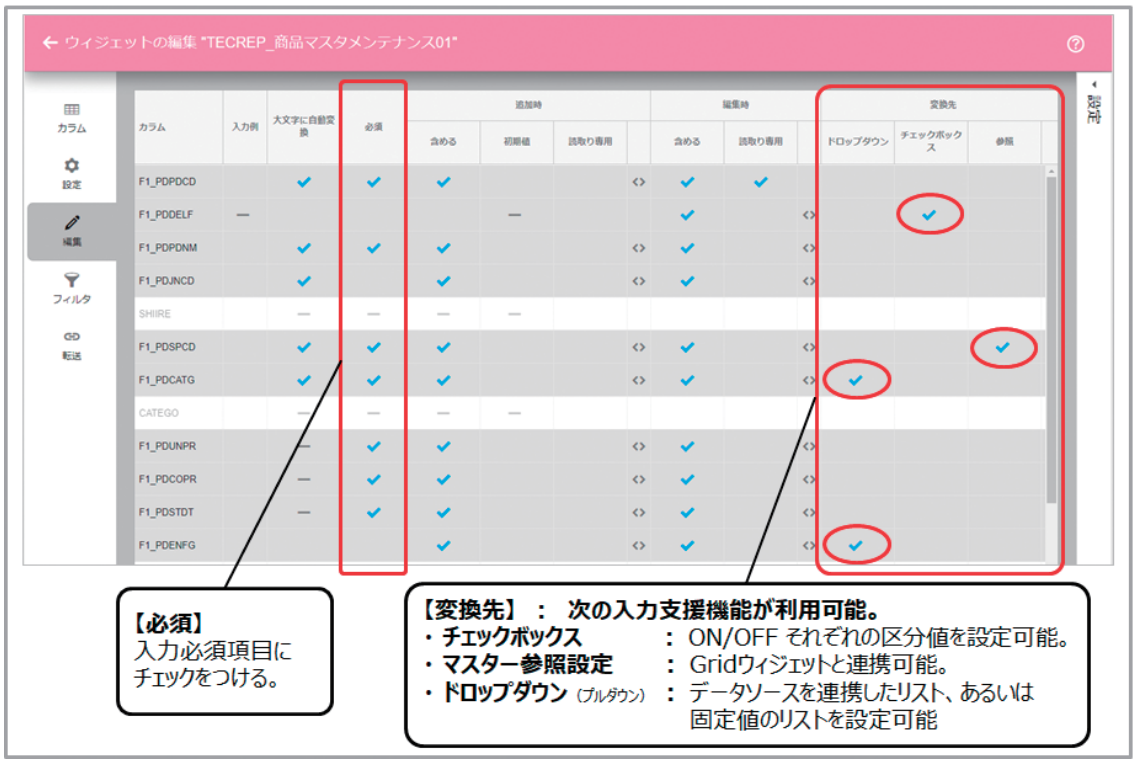
選択	項目名	桁数		属性	キー順	開始	終了	テキスト記述／欄見出し
_	PDDELF	1		A		1	1	削除フラグ
_	PDPDCD	10		A	1 ANN	2	11	商品コード
_	PDPDNM	42		O		12	53	商品名
_	PDJNCD	13		A		54	66	ＪＡＮコード
_	PDSPCD	6		A		67	72	仕入先コード
_	PDCATG	2		A		73	74	カテゴリー
_	PDUNPR	9	0	P		75	79	販売単価
_	PDCOPR	9	0	P		80	84	仕入単価
_	PDSTDT	8	0	S		85	92	発売日
_	PDENFG	1		A		93	93	取扱終了フラグ
_	PDUPDT	8	0	S		94	101	最終更新日
_	PDUPUS	10		A		102	111	最終更新者

仕入先コード ： 仕入先マスタ（MSUPLP）に登録された仕入先ＣＤとする。
カテゴリー ： カテゴリマスタ（MCATGP）に登録されたカテゴリＣＤとする。
取扱終了フラグ ： ブランク、"1"(生産終了)、"2"(在庫切れ)のいずれかとする。
販売単価、仕入単価 ： **販売単価＜仕入単価**は**エラー**となるよう**入力値チェック**を行う。
最終更新日 ： データ登録/変更時に、システム日付値数値8桁をセットする。
最終更新者 ： データ登録/変更時に、ログオンしているユーザーIDをセットする。

商品マスタの項目に含まれる[PDCATG](カテゴリー)は、先に紹介したカテゴリーマスタと関連付けする。同様に、[PDSPCD](仕入先コード)は仕入先マスタと関連付ける。[PDENFG](取扱終了フラグ)の値は、ブランクか“1”(生産終了)、“2”(在庫切れ)のいずれかとする。また、[PDUNPR](販売単価)と[PDCOPR](仕入単価)は、登録・更新時に大小チェックを行い、販売単価<仕入単価の場合はエラーとする入力チェックを設けたい。さらに画面上の表示項目とはしないが、データ登録・更新時には、[PDUPDT](最終更新日)、[PDUPUS](最終更新者)にも自動的に値をセットしたい。

商品マスタメンテナンスで使用する[Edit Grid]ウィジェットの設計画面は、**【図4】**である。

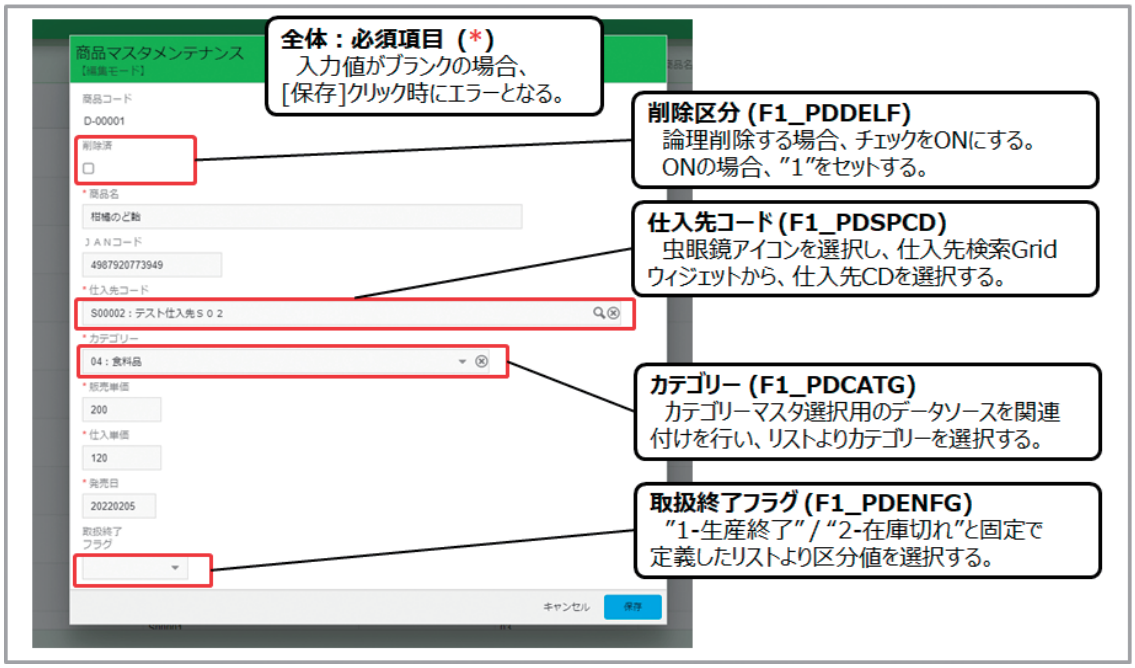
図 4　商品マスタメンテナンス　Edit Grid設計画面 – [編集]



