



# *NXJ* プログラミング 言語ガイド

*Release 11.5/Composer*

---

© 2002-2006 Unify Corporation All rights reserved. Sacramento California, USA

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise without the prior written consent of Unify Corporation.

Unify Corporation makes no representations or warranties with respect to the contents of this document and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Unify Corporation reserves the right to revise this document and to make changes from time to time in its content without being obligated to notify any person of such revisions or changes.

The Software described in this document is furnished under a Software License Agreement. The Software may be used or copied only in accordance with the terms of the license agreement. It is against the law to copy the Software on tape, disk, or any other medium for any purpose other than that described in the license agreement.

The Unify Corporation Documentation Group values and appreciates any comments you may have concerning our documents. Please address comments to:

doc@unify.com

(800) 468-6276 or (800) 468-6343; (916) 928-6400  
FAX (916) 928-6401

UNIFY and DataServer are registered trademarks of Unify Corporation. Unify NXJ is a trademark of Unify Corporation. Java and J2EE are registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. JReport is a trademark of Jinfonet Corporation. IBM, Lotus, Lotus Notes, Cloudscape, and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries, or both. CAS AHL Technology and ecKnowledge are registered trademarks of CAS AHL Technology, Inc. in the U.S. and other countries. All other products or services mentioned herein may be registered trademarks, trademarks, or service marks of their respective manufacturers, companies, or organizations.

Name: NXJ Programming Language Guide

Release: Unify NXJ 11.5/Composer

Last Revision: January 16, 2006 10:34 am

---

# 目次

---

<b>1. 概要</b>	<b>1</b>
イベント駆動の実行	1
NXJ プログラミング言語の機能	1
フォームスクリプトを使用する理由	2
フォームスクリプトとは？	3
フォームスクリプトの開発プロセスの概要	5
Unify NXJ フォームプロセッシング API	7
その他の Java Class を使用	8
<b>2. スクリプトエディタの使い方</b>	<b>9</b>
スクリプトエディタを使うには	9
フォームスクリプトのテキストの色	11
イベントセクションの追加	11
自動補完機能の使用	13
テキストの検索	14
ブックマークの使い方	15
スクリプトエディタのその他のコマンド	16
Java Class ファイルの扱い	17
<b>3. NXJ 変数</b>	<b>19</b>
複数値変数と単一値変数	19
ターゲット（複数値）変数とフィールド	21
非ターゲット複数値変数とフィールド	23
単一値変数とフィールド	24
ターゲットテーブルの NULL 値の扱い方	26
NXJ 変数プロパティの使い方	29
clearFindExp	31
clearAddExp	32

フィールドとコントロールへのアクセス例 .....	33
外部リファレンス .....	38
<b>4. スクリプトの構造と構文</b> .....	<b>41</b>
スクリプトの構造 .....	41
スクリプトの構文 .....	43
FORM セクション .....	44
開発者定義コマンド宣言 .....	46
Form イベントセクション .....	50
DATA VIEW セクション .....	53
Data View イベントセクション .....	55
FIELD セクション .....	60
Field イベントセクション .....	61
Control セクション .....	63
Control イベントセクション .....	65
推奨するイベントセクションの順 .....	66
SQL 文の埋め込み .....	68
シンプル埋め込み SQL 文 .....	68
コネクション .....	69
動的パラメータ .....	70
例 .....	70
埋込み型 SELECT 文 .....	70
埋込み型ロック文 .....	73
エラー処理 .....	75
<b>5. アプリケーションの実行</b> .....	<b>79</b>
NXJ のイベント .....	79
実行シーケンスのダイアグラム .....	81
アプリケーションの開始 .....	82
アプリケーションの終了 .....	83
フォームの初期化 .....	85
選択セットの初期化 .....	86
次フォームに移動 .....	88
前フォームに移動 .....	91
ズームフォームに移動 .....	93
ズームフォームの取り消し .....	95

レコードの検索 .....	96
Clear-to-find 操作 .....	98
SQL Select 文の構成 .....	100
レコードの追加と更新 .....	102
Clear-to-add 操作 .....	105
レコードの削除 .....	107
次レコード に移動 .....	110
前レコードに移動 .....	112
フィールド位置 .....	114
次フィールドに移動 .....	116
前フィールドに移動 .....	117
ユーザ入力の処理 .....	120
ユーザ入力のプロンプト .....	122
<b>6. フォームスクリプトのデバッグ</b> .....	<b>127</b>
デバッグのプロセス .....	128
タスク 1: デバッグアプリケーションサーバが 開始されていることを確認する .....	128
タスク 2: フォームがコンパイルされていることを確認する .....	128
タスク 3: 最初のブレークポイントを設定する .....	128
タスク 4: デバッグ > 開始 を選択して NXJ デバッガを開始 .....	129
タスク 5: アプリケーションの実行を調査 .....	129
タスク 6: デバッグ セッションの終了 .....	130
アプリケーションデータの表示 .....	131
<b>A. データタイプ マッピング</b> .....	<b>135</b>



この章では Unify NXJ プログラミング言語の概要を説明します。NXJ プログラミング言語は、NXJ アプリケーションレベルのオブジェクトをサポートするビジネス言語でコンパイラも付属しており、Java 2 Platform, Standard Edition (J2SE) で提供される Java オブジェクトもサポートしています。

NXJ プログラミング言語の機能はオプションなので、NXJ プログラミング言語を使わなくても対話型のデータベース操作を行うアプリケーション全体を定義できますが、NXJ プログラミング言語を使えばアプリケーションデザイナーのフォームやフィールドコントロールのプロパティでは提供されていないカスタマイズを行うことができます。

## イベント駆動の実行

重要なのは、NXJ プログラミング言語では NXJ アプリケーションをイベント駆動で実行するという考え方であることです。実行時にあるイベントが発生した場合、例えば、ユーザが NXJ ツールバーの **検索** ボタンを押下した場合、NXJ インタクションサーバは一定のタスクを開始し、対応する検索操作が実行されます。

このような実行形態により、NXJ プログラミング言語文は、フォームスクリプト（あるイベントが発生したときに、実行する NXJ インタクションサーバへの命令を示したファイル）に記述されます。例えば、AFTER FIND イベントセクションにフォームを検索した後で実行するビジネスルールを記述して、レコードの選択セットの列の合計を出すこともできます。

## NXJ プログラミング言語の機能

NXJ プログラミング言語は、従来のプログラミング言語よりも速く簡単にビジネスロジックが作成できるように設計されています。この利点を実現するために、NXJ プログラミング言語には次のような機能が備わっています。

- 
- フォームのフィールドコントロールに相当する変数が自動的に宣言され、対応するデータベースの列にバインド
  - JDBC インタフェースではなく標準の SQL 文で記述
  - NXJ 変数を使うと、データベースの null になりうる値を保持
  - 必要に応じて JavaBean 形式のプロパティをプログラミング可能
  - NXJ プログラミング言語スクリプトは、アプリケーションデザイナーのスクリプトエディタを使って構文要素の自動補完機能を利用した編集が可能

NXJ プログラミング言語には以下のような仕様があるので、Java プログラマは注意が必要です。

- 式を持つプロパティを設定する場合、Java プログラムでは式は設定時に評価されますが、NXJ プログラミング言語では設定時には評価されず式を持つプロパティが使用されるたびに評価されます。これには以下のプロパティが該当します。
  - clearAddExp
  - clearFindExp
  - zoomReturnExpressions
- NXJ 変数と Java 変数では null の扱い方が異なります。

## フォームスクリプトを使用する理由

NXJ アプリケーションは、通常以下のような機能を提供するためにフォームスクリプトを使用します。

- 実行時の条件に基づいて、フォーム、データビュー、フィールドコントロール、コントロールオブジェクトのプロパティを上書きできます。例えば、注文書のフォームで、ユーザが現在住んでいる州と“州”フィールドが異なる場合にのみ、州によって税率が違うので“税”フィールドを利用可能にすることが、NXJ プログラミング言語文では、“税”フィールドの ENABLED と VISIBLE プロパティを TRUE に設定することで実現できます。また、NXJ プログラミング言語を利用して、ユーザがフォーム上のフィールドに入力した値を判定し、無効なデータが入力されたフィールドのラベルを赤のイタリック書体で表示させることもできます。
- フォーム上に対応するフィールドコントロールがない場合でもデータソースのデータが操作できます。例えば、特定の列の合計を計算したり、注文時の税額を計算できます。

- 
- コマンドを待ち行列に入れて、プログラマ的に操作をトリガーできます。詳しくは、NXJ Javadoc で NXJSession ファウンデーションクラスの `queueCommand` メソッドを参照してください。
  - アプリケーションの実行前と実行後にアクションを実行します。例えば、カウンタ変数の初期化やクリアを行います。

## フォームスクリプトとは？

フォームスクリプトは、アプリケーションデザイナーで作成して管理するファイルです。プロジェクトのフォームクラスを作成すると、アプリケーションデザイナーでフォームスクリプトのテンプレートが作成されます。フォームスクリプトには、イベント発生時に実行する NXJ プログラミング言語文を持つイベント固有のコードセクションを追加できます。このセクションはイベントセクションと呼ばれ、例えば NXJ が提供するメソッドを呼び出したり、このフォームのスクリプトやアプリケーションの他のスクリプトでも定義されているユーザ独自のメソッドを呼び出したりできます。