入力必須項目は、単に[必須]チェックを付ければよい。また入力項目には、入力支援機能としてドロップダウン(プルダウン)、チェックボックス、あるいは[Grid]ウィジェットを関連付けしたマスタ検索画面の呼出しが定義可能である。ドロップダウンについては、リスト項目を別途定義したデータソースから作成する事や固定値の選択肢を直接定義する事ができる。詳細な定義内容は、実際のサンプルアプリの設計画面で確認してほしい。

作成したアプリケーションの実行画面は、**【図5】**である。設計画面で定義したとおりに、入力項目に入力支援機能が付加されている事がわかる。

図 5　商品マスタメンテナンス実行画面 - 編集ダイアログ



これで[Edit Grid]ウィジェットによる編集画面の定義は完了であるが、現状では、商品マスタメンテナンスアプリとして、以下の処理の実装が不足している。

- 販売単価と仕入単価の入力値妥当性チェック
- 新規登録時の商品コードキー重複チェック
- 非表示項目（最終更新日、最終更新者）の書き込み

このように、[Edit Grid]ウィジェットの設計画面では定義できないものは、どのように対処すればよいだろうか？「App Builder」の設計画面で定義できない独自のビジネスロジックや内部処理等の追加には、RPGプログラムを使用する。次章では、[Edit Grid]ウィジェットにおけるRPG追加手順を紹介する。

3. RPG連携テンプレート"EXNABVAL"の使用方法

「App Builder」におけるRPGプログラムは、1から全てを新規に作成する事も可能であるが、専用のAPIを利用する為に必要な各種定義を都度記述するのは面倒である。「App Builder」には、RPG連携の種類毎に合わせたテンプレートソースが用意されており、連携に必要な宣言等が予め定義されている為、このテンプレートソースを自身のソースライブラリにコピーして利用するのが一般的である。2023年8月現在の最新版であるValence6.2には、全部で8種類のテンプレートプログラムが用意されている。**【図6】**

図 6 Valence App Builder 連携RPGテンプレート

ソースファイル名	機能	概要
EXNABBTN	ボタンクリック	ボタンのクリックや、行クリック等のイベント処理を記述するテンプレート
EXNABCLOSE	アプリ終了	アプリケーション終了時実行される終了処理を記述するテンプレート
EXNABDS	データソース実行前	データソースを開く直前に実行したい処理を記述するテンプレート
EXNABFHLP	フォームヘルパー	Formウィジェットへの初期値セットや入力項目の値変更時のイベント処理を記述するテンプレート
EXNABFLT	フィルター	フィルター処理実行時に独自のフィルター条件を記述するテンプレート
EXNABIV	フィルター初期値	フィルター条件項目への初期値セットを記述するテンプレート
EXNABSTART	アプリ起動	アプリケーション起動時に実行される初期処理を記述するテンプレート
EXNABVAL	EditGridチェック	EditGridウィジェットで行追加・行変更・行削除において独自処理を記述するテンプレート

- **VALENCE6/QRPGLESRC**内に収録
  ※Valence6.2から完全なフリーフォーム形式で収録されている
  ※以前の形式のテンプレートは VALENCE6/QRRGLESRC2内に収録されている
- EXNABVAL以外は、全て**"process"**サブルーチンの中にユーザープログラムを記述すればよい。

本章でメインに紹介する"EXNABVAL"以外のテンプレートには、"process"サブルーチンが定義されている為、独自のユーザーロジックは、この中に記述すればよい。
なおRPG連携プログラムの基本的な作成方法やソースのコンパイル手順については、2019年度テクニカルレポート「Valence App Builder RPG連携テクニック」（https://www.migaro.co.jp/tr/no12/tech/12_01_06.pdf）で紹介しているのでそちらも参照してほしい。

では、ここから本題となる[Edit Grid]ウィジェットにおけるRPG追加手順を紹介する。ここで使用するテンプレートソースは、"EXNABVAL"である。このテンプレートは、[Edit Grid]ウィジェットで行追加/行編集/行削除の各種更新処理を実行する際に呼び出されるものである。
始めに「App Builder」上の設定箇所を見ておこう。[Edit Grid]ウィジェットの[編集]タブの中にある[設定]を開くと、オプション設定画面が開く。この中にある[検証]欄に呼出ししたいRPGのプログラムIDを記述すればよい。**【図7】**

図 7 Edit Grid – RPGプログラム設定方法

次にRPGロジックだが、この“EXNABVAL”は、追加/編集/削除の状態により、[更新前]、[更新後]、[エラー時]に処理を行う複数のサブルーチンが用意されている。**【図8】**

図 8　EXNABVAL サブルーチン一覧

サブルーチン名	概要
ProcessAdd	レコード登録前に　エラーチェックや 非入力項目へ値を設定するサブルーチン
ProcessDelete	レコード削除前に　エラーチェックを行うサブルーチン
ProcessEdit	レコード更新前に　エラーチェックや 非入力項目へ値を設定するサブルーチン
ProcessPostAdd	レコード登録後に、独自の追加ロジックを実行したい場合に使用するサブルーチン
ProcessPostDelete	レコード削除後に、独自の追加ロジックを実行したい場合に使用するサブルーチン
ProcessPostEdit	レコード更新後に、独自の追加ロジックを実行したい場合に使用するサブルーチン
ProcessErrAdd	レコード登録時に、エラーが発生した場合に独自のエラー処理を記述するサブルーチン
ProcessErrDelete	レコード削除時に、エラーが発生した場合に独自のエラー処理を記述するサブルーチン
ProcessErrEdit	レコード更新時に、エラーが発生した場合に独自のセラー処理を記述するサブルーチン

今回のプログラムでは、行追加/行編集の更新前に行うエラーチェックと非表示項目への値の書き込みが目的となる為、“ProcessAdd”と“ProcessEdit”の各サブルーチンに処理を記述すればよい。例えば、“ProcessAdd”サブルーチンの実装例は、**【ソース1】**である。