例えば、以下のフォームスクリプトでは“getNextNum”というメソッドが定義され、使用されています。

```
FORM XReport
{
  private int getNextNum(String type)
  throws java.sql.SQLException,
  com.unify.NXJ.mgr.datatypes.NXJNullValueException,
  com.unify.NXJ.mgr.dataConnection.NXJDataConnectio...
  {
    int result=0;
    EXEC SQL SELECT NUM FROM ROFIDA.NEXT_NUMBER
      WHERE NTYPE= :type
    INTO result;
    EXEC SQL UPDATE ROFIDA.NEXT_NUMBER SET
      NUM=:result + 1 WHERE NTYPE=:type;
    return result;
  }
}

ON CLEAR TO ADD
{
  throws java.sql.SQLException,
  com.unify.NXJ.mgr.datatypes.NXJNullValueException,
  com.unify.NXJ.mgr.dataConnection.NXJDataConnectio...
  {
    REPORT_ID = getNextNum( "XREPORT" );
  }
}

...(rest of the script not shown)
```

ローカル  
メソッド宣言。  
埋め込み SQL  
が使用されて  
いる

ユーザが新規  
レコードを追  
加するために、  
追加操作を開  
始する New ボ  
タンをクリック  
すると  
メソッド呼び  
出しが発生す  
る

フォームスクリプトには、変数宣言やコマンド定義など、他のタイプの文も含めることができます。フォームスクリプトに含めることができる文については、[41 ページの「スクリプトの構造と構文」](#)を参照してください。

---

## フォームスクリプトの開発プロセスの概要

NXJ プログラミング言語フォームスクリプトを作成する基本ステップは、以下のとおりです。

1. デザインパネルでフォームの基本レイアウトを完成します。

スクリプトのテンプレートは、フォームのデータビューとフィールドコントロールで構成されているので、これらの要素を使うと、フォームスクリプトを簡単に書くことができます。

要素はどの時点でも追加 / 削除でき、それと同時にスクリプトも更新されます。

フォームのレイアウトについては、『Unify NXJ 開発者ガイド』の第 5 章「フォーム」と第 6 章「コントロール」を参照してください。

2. ビジネスルール、開発者定義コマンド、その他のロジックをフォームスクリプトに追加します。

NXJ クラスはパッケージをインポートしなくても利用することができます。他のクラスを使用する場合は、プロジェクトクラスパスを設定してそのプロパティで Jar ファイルが利用できるようにします。アプリケーションデザイナーでクラスパスを設定するには、**プロジェクト > プロパティ > クラスパス** を選択します。

3. **プロジェクト > すべて Make** を選択してフォームスクリプトをコンパイルします。

**Make** コマンドでスクリプトをコンパイルし、フォームごとに対応した Java コードを作成します。

コンパイルの方法については、『Unify NXJ 開発者ガイド』の第 10 章「テスト、パッケージングと配備」を参照してください。

4. コンパイルエラーが報告されたら、エラーを修正してステップ 3 を繰り返します。

コンパイルエラーに関してはメッセージパネルを参照してください。報告されたエラーをクリックすると、フォームスクリプト内の該当行が、スクリプトエディタ ウィンドウにハイライト表示されます。

- 
5. フォームの **実行** ボタンのドロップダウンリストから、フォーム名を選択して、フォームを実行します。

ツールバーの **実行** コマンドをフォームを選択して実行すると、指定したフォームのコンパイル、make、配備が行われます。プロジェクト全体を実行する場合は **プロジェクト > 実行** を選択します。  
(メニューバーの **プロジェクト > 実行** を選択した場合も同様です。)

実行 ボタン



フォームリスト  
にアクセスする  
ためにクリック

6. フォームをテストするには、実際に利用する場合と同じようにフォームを使用します。

詳細については、『Unify NXJ チュートリアル』を参照してください。

7. アプリケーションが予想外の結果を起こした場合、フォームスクリプトをデバッグするために NXJ デバッガを使用します。

[127 ページの「フォームスクリプトのデバッグ」](#)を参照してください。

## Unify NXJ フォームプロセッシング API

Unify NXJ フォームプロセッシング API は、フォームスクリプトで使用できるメソッドとプロパティを含む、NXJ プログラミング言語ファウンデーションクラスを説明しています。Javadoc を利用するには、アプリケーションデザイナのメニューバーで、ヘルプ > Javadoc を選択します。Unify NXJ フォームプロセッシング API リンクをクリックします。最初のページが表示されます。

**All Classes**

- [DataViewEventListener](#)
- [FieldEventListener](#)
- [FormEventListener](#)
- [Nullable](#)
- [NullableAmount](#)
- [NullableAmountField](#)
- [NullableAmountVariable](#)
- [NullableAny](#)
- [NullableAnyVariable](#)
- [NullableBinary](#)
- [NullableBinaryField](#)
- [NullableBinaryVariable](#)
- [NullableBoolean](#)
- [NullableBooleanField](#)
- [NullableBooleanVariable](#)
- [NullableDate](#)
- [NullableDateField](#)
- [NullableDateTime](#)
- [NullableDateTimeField](#)
- [NullableDateTimeVariable](#)
- [NullableField](#)
- [NullableFloat](#)
- [NullableFloatField](#)
- [NullableFloatVariable](#)
- [NullableNumeric](#)
- [NullableNumericField](#)
- [NullableNumericVariable](#)
- [NullableString](#)
- [NullableStringField](#)
- [NullableStringVariable](#)
- [NullableText](#)

Welcome to the Javadoc for the NXJ jBiz foundation classes.

Here are some pointers on how to find your way around the Javadoc:

- The foundation classes correspond to elements in a jBiz script. The table below can be used to find the foundation class that corresponds to a particular element.
- Notice the list of interfaces and classes in the frame to the left. This list can be used to quickly navigate to the description of that particular interface or class -- just click on the name of the class you want to see.
- If you know what you are looking for but do not know where to find it, try the [alphabetical index](#) of all fields, methods, and classes.

The following table identifies the foundation class that corresponds to the indicated script element.

Script Element	Foundation Class
COMMAND	<a href="#">NXJCommand</a>
DATA VIEW	<a href="#">NXJDataView</a>
FIELD	The foundation class for a field depends upon its data type. See the table below.
FORM	<a href="#">NXJForm</a>
<form>.session	<a href="#">NXJSession</a>

The following table identifies the foundation interface for fields and variables of the various data types.

各ファウンデーションクラスのプロパティとメソッドが説明されています。以下のファウンデーションクラスがあります。

- セッション、すなわち、実行時にアクティブなフォームのセット

- 
- データビュー
  - フォーム
  - フィールド、すなわち、フォーム上のフィールドコントロールと NXJ 変数
  - コマンド
- 

**注** – Javadoc では、ファウンデーションクラスのプロパティは“フィールド”として識別されます。

---

## その他の Java Class を使用

NXJ アプリケーションでは、J2SE クラスが提供するメソッドを使用することができます。例えば、NXJ アプリケーションが必要としている、ビジネスルールの振る舞いを定義した Java クラスセットがすでにある場合、一度フォームからクラスを利用できるようにすれば、フォームスクリプトからクラスメソッドを呼び出すことができます。

NXJ アプリケーションで Java Class を利用できるようにするには、次の 3 つの方法があります。

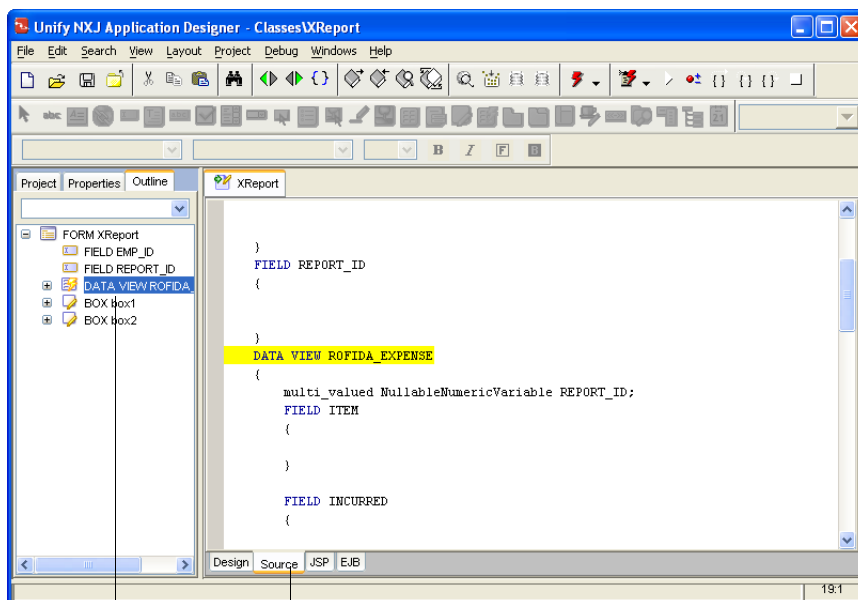
- 既存の Java Class を、個別ファイルとしてプロジェクトにバンドルする  
アプリケーションデザイナーで **プロジェクト > ファイルの追加** コマンドを選択し、プロジェクトにクラスファイルを追加します。追加したファイルは、スクリプトエディタで編集して **Make** コマンドでコンパイルできます。
- 新規 Java Class を作成してプロジェクトにバンドルする  
アプリケーションデザイナーで **ファイル > 新規** コマンドを選択し、新しい Java ファイルに名前を付けて追加します。スクリプトエディタを使ってファイルを編集します。 **Make** コマンドを使ってファイルをコンパイルします。
- Java アーカイブ全体をプロジェクトにバンドルする  
プロジェクトプロパティダイアログを使って、プロジェクトに Jar ファイルを追加します。 **プロジェクト > プロパティ > クラスパス** を選択し、Jar ファイルの名前とディレクトリを指定します。実行時にクラスが必要な場合は、JAR をアプリケーションパッケージに含めるように、チェックボックスをセットします。