ソース 1　REP010:商品マスタメンテナンス更新チェック(一部抜粋)

```
** ------------------------------------------------------------
p ProcessAdd      b
d                 pi
D*-----変数宣言
D INPDCD          S             10A
D RECCNT          S             10S 0
D INUNPR          S              9S 0
D INCOPR          S              9S 0
D SYSDAT          S               D   DATFMT(*ISO) INZ(*SYS)   1-①
D*
 /free
   //-----画面入力値を取得
   INPDCD = GetValue('MPRODP':'PDPDCD');            //商品ＣＤ
   INUNPR = %DEC(GetValue('MPRODP':'PDUNPR'):9:0);  //販売単価
   INCOPR = %DEC(GetValue('MPRODP':'PDCOPR'):9:0);  //仕入単価   1-②

   //-----商品ＣＤのキー重複エラーチェック
   exec sql  SELECT COUNT(*) INTO :RECCNT FROM MPRODP
             WHERE PDPDCD = :INPDCD;
   if RECCNT > 0;
     SendError('キーが重複しています':'F1_PDPDCD');
     return;
   endif;                                           1-③

   //-----販売単価と仕入単価の相関チェック
   if INUNPR < INCOPR;
     SendError('販売単価が仕入単価未満です':'F1_PDUNPR');
     return;
   endif;                                           1-④

   //-----非入力項目の値をセット
   SetValue('MPRODP':'PDUPDT':%CHAR(%DEC(SYSDAT:*ISO)));  //最終更新日
   SetValue('MPRODP':'PDUPUS':GetAppVar('nabUser'));      //最終更新者

 /end-free                                          1-⑤
p                 e
** ------------------------------------------------------------
```

ソースの先頭部分はサブルーチン内で使用する変数宣言部である。今回システム日付値を最終更新日にセットする為に、1-①のようなシステム日付値(日付型)を宣言した。
1-②は、登録/編集画面で入力された値を取得する処理である。GetValueメソッドを使用すればよい。このメソッドは取得対象となるファイル名とフィールド名を指定すれば、画面入力値が文字列として取得できる。従って、数値型の変数へ代入する場合には型変換(%DEC)が必要となる。
1-③は、キー項目の重複チェック処理である。ここではSQLを使用して1-②で取得した商品CDを持つデータのレコード件数を取得し、1件以上存在する場合はエラーとなる処理とした。SendErrorメソッドを使用すれば、ブラウザ側にエラーメッセージを渡して更新処理を中断する事ができる。
1-④は販売単価と仕入単価との大小エラーチェックである。
1-⑤は、非入力項目へ値をセットする処理である。SetValueメソッドを使用すればよい。このメソッドは、ファイル名とフィールド名そしてセットしたい値をパラメータに指定すればよい。値は、フィールドの属性に関わらず文字列としてセットする点に注意してほしい。
例えば、[PDUPDT](最終更新日)は数値8桁の定義だが、"%CHAR(%DEC(SYSDAT:*ISO))"として、日付型のシステム日付値を一旦8桁の数値に変換したものを更に文字列に変換した値をセットしている。また、[PDUPUS](最終更新者)には、「App Builder」に用意されたデフォルトのアプリ変数"nabUser"の取得値をセットしている。"nabUser"は、ログインしているValenceユーザーIDを表している。
このように[Edit Grid]ウィジェットの定義だけで実現できない部分は、RPGロジックを追加する事で制御できるので覚えておこう。

4. [Edit Grid]ウィジェット活用術

前章では、[Edit Grid]ウィジェットの基本的な使用方法を紹介したが、現在の[Edit Grid]は、更なる便利な入力機能が実現できる。今回は、「登録/編集用の独自フォームの設定方法」、「インライン編集機能」そして、「ローカル編集機能」の3つを紹介したい。

4-1. 登録/編集フォームの独自設定

前章で紹介した商品マスタメンテナンスアプリだが、デフォルトの追加/編集用ダイアログ画面は、IBM i標準のDFU(Data File Utility)と同様に、編集画面のレイアウトを一切変更する事ができない。単に[Edit Grid]ウィジェットの[編集]タブの中に定義された入力可能項目を上から順番に表示するのみである。(順番自体は[Edit Grid]設計画面で変更が可能。)また、入力項目の細かな制御を行う事も現状難しい。
この為、[Edit Grid]ウィジェットを本格な入力系アプリに適用するのが以前は難しいと感じていたが、最新のValenceでは、この編集画面を独自に[Form]ウィジェットへ差し替える事ができるようになった。[Form]ウィジェットであれば、画面レイアウトを自由に設計でき、またFormヘルパー機能を組み合わせる事により、入力項目の細かな制御も可能になる。

本節では、前章の商品マスタメンテナンスに登録/編集用の独自フォームを設定する方法を紹介する。
まず、[Edit Grid]ウィジェットの関連元となるデータソースに対し、新たに[Form]ウィジェットを追加する。
次に[Edit Grid]ウィジェットの[編集]タブの中にある[設定]を開く。オプション設定の中にある[フォーム]欄に作成した[Form]ウィジェットを割り当てればよい。**【図9】**

図 9　Edit Grid – 編集用フォームの割り当て

[Form]ウィジェットを割り当てると、[追加時][編集時]の設定欄が、割り当てた[Form]ウィジェット上の項目表示に切り替わる為、ここでは、それぞれの場合に入力項目とするか、読取り専用にするかといった定義のみ行えばよい。

ここまでで、基本的な設定は完了である。アプリケーションを実行すると、編集画面が独自フォームに変更された事が確認できる。**【図10】**

図 10　商品マスタメンテナンス実行画面 –独自フォーム表示

更に[Form]ウィジェットでは、各入力項目に対し、入力制約が設定可能である。[Form]ウィジェットの設計画面にある[フィールド]タブで、各項目の定義を行うのだが、[フォーマッティング]欄の歯車アイコンを選択すると【図11】のような設定画面が使用できる。

図 11　Form – 入力項目の書式設定

従来から、[必須]項目や[大文字に自動変換]は設定可能であったが、Valence6.2では、[形式チェック]において、regexと呼ばれる正規表現設定が可能になった。これにより、入力項目の内容にあった文字列や数値以外エラーとするようなチェックが可能となっている。実際に今回の入力フォームでは、[PDJNCD](JANコード)は、数値13桁、[PDSTDT](発売日)は、数値8桁でかつ、日付値として有効な値のみといった設定を行っている。【ソース2】は、正規表現の定義例である。いろいろな場面で応用できると思われるので参考にしてほしい。

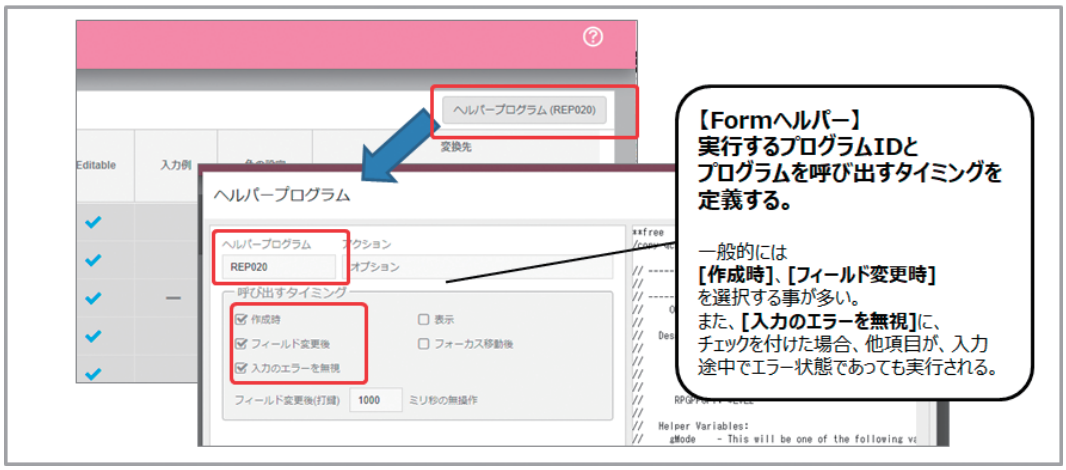
ソース 2　Formウィジェット［形式チェック］：正規表現設定例

```
//-----JANコード（13桁数値）
[0-9]{13}

//-----郵便番号（[3桁数値]-[4桁数値]）
^[0-9]{3}-[0-9]{4}$

//-----電話番号（[2-4桁数値]-[2-4桁数値]-[4桁数値]）
¥d{2,4}-¥d{2,4}-¥d{4}

//-----数値8桁の日付値(yyyymmdd:実在する日付以外はエラー)
^(?!([02468][1235679]|[13579][01345789])000229)(([0-
9]{4}(01|03|05|07|08|10|12)(0[1-9]|[12][0-9]|3[01]))|([0-
9]{4}(04|06|09|11)(0[1-9]|[12][0-9]|30))|([0-9]{4}02(0[1-9]|1[0-9]|2[0-
8]))|([0-9]{2}([02468][048]|[13579][26])0229))$
```