# スクリプトエディタの使い方 2

フォームスクリプトを書いたり、NXJ デバッガを実行するときに、アプリケーションデザイナーのスクリプトエディタを使用します。( [127 ページの「フォームスクリプトのデバッグ」](#)を参照してください。)

## スクリプトエディタを使うには

あるフォームのスクリプトエディタを開くには、ブラウザパネルのプロジェクトタブでフォーム名をクリックします。デザインパネルにフォームが開かれるのでソースタブをクリックすると、スクリプトエディタにフォームスクリプトが表示されます。スクリプトアウトラインを表示するには、アウトラインタブをクリックします。



アウトラインで  
選択したセクション

ソースタブ

---

スクリプト中に行番号を表示するかどうかを設定できます。編集 > オプション > エディタを実行し、エディタオプションを設定して下さい。行番号は、Make コマンドが、ある行に関するエラーを報告した場合などに便利です。

---

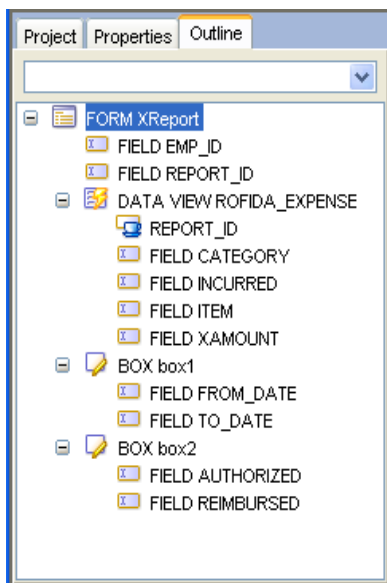
注 - 検索のようなスクリプトエディタのコマンドは、キーストロークにマッピングされています。マッピング内容はアプリケーションデザイナのオプションとして変更できます。マッピングを変更するには、編集 > オプション > キーボード を選択します。

---

アウトラインタブは、大きなスクリプトの中を簡単にナビゲートする方法を提供します。アウトラインタブには、フォーム、コンテナ、スクリプトのコントロールセクションとそれらのイベントセクションが表示されます。コマンドもアウトラインに表示されます。

アウトラインのエントリをクリックすると、ソースパネル内で該当するセクションまで移動してそのソースが表示されます。

もしスクリプトにコンパイルエラーがあると、アウトラインタブ内の先頭にある Errors フォルダにエラーが表示されます。



---

## フォームスクリプトのテキストの色

スクリプトエディタでは、スクリプトの要素タイプ別に決まった色で表示されています。デフォルトの色は、以下のとおりです。

表 2-1 フォームスクリプトの要素で使用されている色

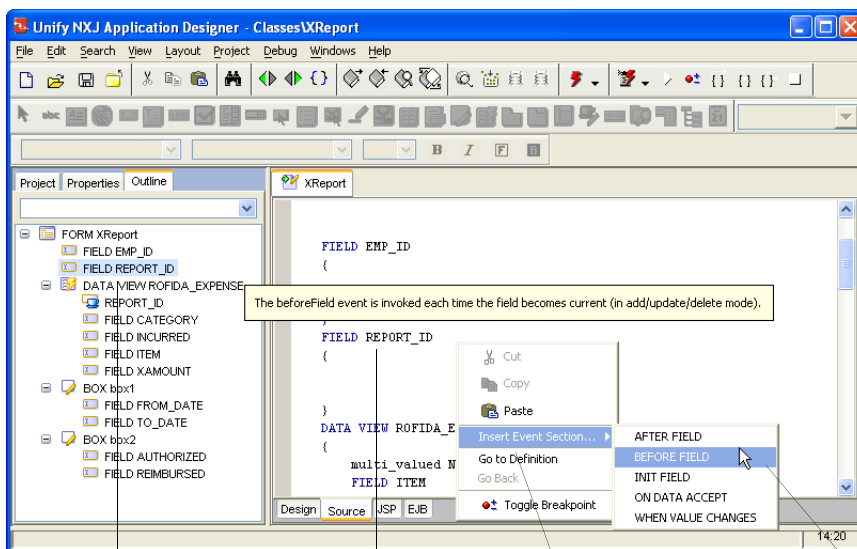
要素	デフォルトの色
コメント	緑
定数	青
区切り文字	黒
識別子	黒
キーワード	紫
Line Numbers	緑
文字列	青

要素タイプに割り当てられている色は、アプリケーションデザイナー オプションダイアログのエディタパネルで変更できます。**編集 > オプション** を選択すると、このダイアログを開くことができます。

## イベントセクションの追加

スクリプトエディタは、イベントセクション名を簡単に追加することができます。フォームスクリプトにイベントセクション名を追加するには、まず、イベントセクションを追加したいスクリプトセクションをクリックします。アウトラインパネルを使うと簡単に該当のスクリプトセクションを指定することができます。

フォームセクションで右クリックしてイベントセクションの挿入を選択します。該当するセクションにおいて有効なイベントセクションだけがリストされます。挿入したいイベントセクションを選択します。



1, 手を加えたいスクリーンセクションをクリックします。

2, イベントセクションを加えたい場所をクリックします。

3, 右クリックをして、イベントセクションの挿入を選択します。

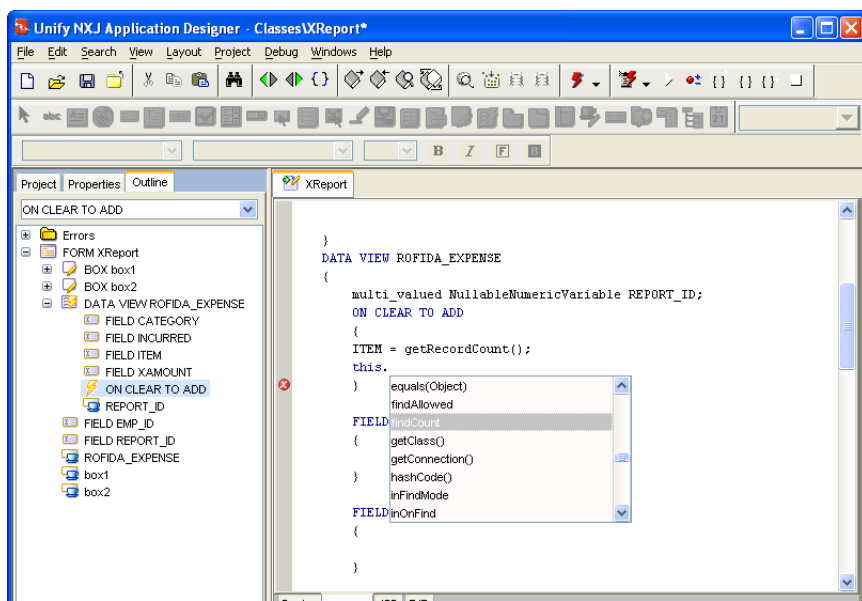
4, 追加したいイベントセクションをクリックします。

ドロップダウンリストからイベントセクションを選択すると、イベントセクション名と中括弧 ( { } ) がフォームスクリプトに追加されます。

イベントセクションを追加した後、中括弧の中に NXJ プログラミング言語文を記述できます。

## 自動補完機能の使用

スクリプトエディタには、フォームスクリプトと Java ソースファイル内のインポート文、変数、フィールド、メソッド、パッケージ、クラス名に関する自動補完機能があります。例えば、DATA VIEW イベントセクションで“this.”と入力すると、自動補完機能により、com.unify.nxj.mgr.datatypes.NXJDataView クラスで利用することが可能なプロパティとメソッド名が表示されます。



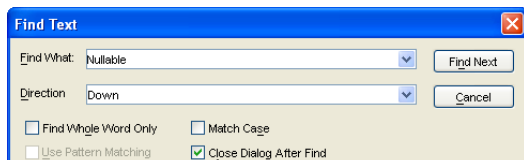
修飾名の最後にピリオドを入力すると、選択されたプロパティとメソッド名が自動補完機能によって自動的に表示されます。自動補完機能による選択肢が自動的に表示されない場合に、それらを表示するには、Esc キーを押下します。

デフォルトでは、自動補完機能はオンになっています。多量で複雑なスクリプトの場合には、コーディングの初期の段階では自動補完機能をオフにすることもできます。自動補完機能の選択肢をポップアップで表示するには数秒かかるのでコーディングの妨げになるからです。自動補完機能をオフにするには、編集 > オプション でアプリケーションデザイナーオプションダイアログを表示し、エディタパネルの自動補完機能を使用 チェックボックスをクリアします。自動補完機能がオフになっていても、ESC キーを押すことで自動補完の候補を出すことができます。

---

## テキストの検索

スクリプト内のテキストを検索するには、検索テキストダイアログを使用します。このダイアログは、**検索 > 検索** を選択すると利用できます。

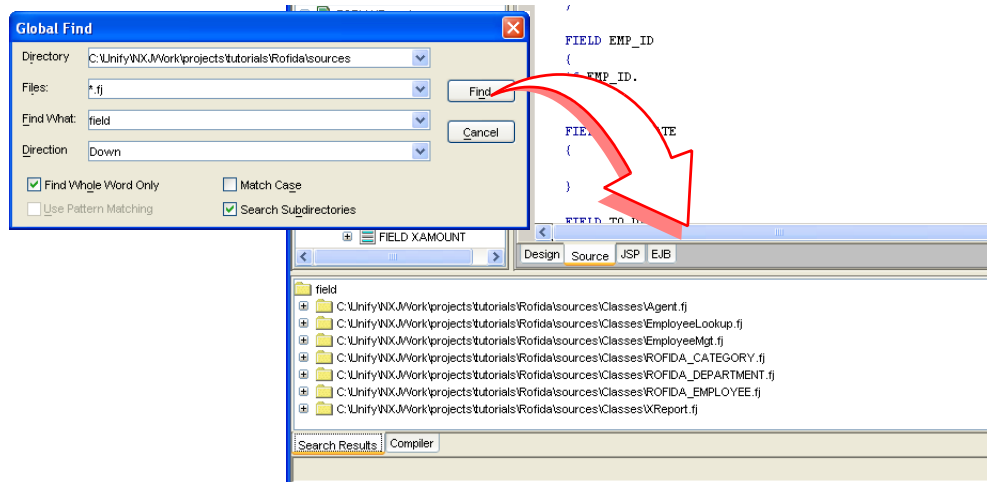


検索文字列フィールドでテキストを入力し、**次を検索** ボタンをクリックします。エディタ上で開かれているファイルのみが検索されます。検索された最初のテキストは、スクリプトエディタでハイライトされます。