また、[Form]ウィジェットには、フォームを初期表示した際の初期値セットや、入力項目の値を変更した場合にインタラクティブなエラーチェックを行う為のFormヘルパー機能が実装できるようになっており、RPGプログラムで記述できる。Formヘルパー機能で使用するRPGテンプレートは、“EXNABHLP”である。この機能は、[Edit Grid]ウィジェットの編集用に設定した独自フォームからも使用する事ができる。

[Form]ウィジェット上の設定方法は、[フィールド]タブ内にある[ヘルパープログラム]ボタンをクリックし、実行したいRPGプログラムや、Formヘルパープログラムを呼び出すタイミングを指定する。**【図12】**

図 12　Form – Formヘルパー設定



Formヘルパーに独自処理を追加した実装例は、**【ソース3】**である。Formヘルパーへ呼び出されるタイミングが変数gModeに自動的にセットされるようになっており、また値変更時には、変更したフィールド名が変数gFieldにセットされるようになっている。**【図12】**の定義どおり、今回はフォーム生成時とフィールド変更時に呼び出す設定となっている為、3-①のような条件分岐を組み込めばよい。

プログラムの詳細だが、3-②は、新規レコード追加時に[PDSTDT](発売日)に初期値としてシステム日付をセットする処理である。

3-③は、[PDSPCD](仕入先コード)に値を入力した際に、SQLを使用して仕入先マスタを検索し、仕入先名をセットする処理である。SQL文を実行した結果対象データが存在しない場合は、SQLSTATEに“02000”がセットされる為、エラーを返す事ができる。

3-④は、[PDUNPR](販売単価)と[PDCOPR](仕入単価)との大小チェックである。

Formヘルパーを使用して、画面上の項目に値をセットする場合、SetValueメソッドを使用し、エラーを渡す場合、SetErrorメソッドを使用するという事を覚えておこう。

ソース 3 REP020:商品マスタ入力フォーム:入力チェック(一部抜粋)

```
** ------------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D INSTDT           S               8S 0
D SYSDAT           S                D   DATFMT(*ISO) INZ(*SYS)
D INUNPR           S               9S 0
D INCOPR           S               9S 0
D INSPCD           S               6A
D SQSPNM           S              42A
D*
 /free
   if gMode = 'formRender';                          3-①
     //フォーム生成時処理
     //-----発売日が０の場合システム日付を初期セット
     INSTDT = vvIn_num('F1_PDSTDT');                 3-②
     if INSTDT = *ZERO;
       SetValue('F1_PDSTDT':%CHAR(%DEC(SYSDAT:*ISO)));
     endif;
   elseif gMode = 'formShow';                        3-①
     //フォーム表示時処理
     //
   else;
     //項目変更時処理
     //-----仕入コード変更時
     if gField = 'F1_PDSPCD';                        3-①
       //入力値を取得
       INSPCD = vvIn_char('F1_PDSPCD');              3-③
       if INSPCD <> *blank;
         //仕入先マスタを検索
         exec sql SELECT SUSPNM INTO :SQSPNM FROM MSUPLP
                  WHERE SUSPCD = :INSPCD;
         //仕入先ＣＤが存在しない場合エラー
         if SQLSTATE = '02000';
           SetError(gField:'仕入先ＣＤが正しくありません');
           return;
         else;
           //仕入先名欄に取得値をセット
           SetValue('F2_SUSPNM':SQSPNM);
         endif;
       else;
         //仕入先名欄にブランクをセット
         SetValue('F2_SUSPNM':'');
       endif;
     endif;
     //-----販売単価ＯＲ仕入単価変更時
     if gField = 'F1_PDUNPR' or gField = 'F1_PDCOPR';  3-①
       //入力値を取得
       INUNPR = vvIn_num('F1_PDUNPR');               3-④
       INCOPR = vvIn_num('F1_PDCOPR');
       //-----販売単価と仕入単価の相関チェック
       if INUNPR <> *ZERO and INCOPR <> *ZERO;
         if INUNPR < INCOPR;
           SetError(gField:'販売単価が仕入単価未満です');
           return;
         endif;
       endif;
     endif;
   endif;
 /end-free
p                  e
```

このようにFormヘルパープログラムを組み込めば、編集フォーム上で値入力時のリアルタイムなエラーチェックやマスタから取得した名称の自動セットが行える。その為、入力系アプリとしての使い勝手を大幅に向上させる事ができる。

4-2. インライン編集機能

次に紹介するのは、[Edit Grid]ウィジェットにおけるインライン編集機能である。第2章で紹介した通り、[Edit Grid]ウィジェットは、Grid上で行メニューから[編集]を選択、あるいは行をダブルクリックし、編集用ダイアログ画面からデータをメンテナンスできる。しかし、インライン編集機能を有効にすると、Grid上のセルに値を直接入力する事や、直接行編集を行う事が可能となる。**【図13】**

図 13　Edit Grid - インライン編集機能

本節では、このインライン編集機能を使用して作成した商品マスタの価格一括入力プログラムを紹介する。設定は至ってシンプルである。[Edit Grid]ウィジェット設計画面の[編集]タブにある[設定]の[インライン]欄を「セル編集」に設定すれば良い。後は、[カラム]タブで設定したGridの表示項目の中から、入力対象となる項目を[編集時]欄に追加すればよい。

なお、インライン編集機能は現在のところ、[編集時]のみに対応している。[追加時]に入力項目を設定した場合は、通常の新規入力用ダイアログ画面が表示されるので注意してほしい。

インライン編集を有効にした場合も、“EXNABVAL”テンプレートを使用すれば、入力値のエラーチェックや非表示項目への値の書き込みが可能である。インライン編集時のプログラム実装例は、**【ソース4】**である。

基本的な考え方は、第3章で紹介した**【ソース1】**と同様だが、インライン編集の場合、エラー時の制御方法が異なる。通常、エラーが発生した場合は、SendErrorメソッドを使用すればよいが、インライン編集の場合はこのメソッドは利用できない。代わりに4-①のように、SetResponseメソッドを使用してレスポンスメッセージを送る必要がある。ただ、このままではエラーと判断されても更新処理が中断されない為、outStopProcess=*on; を記述して、後続の更新処理を明示的に中断させる必要があるので注意してほしい。

ソース 4 REP030:商品マスター括入力行更新チェック(一部抜粋)

```
p ProcessEdit     b
d                 pi
D*-----変数宣言
D INUNPR          S              9S 0
D INCOPR          S              9S 0
D SYSDAT          S               D   DATFMT(*ISO) INZ(*SYS)
D*
 /free
   //-----画面入力値を取得
   INUNPR = %DEC(GetValue('MPRODP':'PDUNPR'):9:0); //販売単価
   INCOPR = %DEC(GetValue('MPRODP':'PDCOPR'):9:0); //仕入単価

   //-----販売単価と仕入単価の相関チェック
   if INUNPR < INCOPR;
     //※インライン編集の場合SendErrorは使用不可
     //SendError('販売単価が仕入単価未満です':'F1_PDUNPR');
     SetResponse('success':'false');
     SetResponse('msg':'販売単価が仕入単価未満です');
     outStopProcess = *on;
     return;
   endif;

   //-----非入力項目の値をセット
   SetValue('MPRODP':'PDUPDT':%CHAR(%DEC(SYSDAT:*ISO))); //最終更新日
   SetValue('MPRODP':'PDUPUS':GetAppVar('nabUser'));      //最終更新者
 /end-free
p                 e
```

4-①

以上で、インライン編集機能の実装は完了である。インライン編集機能を使用すれば、ユーザーは画面を切り替える事なくデータを一括入力できる為、操作性は大きく向上するだろう。

4-3. ローカル編集機能

前節では、インライン編集機能を紹介したが、あくまで[Edit Grid]ウィジェットの基本動作は、選択した行に対するデータの更新である。複数明細が一括で入力可能なインライン編集機能だが、実際には行を選択して入力を行い、確定する毎に、都度データベースへの更新が行われる。しかし、一括入力を行う画面の場合、複数明細を画面入力後に、[保存]等のボタンをクリックして、複数データを一括でデータベースに反映するような画面構成を求められる事が多いだろう。最新の「App Builer」の[Edit Grid]ウィジェットは、この要望にも対応できる。それがローカル編集機能である。この機能は、[Edit Grid]ウィジェットによって表示されたデータについて、画面操作中は、データベースへの更新は行わずに、画面(ブラウザ)の中の値のみが更新される仕組みである。

実装方法だが、まず、[Edit Grid]ウィジェットの[設定]タブを開き、[ページング]カテゴリー内の[アクティブ]を無効にする。次に[データ]カテゴリーを開き、[ローカル]を有効にすればよい。

以上の定義でローカル編集機能が有効となる。この状態でアプリケーションを実行すると、インライン編集で画面上のセル入力で値を変更しても、データベースへは反映されなくなる。実際にデータベースに書き込む処理は、RPGプログラムで実装すればよい。

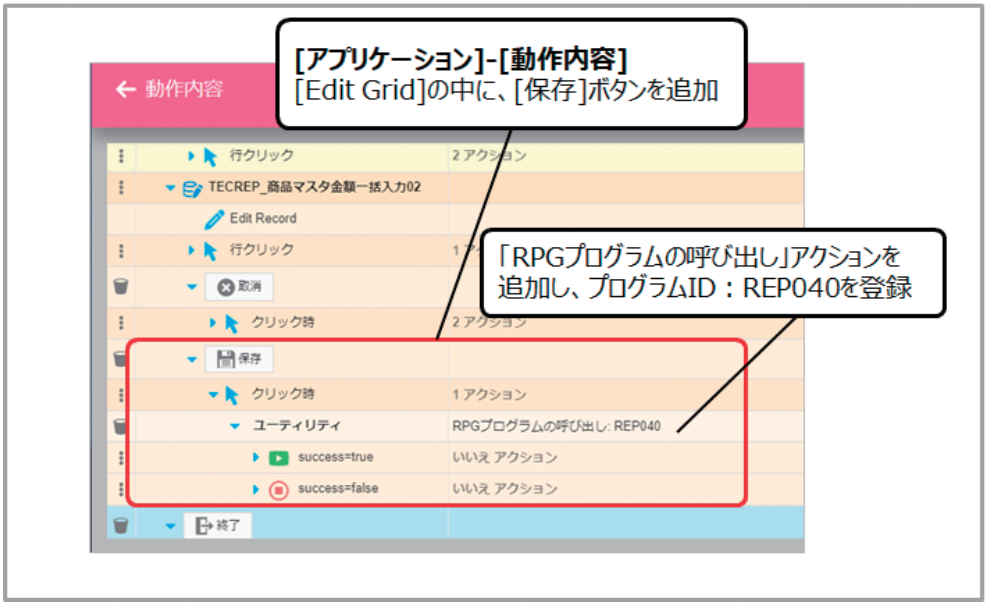
ローカル編集機能の実装例を紹介する。**【図14】**は、完成したアプリケーションの実行例である。

図 14 商品マスタ金額一括更新アプリ実装例

画面左部でカテゴリーを選択すると、該当するカテゴリーの商品が右側の[Edit Grid]ウィジェットに一覧表示される。販売単価と仕入単価がそれぞれセル編集となっている為、値を編集できるが、変更を行ってもローカル編集機能が有効な為、データベースには反映されなくなっている。最後にウィジェット下部にある[保存]ボタンをクリックした時に、一括でエラーチェックおよびデータベースへの反映を行うという流れである。

アプリケーション設計画面をみてみよう。[Edit Grid]ウィジェット上に追加した[保存]ボタンのイベントとして、[RPGプログラムの呼び出し]を追加し、REP040を登録している。**【図15】**

図 15 商品マスタ金額一括更新アプリ設計画面



ボタンクリック時に呼び出されるプログラムのRPGテンプレートは、“EXNABBTN”である。このテンプレートを元に作成したプログラム（REP040）が、**【ソース5】**である。