次を検索するには、このダイアログで **次を検索** ボタンをクリックします。検索後ダイアログを閉じる チェックボックスをクリアしてダイアログを表示させたままにしておくと、**次を検索** ボタンがすぐに使用できます。

スクリプトエディタでは、プロジェクト全体から検索ができます。プロジェクト全体を検索するには、**検索 > 全体から検索** を選択します。全体から検索ダイアログが表示されます。

検索するプロジェクトファイルのタイプを入力します。通常はフォームスクリプトのファイル (.fj) です。全体から検索した結果は、アプリケーションデザイナー ウィンドウ下部の検索結果パネルに表示されます。



検索結果パネルのエントリをクリックすると、一致するテキストを含むオブジェクトのディレクトリがわかります。

## ブックマークの使い方

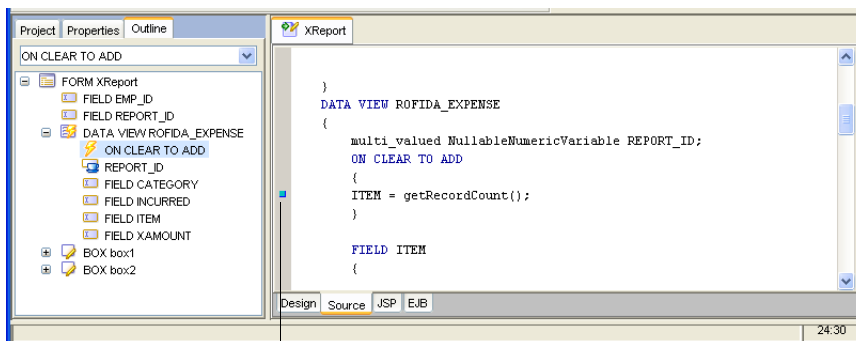
フォームスクリプトのサイズが大きい場合は、ブックマークを使って指定した行にマーカーをつけることができます。ブックマークを設定したり、ブックマークに移動する方法は以下のとおりです。

1. ブックマークを設定するフォームスクリプトの行にカーソルを置きます。

ブックマークを初めて指定する場合は、Control-Shift-0 を押下します。デフォルトでは、このコマンドシーケンスが最初のブックマークに対応していますが、アプリケーションデザイナーオプションダイアログのエディタタグでこの割り当てを変更することができます。

次にブックマークを指定する場合は、別の行で Control-Shift-1 を押下します。その次に指定するブックマークでは、Control-Shift-2 を押下します。以後、同じように押下していきます。

ブックマークは、スクリプトエディタ ウィンドウの左端に青い四角のマークで表示されます。



ブックマーク

2. ブックマークが設定されている行に移動するときは Control-n を押下します。n はブックマークの番号です。

例えば、最初のブックマークに移動するときは Control-0 を押下します。

## スクリプトエディタのその他のコマンド

スクリプトエディタでは、前のセクションで説明したコマンドの他にもテキスト編集用の便利なコマンドをサポートしています。以下の表を参照してください。

表 2-2 スクリプトエディタのコマンド

コマンド	説明
インデント解除	カレント行のインデントを解除します。
インデント	カレント行にインデントを設定します。
定義へ移動	現在の識別子が定義されている行にカーソルを移動します。これは、サブクラスに対して作業をしている場合に便利です。定義が別のファイルにある場合には、アプリケーションデザイナーがそのファイルを開きます。
元の位置へ戻る	定義へ移動の後で、定義へ移動を行った元の場所へ戻します。

表 2-2 スクリプトエディタのコマンド

コマンド	説明
対応する括弧	カーソルに最も近い括弧から次に対応する括弧にカーソルを移動します。
元に戻す	最後に行ったコマンド操作を元に戻します。
コピー	選択したテキストをクリップボードにコピーします。
切り取り	選択したテキストを削除してクリップボードにコピーします。
貼り付け	クリップボードのテキストをカレントのカーソル位置に貼り付けます。
ワードの初めまで選択	テキストをカレントのカーソル位置からワードの先頭まで選択します。
ワードの終わりまで選択	テキストをカレントのカーソル位置からワードの終わりまで選択します。
次のワードまで選択	テキストをカレントのカーソル位置から次のワードの先頭まで選択します。
前のワードまで選択	テキストをカレントのカーソル位置から前のワードの先頭まで選択します。

これらのコマンドに割り当てられたキーボードのキーは変更可能です。アプリケーションデザイナーのメニューから **編集 > オプション > キーボード** を選択し、変更して下さい。

## Java Class ファイルの扱い

スクリプトエディタを使って Java Class とスクリプトファイルを編集することもできます。 [8 ページの「その他の Java Class を使用」](#) を参照してください。

(日本語訳注：Java のクラスファイルを直接編集できるものではありません。)



この章では、NXJ プログラミング言語フォームスクリプトで使用できる変数について説明します。フォームスクリプトでは、任意の Java 変数を Java プログラミングと同じ方法で宣言して使用できます。Unify NXJ は、さらに NXJ アプリケーションのビジネスルールを簡単にプログラミングできるように Java 変数をいくつか拡張して提供します。これらの拡張は、複数值変数、null 可能型 (nullable) とプロパティを持つ変数があります。いずれも独自の宣言キーワードを持ち、式では独自の動作をします。詳しくはこの章で説明します。

NXJ プログラミング言語コードを作成する上で最も重要な技術の一つに、データビューでのインスタンス変数管理があります。このため、この章では次のような重要な概念を特に詳しく説明します。

- 複数值変数の動作を単一値変数との対比で理解
- ターゲットテーブルから NULL 列値を処理
- NXJ 変数プロパティの使い方を理解

## 複数值変数と単一値変数

ビルトインアプリケーションモデルの強力なデータ管理機能の一つに、選択セットがあります。インスタンスメンバ変数に加えて、各フォームとデータビューは、選択セットを持っています。最も単純な選択セットは、フォームが現在扱っているレコードのグループです。このレコードの一つがカレントレコードです。このレコードに対して、対話型のコマンドが適用されます。

各レコードは、ターゲットテーブルの行に対応しています。一部の選択セット、特にマスタ / 詳細関係の詳細ビューは自動的に生成されます。その他の選択セットは、検索条件やユーザが発行する検索コマンドによって作成されます ([96 ページの「レコードの検索」](#)を参照して下さい)。選択セットは、追加 / 更新 / 削除 操作を使って変更することもできます ([102 ページの「レコードの追加と更新」](#)と [107 ページの「レコードの削除」](#)を参照して下さい)。

---

ターゲット列の値だけではなく他の値も選択セットに含めると、ユーザインタフェースを充実させたり、フォームのビジネスロジックを単純化することができます。追加された値は、複数値フィールドコントロールと変数で表現されます。複数値フィールドコントロール（および変数）は、選択セットのレコードごとに独立した値を持ちます。

これとは逆に、複数値として宣言されないデータビューのインスタンス変数は、選択セットのレコード数とは関係なくデータビューに対して単一値を持ちます。もちろん、フォームとデータビューが作成されるたびに単一値のインスタンス変数のコピーが独自に作成されるので、この点には注意が必要です。

---

**注** – 複数値変数は、マルチオカレンス領域を持つフォームやデータビューとは異なります。マルチオカレンス領域を使うと、選択セットの複数のレコードを画面に同時に表示できます。複数値変数の値は、フォームやデータビューにシングルオカレンスがあるかマルチオカレンスがあるかに関わらず、選択セットのレコードごとに独立しています。

---

変数は、2つの方法によってデータビューで作成されます。レイアウトのフィールドコントロールとして作成する方法と、NXJ プログラミング言語スクリプトで宣言する方法です。

- レイアウトのフィールドコントロールは、フォームやデータビューではインスタンス変数になります。フィールドコントロールを使うことで自動的に作成されるため、NXJ プログラミング言語スクリプトで宣言する必要はありません。実際、同じ名前のインスタンス変数をフィールドとして宣言するとエラーが発生します。宣言ができるのは FIELD セクションだけです。フィールド名は、NXJ フィールドコントロールインタフェースで宣言したのと同じように、式で使用できます。NXJ フィールドコントロールインタフェースについては、[26 ページの表 3-2 「Nullable 値のインタフェース」](#)を参照してください。

フィールドが複数値かどうかは Multi Valued プロパティで設定します。このプロパティが設定されていれば、フィールドは複数値になります。このプロパティの詳細については、『Unify NXJ 開発者ガイド』の第 6 章「コントロール」を参照してください。

- NXJ プログラミング言語スクリプトで変数を宣言して、データビューでインスタンス変数を作成することもできます。フォームやデータビューの変数は、Java メンバー変数の構文ルールに従って宣言します。つまり、フォームやデータビューの中括弧内で宣言し、フォームやデータビューの他のオブジェクト（メソッド、FIELD、COMMAND）の中括弧内で宣言しないようにします。