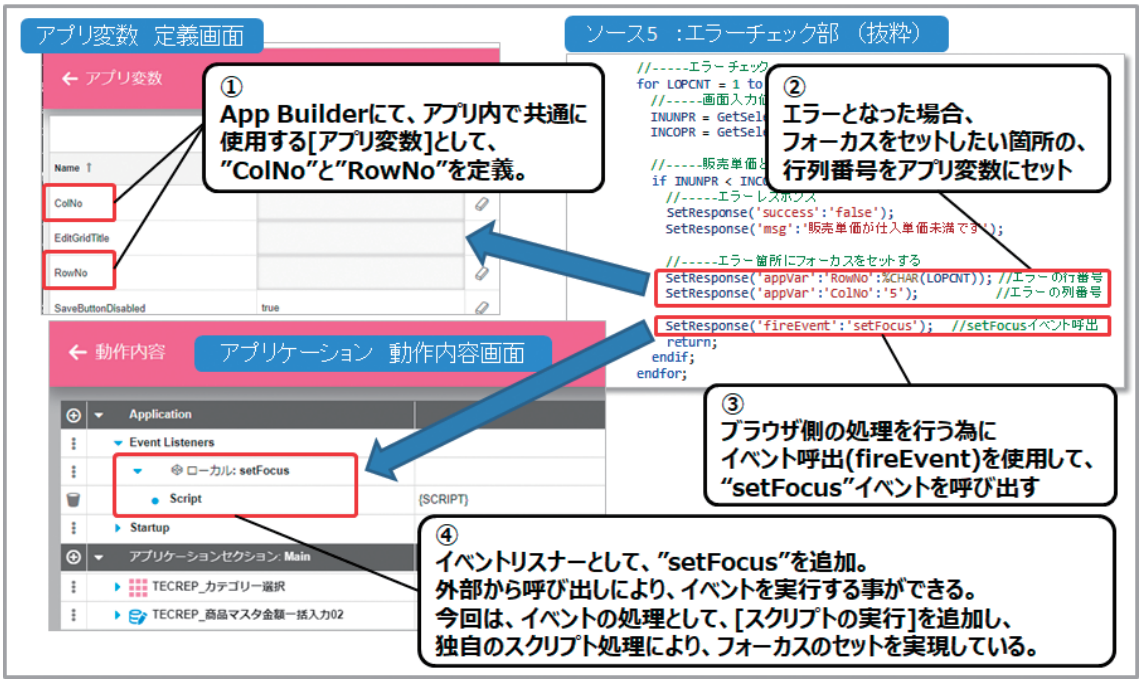
ソース 5　REP040：商品マスター括更新処理（一部抜粋）

```
** ----------------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D LOPCNT           S              9S 0
D INPDCD           S             10A
D INUNPR           S              9S 0
D INCOPR           S              9S 0
D SYSDAT           S               D   DATFMT(*ISO) INZ(*SYS)
D INUPDT           S              8S 0
D INUPUS           S             10A
D*
 /free
   //-----エラーチェック
   for LOPCNT = 1 to gSelectionCnt;                    5-①
     //-----画面入力値を取得
     INUNPR = GetSelectionNum(LOPCNT:'F1_PDUNPR');  //販売単価
     INCOPR = GetSelectionNum(LOPCNT:'F1_PDCOPR');  //仕入単価   5-①

     //-----販売単価と仕入単価の相関チェック
     if INUNPR < INCOPR;                                  5-②
       //-----エラーレスポンス
       SetResponse('success':'false');
       SetResponse('msg':'販売単価が仕入単価未満です');

       //-----エラー箇所にフォーカスをセットする
       SetResponse('appVar':'RowNo':%CHAR(LOPCNT)); //エラーの行番号
       SetResponse('appVar':'ColNo':'5');           //エラーの列番号
       SetResponse('fireEvent':'setFocus');   //setFocusイベント呼出
       return;
     endif;                                               5-④
   endfor;

   //-----更新処理
   //-----共通項目値を取得
   INUPDT = %DEC(SYSDAT:*ISO);           //最終更新日
   INUPUS = %Trim(GetAppVar('nabUser')); //最終更新者

   for LOPCNT = 1 to gSelectionCnt;                    5-①
     //-----画面入力値を取得
     INPDCD = GetSelectionChar(LOPCNT:'F1_PDPDCD'); //商品ＣＤ
     INUNPR = GetSelectionNum(LOPCNT:'F1_PDUNPR');  //販売単価
     INCOPR = GetSelectionNum(LOPCNT:'F1_PDCOPR');  //仕入単価   5-①

     //-----データの更新処理
     exec sql  UPDATE MPRODP SET                          5-③
               PDUNPR = :INUNPR,
               PDCOPR = :INCOPR,
               PDUPDT = :INUPDT,
               PDUPUS = :INUPUS
               WHERE PDPDCD = :INPDCD;
   endfor;

   //-----正常終了レスポンス
   SetResponse('success':'true');
   SetResponse('info':'更新が終了しました');
 /end-free
p                  e
```

このテンプレートでは、[Grid]や[Edit Grid]の各要素にアクセスする為のAPIが使用できるようになっている。5-①の部分に着目してほしい。gSelectionCntには、Gridの全行数が格納される。GetSelectionChar(文字項目)及びGetSelectionNum(数値項目)が、指定した行番号の要素にアクセスして、Grid上の内容を取得するメソッドである。
今回のプログラムでは、大きく分けてエラーチェック処理とデータ更新処理の2段階となっており、それぞれfor–endforを使用したループ処理としている。Grid上のデータを上から順番に取得しながら、5-②では入力値に対するエラーチェックを、5-③ではSQLを使用したデータの更新処理を行っている。

大きな流れは以上だが、5-②のエラーチェックでは、エラー発生時にSetResponseメソッドを使用して、レスポンスメッセージを作成している事がわかる。単にエラーメッセージダイアログをブラウザに表示するだけであれば、処理はシンプルなのだが、残念ながら現在の「App Builder」では、Grid上の項目でエラーが発生した際に、エラー項目にフォーカスをセットする事ができない。今回のアプリケーションでは、フォーカスをセットする為にもう一工夫を行っているので、その部分の実装方法を紹介する。
【図16】がエラー発生時にフォーカスをセットする処理の流れである。

図 16　エラー時に、対象フィールドにフォーカスセットする仕組み



フォーカス制御を行う為に、アプリケーション内に独自のアプリ変数"ColNo"(列番号)と"RowNo"(行番号)を定義している。**【ソース5】**の5-④の部分で、SetResponseメソッドのパラメータ"AppVar"を使用して、エラーが発生した行列番号をアプリ変数にセットしている。このようにRPG側からアプリ変数へ値をセットする事ができる。さらにSetResponseメソッドのパラメータ"fireEvent"を使用すれば、「App Builder」に定義した任意のイベントを呼び出す事もできる。ここでは、フォーカスをセットする為のイベントとして、"setFocus"イベントを呼び出している。
「App Builder」側の[アプリケーション]→[動作内容]画面では、独自のイベントを追加する[Event Listener]が用意されているので、ローカル新規イベントとして、"setFocus"を定義している。そして追加した"setFocus"イベントに、[スクリプトの実行]を追加すれば、独自のスクリプト処理を実行する事ができ、今回はスクリプトを使用してフォーカスをセットする処理を実行している。

スクリプトイベントの実装例は、**【ソース6】**である。少し複雑な処理の為、詳細は割愛するが、JavaScriptは原則非同期実行となる為、単純にスクリプトを記述するだけでは、実行時に表示されるエラーメッセージダイアログが表示されている間にスクリプトが終了してしまい、フォーカスが制御できない。従って、6-①のように、メッセージダイアログのオブジェクトを取得しておき、そのオブジェクトの要素が画面から削除された事（つまりメッセージダイアログが閉じられたという状態）を検知して、その後スクリプト処理を実行するような仕組みとしている。

6-②がアプリ変数の値を取得し、対象となる［Edit Grid］ウィジェットを取得する処理である。なお、ソース中の“PRODGrid”は、アプリケーション画面で、ウィジェットに付けた名前である。

6-③が、ウィジェット内のGrid上の対象となる行列番号の要素を取得する処理で、6-④がフォーカスをセットする処理である。実際には、対象となるセルを疑似的にダブルクリックする事で、フォーカスがセットされるという仕掛けとなっている。

ソース 6　**setFocusイベント　：　【スクリプトの実行】**

```
//メッセージBOX要素を取得
var Box = document.querySelector('.vv-common-dialog');  6-①

//要素が変化した場合、処理を実行
let observer = new MutationObserver(mutationRecords => {
    //フォーカスをセットする行列番号を取得
    var rowNo = getAppVar('RowNo');
    var colNo = getAppVar('ColNo');
    //対象となるEditGridウィジェットを取得
    var gridWidght = getWidget('PRODGrid'); //----【EditGridに設定した名称を記述】  6-②
    //ウィジェット内のgridを取得
    var grid = gridWidght.down('grid');
    //フォーカスをセットするセル要素を取得
    const el = '[data-recordindex="' +  String(Number(rowNo) - 1) + '"]';
    var Td = grid.getEl().dom.querySelector(el).querySelectorAll('td')[Number(colNo) - 1];  6-③

    //ダブルクリックイベントの定義
    const dblclick =  new MouseEvent("dblclick", {
      bubbles: true,
      cancelable: true,
      view: window,
    });

    //セル要素ダブルクリックイベントの呼出し
    function settd(){
        Td.dispatchEvent(dblclick);
    }

    //500ミリ秒経過後にダブルクリックイベントを実行
    setTimeout(settd,500);
});  6-④

// 対象となる要素の追加・削除を監視
observer.observe(Box, {
  childList: true
});
```

この仕組みは若干複雑だが、他のアプリケーションでも応用できるよう汎用的な仕組みとして検討しているので、是非試してみてほしい。

5.複数行明細入力フォーム作成テクニック

ここまで主に商品マスタを使用して、[Edit Grid]ウィジェットの活用方法を紹介してきたが、最後に応用例として受注入力を行うアプリケーションの実装例を紹介する。
【図17】が、完成したアプリケーションの実行画面である。
アプリケーションを起動すると、新規受注を登録する為の画面として、左側に[受注見出]用の[Form]ウィジェットが、右側に[受注明細]用の[Edit Grid]ウィジェットが表示される。受注明細は、10明細分の登録が可能な構成となっている。画面上部の[保存]ボタンをクリックする事で、受注ファイルおよび受注明細ファイルに新規レコードを登録するといった処理である。

図 17　受注入力(新規登録)画面実装例

このアプリケーションのポイントとなるのが、[Edit Grid]に関連付けられる受注明細情報である。前章までに紹介した商品マスタとは異なり、今回のアプリは、受注明細情報の新規登録であり、元となるデータが存在しない為、そのまま受注明細ファイルをデータソースとして紐づける事はできない。そこで今回は、明細入力用に**【図18】**のようなレイアウトをもつ受注明細ワークファイルを定義した。

図 18　受注明細ワークファイル　ファイルレイアウト

DSPFMT　　レ コ ー ド 設 計 書　　日付 23/09/01　時刻 19:30:36

物理ファイル	TECREPLIB/WODRDP	様式名	WODRDR	レコード長	101
様式記述	受注明細ワーク				

5= 詳細

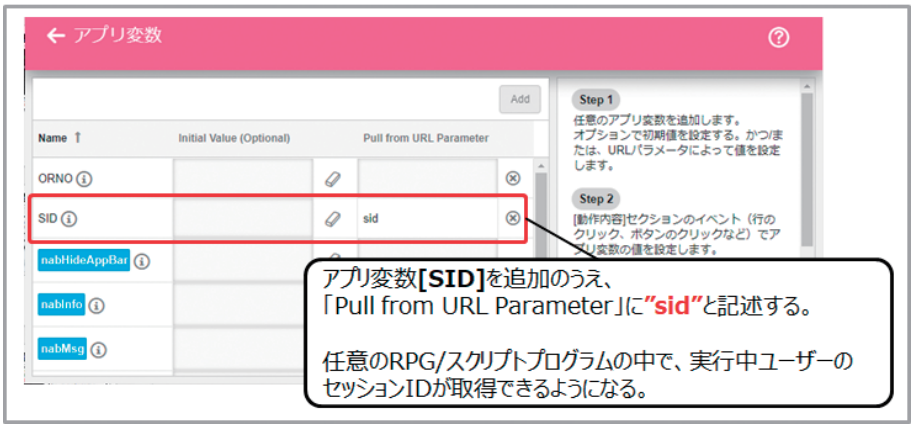
選択	項目名	桁数	属性	キー順	開始	終了	テキスト記述／欄見出し
_	WDSID	64	A	1 ANN	1	64	セッションＩＤ
_	WDLNNO	2 0	S	2 ASN	65	66	行ＮＯ
_	WDPDCD	10	A		67	76	商品ＣＤ
_	WDQTY	6 0	S		77	82	数量
_	WDUNIT	9 0	S		83	91	単価
_	WDPRCE	10 0	S		92	101	金額

セッションID　：　実行しているブラウザ（セッション）を識別するID。
行NO　：　10明細分（1～10）の行番号

一般的にPC5250アプリをRPG等で開発する場合、ワークファイルを使用する際に、QTEMPライブラリを使用する事が多い。実行中一つのジョブを専有する形で処理される5250セッションであればこの方法は有効だが、残念ながらValenceは、ブラウザ上の複数セッションでジョブを共有利用する仕組みの為、QTEMPライブラリを使用する事は現実的ではない。そこで、Valenceで実行するユーザーのブラウザ毎に独自のワークファイルを作成する際に役立つのが、[セッションID]である。「Valence Portal」に含まれる「活動セッション」アプリを実行するとわかるが、Valenceを実行中のユーザーには、一意となる64桁の[セッションID]が自動的に付与されるようになっている。この情報をワークファイルのキー項目にする事で、ユーザー毎に独立したワークファイルを処理する事が可能となる。

「App Builder」で作成するアプリケーションにおいて、[セッションID]を使用する方法を紹介する。アプリ変数の設定画面にて変数“SID”を追加する。そして、オプションパラメータの[Pull from URL Parameter]に小文字で“sid”を記述すれば良い。これで、任意のRPGプログラムやブラウザ側のスクリプト処理のなかで[セッションID]が利用できるようになる。**【図19】**