宣言した変数が複数値であるかどうかは、宣言に “multi\_valued” フラグが使用されたかどうかで決まります。“multi\_valued” というキーワードは、“public” や “private” といった他の Java 変数フラグとともに、任意の順序で使用できます。

以下の表に変数のカテゴリがまとめられています。内容については、この後で説明します。

表 3-1 NXJ 変数のカテゴリ

	ターゲット	非ターゲット
複数値	ターゲットテーブルのデータを表します。	選択セットの各レコードの追加情報が格納されます。
単一値	ターゲット変数は、単一値にはできません。	データビュー自身の情報が格納され、選択セットのレコードの情報は格納されません。

## ターゲット（複数値）変数とフィールド

データビューのインスタンス変数の最も重要な利用方法は、ターゲット列の値を選択セットに保持することでしょう。行がターゲットテーブルから取り出されると、ターゲット列に相当する複数値は、自動的に値が設定されます。ターゲット変数（およびフィールド）の値は、add/update 操作を行うことで列に値を設定します。

変数をターゲット変数とするには、2つの条件が真でなければなりません。まず、変数名と列の名前が一致すること、そして変数が複数値であることです。フォームやデータビューのレイアウトを設計する場合、アプリケーションデザイナーにより、フィールド名とターゲットテーブルの列の名前が自動的に一致します。名前が一致すると、フィールドは自動的に複数値になります。アプリケーションデザイナーのデータベースウィザードにより、ターゲットフィールドが自動的に作成されます。

ターゲットフィールドの例は、ROFIDA.EMPLOYEE テーブルの EMP\_ID、FIRST\_NAME、LAST\_NAME、EMAIL、DEPT、MANAGER です。次の図の RERS Employee Details のフォームには、これらのターゲットフィールドに対するフィールドコントロールがあります。

---

**RERS Employee Details**

DEPT:

EMP\_ID:  MANAGER:

Personal Information

FIRST\_NAME:

LAST\_NAME:

EMAIL:

フォームスクリプトでは、ターゲットフィールドに対応する変数が、適切なインタフェース型、そして対応するフィールドコントロールと同じ名前ですべて宣言されます。変数は、フォームスクリプトのどのセクションでも利用できます。以下の例では、FIRST\_NAME 変数と LAST\_NAME 変数を使って、カレントレコードの fullName 値を作成する方法が示されています。

```
BEFORE RECORD
{
    NullableString fullName;
    fullName = FIRST_NAME + " " + LAST_NAME;
    ...
}
```

ターゲットフィールドと同じように、ターゲット変数もデータベースの列にマッピングされます。ただし、ターゲット変数はフォームのフィールドコントロールにも割り当てられるわけではありません。ターゲット変数は、ターゲットフィールドとは異なるインタフェースのセットで宣言されます。

ビジネスロジックが選択セット内の値を扱う必要のあるとき、ターゲット変数は有効ですが、フォームで表示するには意味がなかったり、適切ではない場合もあります。例えば、XReport フォームを変更して Authorized や Reimbursed といったラベルやフィールドを削除し、“status” という String フィールドコントロールや “status\_date” という Date フィールドを追加すると、マネージャはこれらのフィールドを見て、承認済みのレポートと未承認のレポートを簡単に区別できます。

フォームスクリプトには、以下のような新しい行が追加されます。

```
Form XReport
{
    multi_valued NullableDateVariable AUTHORIZED;
    multi_valued NullableDateVariable REIMBURSED;
}
```

---

```
...
ON FIND
{
  if ( REIMBURSED.isNull() )
  {
    if ( AUTHORIZED.isNull() )
    {
      status = "Open";
      status_date = null;
    }
    else
    {
      status = "Authorized";
      status_date = AUTHORIZED;
    }
  }
  else
  {
    status = "Reimbursed";
    status_date = REIMBURSED;
  }
}
```

## 非ターゲット複数値変数とフィールド

変数とフィールドコントロールは、必ずしもデータベースの列に割り当てなくても複数値にすることはできます。時には、フォーム上の各ターゲット変数に追加の情報を含めるために、フィールドコントロールがレイアウトに追加されます。フォームとデータビューがマルチオカレンスであれば、オカレンスごとにそのレコードに適した値が含まれるように、フィールドを複数値にする必要があります。

非ターゲット複数値変数の例には、関連するテーブルの値の合計、ユーザが読めるように表示された外部キーであるターゲット値の説明（例えば XReport の ROFIDA\_EXPENSE データビューにある CategoryName フィールド）、計算内容（例えば、ターゲット変数が表す項目の税額合計）があります。

非ターゲット変数 / フィールドの値はターゲットテーブルから与えられないので、NXJ プログラミング言語スクリプトでプログラムの与える必要があります。一般的に、複数値変数 / フィールドは ON FIND イベントセクションで設定します。

---

**注** – ON FIND イベントセクションは頻繁に実行されるので、イベントセクションに追加する処理 / コードのステップ数に注意する必要があります。できれば、非ターゲット値を計算する前に不要なレコードをリジェクトしてください。

---

以下の例では、“notifyEmail” という非ターゲット複数値変数を使ってマネージャの電子メールアドレスを格納しています。

```
FORM XReport
```

```
multi_valued NullableStringVariable notifyEmail;
```

```
private void sendMail (String emailAddr, int reportID )
{
    //pretend we sent mail
    System.out.println( "To: " + emailAddr );
    System.out.println("From: Expense Request System");
    System.out.println("Subject: Request Updated");
    System.out.println("This is to notify you that the Request #" +
        reportID + " has been updated");
}
...
ON FIND
{
    EXEC SQL SELECT EMAIL FROM ROFIDA.EMPLOYEE
        WHERE EMP_ID = :MANAGER
        INTO notifyEmail;
}
AFTER UPDATE
{
    //Notify the employees manager of updates
    sendMail( notifyEmail, REPORT_ID);
}
```

## 単一値変数とフィールド

単一値だけが必要なとき、すなわち選択セットのレコード数との関連がない場合は、単一変数を利用した方がいいでしょう。これらは、Java プログラミングの一般的な手法で、例えばルックアップして獲得する値のコピーや、関連する Java オブジェクトへのリファレンスを保持する場合に使われます。

---

例えば、XReport フォームに合計という追加フィールドコントロールがあり、XAMOUNT フィールドの下に表示されるとします。

フォームスクリプトは合計を計算し、Amount フィールドにそれを表示します。

```
Form XReport
{
...
DATA VIEW ROFIDA_EXPENSE
{
  NullableAmountVariable tempAmount;
  ...
  BEFORE FIND
  {
    total = 0.0;
  }
  ON FIND
  {
    total += XAMOUNT;
  }
  AFTER ADD
  {
    if ( session.status == StatusCode.SS_NORM )
      total += XAMOUNT;
  }
  BEFORE DELETE
  {
    tempAmount = XAMOUNT;
  }
  AFTER DELETE
  {
    if ( session.status == StatusCode.SS_NORM )
      total -= tempAmount;
  }
  BEFORE UPDATE
  {
    EXEC SQL
      SELECT XAMOUNT FROM ROFIDA.EXPENSE
      WHERE ITEM = :ITEM AND REPORT_ID = :REPORT_ID
      INTO tempAmount;
  }
  AFTER UPDATE
  {
    if ( session.status == StatusCode.SS_NORM )
      total = total - tempAmount + XAMOUNT;
  }
}
```

```
}  
}  
...
```

## ターゲットテーブルの NULL 値の扱い方

列で null 値を扱うことがなければ、対応するターゲット変数には単純に列に対応するタイプを宣言します。列で null 値を扱う可能性があれば、ターゲット変数には null 値を扱えるタイプを使用する必要があります。Java 基本型は null 値を扱わないので、代わりに Java オブジェクトタイプを使用します。

Java 基本型の変数で null 値を選択すると、NXJNullValueException がスローされません。

NXJ プログラミング言語は、null 値を許可されている列の値を表すために使うことのできる Java タイプのセットが用意されています。nullable 値変数を宣言するには、以下の表に示す Nullable インタフェースのいずれかを使用します。

表 3-2 Nullable 値のインタフェース

データタイプ	インタフェース
AMOUNT	NullableAmount
BINARY	NullableBinary
BOOL	NullableBoolean
DATE	NullableDate
DATETIME	NullableDateTime
FLOAT	NullableFloat
NUMERIC	NullableNumeric
STRING	NullableString
TEXT	NullableText
TIME	NullableTime
任意のデータタイプ、変更可能	NullableAny

---

**NullableAny** インタフェースは、この型の変数がどんな型のデータでも保持することができることを意味します。NullableAny 変数は最初に代入されたデータの型を以降保持できるよう設定されます。ここで一旦設定された型は、**makeTypeUndefined()** が呼び出され、NullableAny 変数が他のデータ型変数を受け取れるようになるまでは、NXJ インタラクションサーバによって関連付けられたままになります。

Nullable 値変数は、フォームスクリプト内であればどこでも宣言して使用することができます。

Nullable 値のインタフェースには特徴的な動作があります。Java プログラマには不慣れな動作かもしれませんが、SQL プログラマには自然な動作に思えるものです。具体的には次のような仕様があります。

- 変数名だけが Java 式においてその値を表す。null の場合も同様である
- 式で変数を使う場合は、その変数では null は ANSI SQL セマンティクスに準拠する
- **isNull()** メソッドを使って値の null チェックができる

Nullable 値変数は、SQL null を格納する以外にも、基本型の Java インスタンス変数と同じように使用できます。

例えば、以下の NXJ プログラミング言語文では NullableAmount 変数を使って合計を算出しています。

```
NullableAmount totalPrices(  
    NullableAmount[] prices  
)  
{  
    NullableAmount total = 0.00;    // assignment of constant value  
    for ( int index = 0 ; index < prices.length ; ++index )  
    {  
        total = total + prices[index]; // addition and assignment  
    }  
    return total;  
}
```

---

Nullable 値変数は論理式にも使用できます。例えば、以下のようになります。

```
boolean paymentEqualToTotal(
    NullableAmount paymentValue
)
{
    return (paymentValue == totalPrices(priceTable)); // comparison
}
```

Nullable タイプ変数が SQL null 値を保持する場合、結果として SQL null セマンティクスを表します。例えば、計算式に SQL null を使用すると、式全体が null 値を受け取ります。

```
NullableAmount a = null;
NullableAmount b = 5.25;
NullableAmount c = a + b; // the result of the addition is null.
```

SQL null 値を保持する Nullable タイプ変数を真偽判定の文脈で使用すると、変数は FALSE と評価されます。例えば、以下のような宣言があるとします。

```
NullableBoolean m = null;
NullableBoolean n = ! m; // negating null results in null
```

以下のような評価になります。

- “if ( m )” は false と評価される
- “if ( n )” は false と評価される
- “if ( ! m )” も false と評価される

`isNull()` メソッドを使うと、Nullable タイプ変数が null を保持しているかどうかテストすることができます。等号や不等号の演算子 (`==` や `!=`) は使用しないでください。例えば、先ほどの宣言で “m” を null に設定した `NullableBoolean` にすると、評価は以下のようになります。

- “if ( m.isNull() )” は true と評価される
- “if ( m == null )” は false と評価される
- “if ( m != null )” も false と評価される

---

Nullable タイプの変数を Java 基本型に代入する文脈で使用する場合、値はコンパイラによって適切な基本値に自動的に変更されます。値が NULL の場合、NXJNullValueException がスローされます。

例えば

```
NullableNumeric n = 5;
int i;
i = n;
n = null;
i = n; // this will throw NXJNullValueException
```

## NXJ 変数プロパティの使い方

NullableVariable インタフェースによって記述されたプロパティは、フォームやデータビューで定義されたメンバインスタンス変数に関連付けることができます。プロパティは、フォームやデータビューのメンバインスタンス変数にだけ関連付けられることに注意して下さい。プロパティを関連付けるには、以下の表にある NXJ メンバ変数インタフェースのどれかを変数タイプとして使用します。

表 3-3 NXJ 変数インタフェース

データタイプ	インタフェース
AMOUNT	NullableAmountVariable
BINARY	NullableBinaryVariable
BOOL	NullableBooleanVariable
DATE	NullableDateVariable
DATETIME	NullableDateTimeVariable
FLOAT	NullableFloatVariable
NUMERIC	NullableNumericVariable
STRING	NullableStringVariable
TEXT	NullableTextVariable
TIME	NullableTimeVariable
任意のデータタイプ、変更可能	NullableAnyVariable

NullableAny と同じように、NullableAnyVariable インタフェースはこのタイプの変数が任意のタイプのデータを保持できることを表します。NullableAnyVariable 変数は最初に代入されたデータの型を以降保持できるよう設定されます。ここで

一旦設定された型は、`makeTypeUndefined()` が呼び出され、`NullableAnyVariable` 変数が他のデータ型変数を受け取れるようになるまで、インタラクショナルサーバによって関連付けられたままになります。

インタフェースについての詳しい説明は、アプリケーションデザイナーのメニューバーでヘルプ > Javadoc を選択して Unify NXJ フォームプロセッシング API を参照して下さい。

すでに説明されているように、フォームレイアウトのフィールドコントロールには暗黙的に宣言された NXJ 変数があり、名前はフィールドコントロールと同じです。これらの変数は `NullableField` インタフェースによって記述されるのでプロパティセットを持っています。NXJ プログラミング言語でこれらのプロパティを利用できるようにするために、フィールドは、以下のテーブルで記述される NXJ フィールドコントロールインタフェースの 1 つを実装して宣言されます。

表 3-4 フィールドコントロールのインタフェース

フィールドコントロールのタイプ	暗黙的に宣言するインタフェース
AMOUNT	<code>NullableAmountField</code>
BINARY	<code>NullableBinaryField</code>
BOOL	<code>NullableBooleanField</code>
DATE	<code>NullableDateField</code>
DATETIME	<code>NullableDateTimeField</code>
FLOAT	<code>NullableFloatField</code>
NUMERIC	<code>NullableNumericField</code>
STRING	<code>NullableStringField</code>
TEXT	<code>NullableTextField</code>
TIME	<code>NullableTimeField</code>

フィールドのプロパティや変数を参照するには、Java のメンバ構文を使用します。例えば、フィールド “Field1” に `stopForInput` プロパティを設定します。

```
Field1.stopForInput = false;
```

いくつかのプロパティは式になります。式のプロパティは、設定時ではなく使用時に評価されます。この点は Java アプリケーションと異なります。この仕様は、NXJ アプリケーションで最も大切なことです。なぜならプロパティは、それがアクセスするごとに意味がある値だからです。

---

NXJ プログラミング言語には次のような式のプロパティがあります。

- clearFindExp
- clearAddExp
- zoomReturnExpressions

前フォーム 操作が、ズームフォームから起きたと評価されます。[91 ページの「前フォームに移動」](#)を参照してください。

## clearFindExp

**clearFindExp** プロパティは、変数の最初の検索条件を指定します。ON CLEAR TO FIND イベントセクションか、または BEFORE FORM のような FIND 操作の前に実行される別のコードセクションでこのプロパティを設定します。

FIND 操作を使用する唯一のフィールドであるフォームのターゲットフィールドだけがこのプロパティを使用します。プロパティが評価されるとき、結果はフィールドコントロールに表示されます。ユーザは、表示された値を FIND 操作を始める前に変更することができます。

clear-to-find 操作が起こると clear-to-find 式は評価されます。[98 ページの「Clear-to-find 操作」](#)を参照してください。

**clearFindExp** プロパティは、互換性のあるデータタイプに評価される定数、または式に設定することができます。

**clearFindExp** プロパティが String 値に設定される場合、値は範囲、あるいはデータベースによってサポートされる“ワイルドカード”であることを示す特殊な文字を含むことができます。

以下の表は、チュートリアルアプリケーションの EMPLOYEE フォームの検索条件を定義する様々な **clearFindExp** の設定を示します。

表 3-5 *clearFindExp* プロパティの設定例

テスト	使用する式
特定の値	EMP_ID.clearFindExp = 1103 ; FIRST_NAME.clearFindExp = "Hugh" ;
値より小さい	EMP_ID.clearFindExp => 1103 ;
値より大きい	EMP_ID.clearFindExp = 1103 => ;

表 3-5 *clearFindExp* プロパティの設定例

テスト	使用する式
値以下	<code>EMP_ID.clearFindExp = 1103, =&gt; 1103 ;</code>
値以上	<code>EMP_ID.clearFindExp = 1103, 1103 =&gt; ;</code>
範囲内の値	<code>EMP_ID.clearFindExp = 1103 =&gt; 1105 ;</code>
一組の有効な値の中にある値	<code>EMP_ID.clearFindExp = 1101, 1103, 1105 ;</code> <code>EMP_FIRST_NAME.clearFindExp = "Hugh", "Lisa", "Jeff" ;</code>
値と等しくない値	<code>EMP_ID.clearFindExp = !1101;</code> <code>EMP_FIRST_NAME.clearFindExp = ! "Hugh";</code>
NULL と等しい値	<code>EMP_ID.clearFindExp = @;</code>
NULL ではない値	<code>EMP_ID.clearFindExp = ! @;</code>

## clearAddExp

以下の例では、変数の `clear-to-add` 式を設定しています。`clear-to-add` 式とは、以降の `add/update` 操作時に列で使用する値です。

例えば、`ROFIDA.EMPLOYEE` テーブルに `DATE_ADDED` という列があるとします。この列はバックグラウンドで使用され、`ROFIDA_EMPLOYEE` フォームには表示されません。以下の `NXJ` プログラミング言語文は、新規レコードの為に列を初期化します。

```
FORM ROFIDA_EMPLOYEE
{
    // Declares a Date variable to be associated with
    // the DATE_ADDED column.
    public multi_valued NullableDateVariable DATE_ADDED;

    BEFORE FORM
    {
        // Set the DATE_ADDED clearAddExp to the current date.
        // Whenever the user creates a new Employee record, this field
        // is set to this value.
        DATE_ADDED.clearAddExp = session.currentDate;
    }

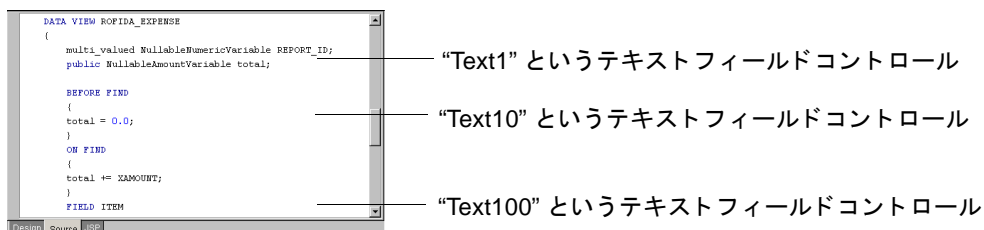
    // Remainder of the form script
}
}
```