図 19　アプリ変数へセッションID追加

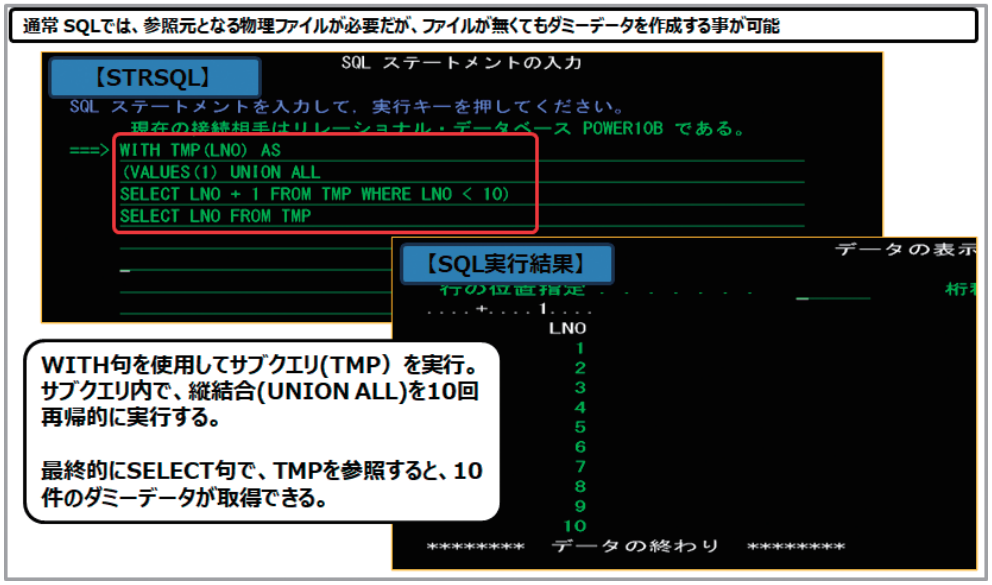


次に、この[セッションID]を使用して受注明細ワークファイルを作成する方法を紹介する。「App Builder」には、アプリケーションの起動時及び終了時に呼び出し可能なRPGプログラムが設定できるようになっている。[動作内容]画面の中にある[起動時/閉じる]欄である。この中に、起動時/終了時に実行したいプログラムを定義すれば良い。**【図20】**

図 20　アプリケーション – 動作内容　：　起動時 / 閉じる

最後にワークファイルを作成するテクニックを紹介する。今回はアプリケーション起動時にセッションIDをキー項目とした空のワークファイルを10明細分作成する必要がある。もちろんRPG処理において、ファイルを10件WRITEすれば良いのだが、SQLを使用すれば1回のSQLで任意の件数の空データを作成できる。肝となるのが、**【図21】**のようなSQLである。

図 21　複数レコードのダミーデータを作成するSQL



通常SQLを使用する場合、参照元のファイル(テーブル)をFROM句に指定する必要があるが、実はファイルを使用しないSQLも作成できる。今回のSQLでは、数値1つを取得するサブクエリを10回再帰的に実行する事で、1から10の10明細分の結果セットを取得する事ができる。この仕組みを使用して取得した複数レコードのダミーテーブルをワークファイルに挿入(INSERT)すれば良い。

ソース記述例を紹介する。まず、起動時に実行するRPGプログラムは、**【ソース7】**である。このプログラムは、RPGテンプレート“EXNABSTART”を元に作成すれば良い。

ソース 7

REP050:アプリケーション起動時処理(一部抜粋)

```
** ------------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D LNCNT            S              2S 0
D SID              S             64A
D*
 /free
   //-----ワークファイル作成に必要な設定値
   LNCNT = 10;                 //ワークファイルを作成する行数
   SID = GetAppVar('SID');     //セッションＩＤ

   //-----同じセッションＩＤのレコードを全て削除
   exec sql DELETE FROM WODRDP WHERE WDSID = :SID;

   //-----指定した明細行分、ダミーレコードを作成する
   //SQLを使用して、明細用ワークファイルにレコードを追加
   exec sql INSERT INTO WODRDP (WDSID, WDLNNO)
            WITH TMP(LNO) AS (VALUES(1) UNION ALL
            SELECT LNO + 1 FROM TMP WHERE LNO < :LNCNT)
            SELECT :SID, LNO FROM TMP;
 /end-free
p                  e
```

7-① 7-②

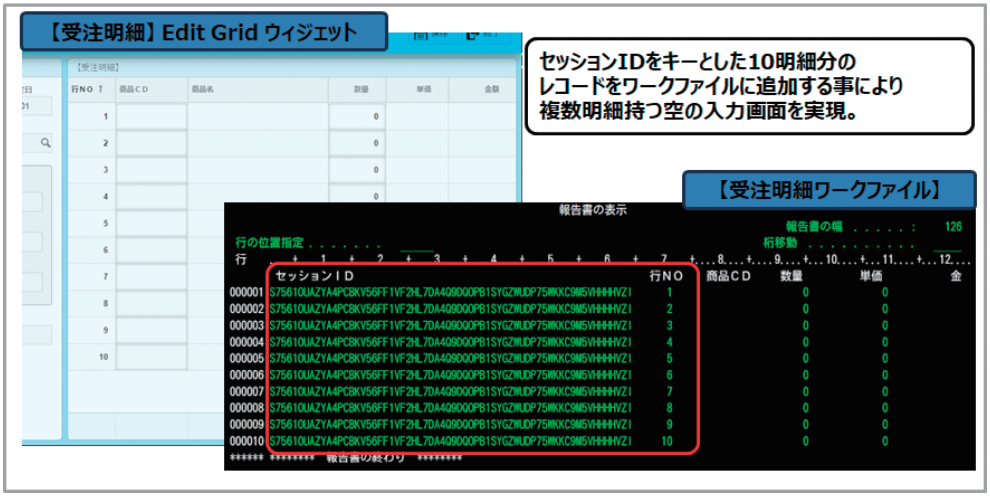
7-①にてセッションIDを取得している。7-②のようなSQLを実行すれば、[セッションID]をキーとした複数件の明細をもつワークファイルを一度のSQL処理で作成できる。
なお、ワークファイルは、アプリケーション終了時に削除する必要もある。削除時のRPGプログラム実装例は**【ソース8】**である。終了時も起動時と同様に、8-①にてセッションIDを取得し、8-②のようなSQLを使用してワークファイルを削除している。なお、アプリケーション終了時プログラムは、“EXNABCLOSE”テンプレートから作成する。
以上で、受注明細ワークファイルを使用した複数明細を持つ入力用[Edit Grid]ウィジェットの作成は完了である。完成したアプリケーションの実行結果は、**【図22】**である。

ソース 8

REP060:アプリケーション終了時処理(一部抜粋)

```
** ----------------------------------------------------------
p Process          b
d                  pi
D*-----変数宣言
D SID              S              64A
D*s
 /free
   //-----セッションＩＤ取得
   SID = GetAppVar('SID');                    8-①

   //-----ワークファイルの削除
   //使用したセッションＩＤのレコードを削除
   exec sql DELETE FROM WODRDP WHERE WDSID = :SID;   8-②
 /end-free
p                  e
```

図 22　初期プログラム(REP050)実行結果



このように複数明細行を持つ入力画面を[Edit Grid]ウィジェットを使用して作成したい場合、今回紹介した仕組みが活用できるので、ぜひ参考にしてほしい。

6.さいごに

本稿では、「App Builder」における『[Edit Grid]ウィジェット活用術』として、[Edit Grid]ウィジェットの基本から機能強化された各種活用法についてサンプルを交えて紹介してきた。冒頭にも案内したとおり、紙面だけでは伝わりにくい部分がある為、今回Valence6.2上で動作するサンプルアプリを用意している。是非、サンプルアプリをインポートの上、実際の動作や、定義内容を確認してほしい。[Edit Grid]ウィジェットを使用したアプリ開発におけるヒントが見つかるであろう。皆様も[Edit Grid]ウィジェットの活用に色々挑戦してみてほしい。