フィールドや変数が複数値でも、プロパティのセットは1つしかないことに注意して下さい。

clear-to-add 操作が起きたときに評価されます。[105 ページの「Clear-to-add 操作」](#)を参照してください。

## フィールドとコントロールへのアクセス例

すでに説明されているように、NullableField 変数は NXJ プログラミング言語文のフィールド名によって参照されます。例えば、あるフォームに以下のような3つのテキストフィールドコントロールがあるとします。



各フィールドコントロールの Data Type プロパティは Numeric に設定されています。各3つのフィールドコントロールは、NullableNumericField タイプの NXJ 変数に宣言されるため、フォームスクリプトで利用できます。この簡単なスクリプトの用途は、ユーザが入力した2つの数値を加算することです。このような動作をする NXJ プログラミング言語文は、以下のとおりです。

```
FIELD Text1
{
  WHEN VALUE CHANGES
  {
    Text100 = Text1 + Text10;
  }
}
...
FIELD Text10
{
  WHEN VALUE CHANGES
  {
    Text100 = Text1 + Text10;
```

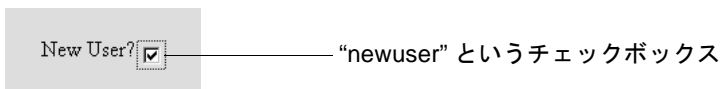
```
}  
}
```

フォームを初期化すると、フィールドコントロールを表す 3 つの変数がすべて null に設定されます。ユーザが **Text1** と **Text10** に値を入力すると、**Text100** 変数に合計が設定されます。

通常、次のコマンドがサーバに送られると、NXJ インタラクショナルサーバは、フィールドの値を一括変更してそれと同時に計算をします。ユーザがフィールドに入力したらすぐに合計を表示する必要があるので、このデフォルトの動作は適切ではありません。代わりに、**Text1** と **Text10** フィールドの Immediate プロパティを TRUE に設定し、どちらかのフィールドがフォーカスを失うとすぐにインタラクショナルサーバで値を計算させるようにします。

また、フィールドコントロールのデータタイプが 3 つとも NUMERIC に設定されていることを確認します。デフォルトのデータタイプ STRING のままの場合は、Text100 フィールドコントロールには式 “Text1 + Text10” の結果として、連結値 ‘10024’ が表示されます。

チェックボックスの NullableField 変数のカレント値にアクセスするには、論理式を使用します。チェックボックスのフィールドコントロールのデータタイプは boolean です。例えば、フォームに以下のようなチェックボックスがあります。



**newuser** 変数の値を使うと、フォームスクリプトに以下のような文を追加して、Welcome メッセージが表示できます。

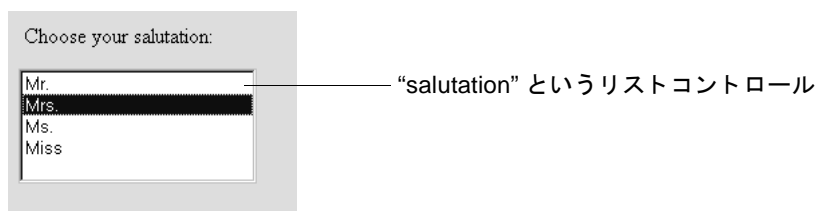
```
FIELD newuser  
{  
  AFTER FIELD  
  {  
    if ( newuser ) {  
      session.displayToMessageBox("Welcome New User!");  
    }  
  }  
}...
```



---

リストボックスやドロップダウンリストボックスのカレント値を参照するには、フィールドコントロール名を使います。これはテキストフィールドコントロール名の場合と同じです。関連する NullableField 変数の値は、カレントのリストボックスの対応値、つまりフォーカスがある Name の対応値です。どの Name エントリにもフォーカスがなければ、NullableField 変数の値は null です。

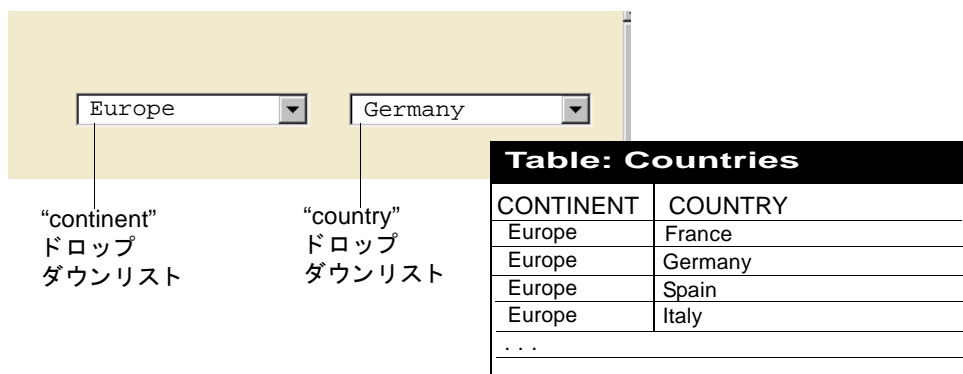
例えば、リストボックスのエントリを選択すると名前に用いる敬称が選択できます。



以下のスクリプトの NXJ プログラミング言語文では、姓と敬称を連結させて、メッセージの挨拶部分で使用する文字列を作成しています。

```
NullableString Greeting;cd o
...
Greeting = "Hello " + salutation + " " + LastName;
```

NXJ プログラミング言語を使用することで、リストボックスの項目を、別のリストボックスの値に基づいて変更することができます。例えば、フォームが2つのドロップダウンリストボックスを含み、フォームのターゲットテーブルがそれらの continent に country をマップするテーブルである、と仮定します。



Countries テーブルは、CONTINENT と COUNTRY の 2 つの列を持ちます。continent ドロップダウンリスト（フィールド名 “continent”）は、Query プロパティに対して以下の値を持ちます。

```
select distinct continent from companydata.Countries
```

フォームの初期化で、このリストボックスは、テーブルに定義された各 continent 名を含みます。

country ドロップダウンリスト（フィールド名 “country”）は、Query プロパティに対して以下の値を持ちます。

```
select country from companydata.Countries where continent = :continent
```

このクエリは、フォームの初期化中に実行します。[85 ページの「フォームの初期化」](#)を参照してください。continent リストボックスの現在の値が一致しない場合、リストボックスは、空の文字列 (“”) の行を表示します。

---

ユーザが continent ドロップダウンリストから別の continent を選択すると、NXJ インタラクショナルサーバは、その continent の country のみを示すために country ドロップダウンリストを再表示します。これを行う NXJ プログラミング言語文は、以下のようになります。

```
...
FIELD continent
{
    WHEN VALUE CHANGES
    {
        country.refetchOptions();
    }
}
```

非フィールドコントロールも、NXJ プログラミング言語を使ってアクセスすることができます。以下の例は、“redFlag” と名づけられたスタイルテキストコントロールの Visible プロパティにアクセスします。XREPORT フォームの経費レコードが 60 日以上承認されなかった場合にのみ、コントロールが表示されます。

```
...
ON FIND
{
    if ( APPROVAL.isNull() )
    {
        int daysOld = session.currentDate - INCURRED;
        if ( daysOld > 30 )
            redFlag.visible = false;
        if ( daysOld > 60 )
            redFlag.visible = true;
    }
}
```

---

## 外部リファレンス

他のフォームクラスの変数を参照することを *外部リファレンス* と言います。カレントフォームスタック上のフォームの変数を参照できます。つまり、カレントユーザセッションでアクティブなフォームの変数を参照できます。カレントフォームクラスのデータビューにある変数を参照することは、外部リファレンスではありませんが、他のフォーム上のデータビューの変数を参照することは、外部リファレンスになります。

外部リファレンスの構文は以下のとおりです。

```
<form-name>#<variable-reference>
```

変数が、データビューで宣言される場合、外部リファレンスは以下の構文を持ちます。

```
<form-name>#<data-view-name>.<variable-reference>
```

表 3-6 外部リファレンスの構文要素

要素	説明
<container-name>	変数が宣言されているフォーム名、データビュー名、ボックス名あるいはタブ名
<variable-reference>	参照されるコンテナ内の変数名

以下の NXJ プログラミング言語文を使うと、EmployeeMgt フォームから EMP\_ID 変数値が利用できます。

```
BEFORE FIND
{
  EMP_ID.searchRanges = EmployeeMgt#EMP_ID;
}
```

---

以下の NXJ プログラミング言語文を使うと、EmployeeMgt フォームの EXPENSE\_REQUEST データビューから “total” 変数が利用できます。

```
BEFORE FORM
{
  if ( XReport#ROFIDA_EXPENSE.total > 1000.00 )
  {
    session.displayToMessageBox("Total exceeds $1000.00!");
  }
...

```

外部リファレンスは、埋め込み SQL 文で使用することもできます。例えば、以下の NXJ プログラミング言語文を使うと、“Entry” という名前の親フォームから EMP\_ID 変数が利用できます。

```
NullableAmount amt;
EXEC SQL SELECT LIMIT FROM ROFIDA.LIMIT
  WHERE EMP_ID = :Entry#EMP_ID
  INTO workEMP_ID;
```



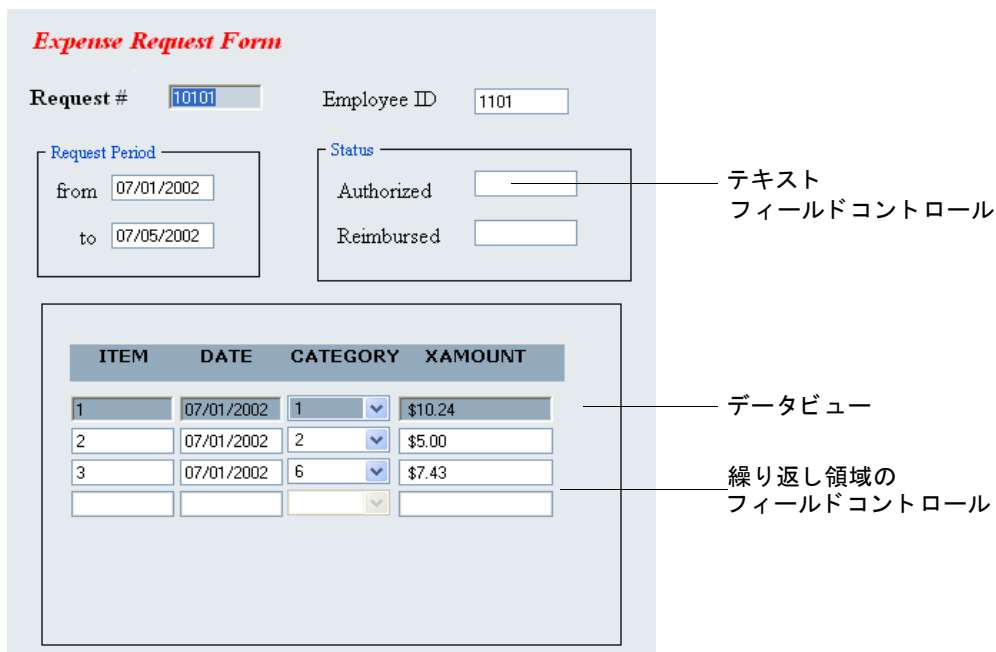
# スクリプトの構造と構文

# 4

この章では、NXJ プログラミング言語フォームスクリプトの構造と構文を説明します。

## スクリプトの構造

フォームスクリプトのハイレベルな構造は、フォームに追加されたフィールドコントロールとデータビューに基づいてアプリケーションデザイナーが、作成、管理します。例えば、“XReport” というフォームを作成して以下の項目を含めたとします。



The screenshot shows an "Expense Request Form" with the following elements:

- Request #**: Text input field containing "10101".
- Employee ID**: Text input field containing "1101".
- Request Period**: A group box containing two text input fields: "from" (07/01/2002) and "to" (07/05/2002).
- Status**: A group box containing two text input fields: "Authorized" and "Reimbursed".
- Table**: A data view table with columns: ITEM, DATE, CATEGORY, XAMOUNT.

ITEM	DATE	CATEGORY	XAMOUNT
1	07/01/2002	1	\$10.24
2	07/01/2002	2	\$5.00
3	07/01/2002	6	\$7.43

Annotations on the right side of the form:

- Text: テキスト フィールドコントロール (Text Field Control) - points to the "Authorized" and "Reimbursed" input fields.
- Text: データビュー (Data View) - points to the table.
- Text: 繰り返し領域のフィールドコントロール (Field Control in Repeating Area) - points to the empty row in the table.

作成するフォームスクリプトに対応するテンプレートが作成されます。テンプレートには、フォームのフィールドコントロールとデータビューの各セクションと、フォーム自身のセクションがあります。

```
FORM XRequest
{
  FIELD REPORT_ID
  {
  }
  FIELD EMP_ID
  {
  }
  FIELD FROM_DATE
  {
  }
  FIELD TO_DATE
  {
  }
  FIELD AUTHORIZED
  {
  }
  FIELD REIMBURSED
  {
  }
  DATA VIEW ROFIDA_EXPENSE
  {
    FIELD CATEGORY
    {
    }
    FIELD ITEM
    {
    }
    FIELD INCURRED
    {
    }
    FIELD XAMOUNT
    {
    }
  }
}
```

FIELD セクション

FORM セクション

DATA VIEW セクション

---

フォームのボックスは実行時に操作できないので、フォームスクリプトにはこれらのセクションがありません。繰り返し領域も全般的には実行時に操作できません。代わりに、フォームスクリプトには、繰り返し領域中の個別のフィールドのイベントセクションがあります。フォームのラベルは、通常、実行時に操作できないので、フォームスクリプトにはこれらのオブジェクトのセクションがありません。

それぞれのイベントセクションは、Java プログラムと同じように中括弧 ( { } ) で区切られています。イベントセクションに NXJ プログラミング言語文を追加するときはこの中括弧の間に追加します。

各セクションの名前は、アプリケーションデザイナーで定義したフィールドコントロールやデータビューの名前に対応しています。例えば FIELD REPORT\_ID フィールドセクションは、フォーム左上の REPORT\_ID 数値型フィールドコントロール (「Request #」というラベル) を作成します。

セクションによっては、他のイベントセクションにネストされているものもあります。例えば、データビューのフィールドコントロールはすべて DATA VIEW セクションに入っていますが、これはこれらのフィールドコントロールがデータビューに属しているためです。

---

**注** - フォームスクリプトのハイレベルな構造は アプリケーションデザイナーで管理されており、フィールドコントロールの削除や名前の変更などによって、フォームが変更されると自動的に更新されます。

---

## スクリプトの構文

フォームスクリプトのイベントセクションには次のものがあり、タイプによって独自の構文があります。詳しくはこの後で説明します。

- FORM セクション
- DATA VIEW セクション
- FIELD セクション
- Contorol セクション

---

以下の章にある構文の表では、次のような表記規則を使用します。

表 4-1 構文の表記規則

表記規則	目的
<b>太字</b>	太字はキーワードを表します。文に表示されるテキスト文字です。
<要素>	山型の括弧は構文の要素を表します。
[<要素>]	角括弧はオプション要素を表します。
	縦線は要素をグループに分けるためのもので、1つのグループには1度しか使用できません。
(<要素>)	丸括弧は要素グループを表します。通常は各種の選択肢や繰り返すことのできる要素グループを表します。
*	アスタリスクは、上記のグループを任意の回数記述できることを示します。記述しなくてもかまいません。

## FORM セクション

基本的に、FORM セクションはフォームスクリプト全体です。フォームスクリプトの構文は以下のとおりです。

```
<package-identification>
<import-statements>
FORM <form-name>
  [ extends <form-parent-class> ]
{
  ( <NXJ-variable-declaration>
  | <command-declaration>
  | <form-event-declaration>
  | <field-declaration>
  | <data-view-declaration>
  | <method-declaration>
  ) *
}
```

表 4-2 FORM セクションの構文要素

要素	説明
<package-identification>	アプリケーションデザイナーでパッケージ修飾名のあるフォームを作成した場合は、Form を宣言する前にパッケージを宣言します。例えば “com/rofida/expense/EmployeeMgt” という名前でフォームを作成するときは、フォームスクリプトに以下のようなパッケージ 文を含めます。  package com.rofida.expense;
<import-statements>	フォームスクリプトによって必要とされる import 文です。フォームスクリプトは、Unify NXJ ファウンデーションクラス以外のクラスを使う場合は import 文が必要です。
<b>FORM</b> <form-name>	フォーム宣言文。フォーム作成時にフォームスクリプトに自動的に追加されます。
<NXJ-variable-declaration>	Java 単一値変数宣言、または複数値変数宣言です。 <a href="#">19 ページの「NXJ 変数」</a> を参照してください。
<form-parent-class>	親クラスにはかかわらず、名前は “sources/Classes” ディレクトリに関連付けられる。
<command-declaration>	<a href="#">46 ページの「開発者定義コマンド宣言」</a> を参照してください。
<form-event-declaration>	<a href="#">50 ページの「Form イベントセクション」</a> を参照してください。
<field-declaration>	<a href="#">60 ページの「FIELD セクション」</a> を参照してください。
<data-view-declaration>	<a href="#">53 ページの「DATA VIEW セクション」</a> を参照してください。
<method-declaration>	標準の Java 構文に準拠したメソッド宣言ですが、Unify NXJ ファウンデーションクラスと Unify NXJ 埋込み型 SQL が使用できます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

フォーム宣言は実行時に同じ名前の Java クラスに変わり、この名前によって Java パッケージの form クラスが特定されます。このクラスは Unify NXJ ファウンデーションクラス (`com.unify.NXJ.mgr.NXJForm`) を継承します。NXJForm ファウンデーションクラスもまたファウンデーションクラス (`com.unify.NXJ.mgr.NXJDataView`) を継承します。これは、フォームオブジェクト

---

は暗黙的なデータビューオブジェクトであるためです。暗黙的なデータビューには、フォームと同じ名前があります。フォーム上に置かれる他のデータビューは、作成時に名前が付きます。

フォームレイアウトに置かれるフィールドコントロールとデータビューはすべて、フォームクラスの内部クラスになります。

NXJForm ファウンデーションクラスと NXJDataView ファウンデーションクラスにはプロパティとメソッドがあり、実行時にアプリケーションの情報を表示したり動作を変更したりできます。これらのプロパティとメソッドについては、アプリケーションデザイナのメニューバーで、ヘルプ > Javadoc を選択して Unify NXJ フォームプロセッシング API を参照して下さい。

## 開発者定義コマンド宣言

開発者定義コマンドとは、開発者が NXJ プログラミング言語文を使って定義するコマンドです。これらのコマンドは、Unify NXJ で提供されるコマンド（例：FIND）と区別するために、開発者定義コマンドと呼ばれています。通常、コマンドはフォームやファンクションキーのボタンにマッピングされます。ユーザが実行時にボタンやファンクションキーをクリックすると、コマンドが実行されます。また、queueCommand メソッドを使ってフォームスクリプトからコマンドを呼び出すこともできます。

コマンド宣言の構文は以下のとおりです。

```
COMMAND <command-name> [( <parameter-declaration parameter> ) ]
  [ throws <Java-exception-list> ]

  [ PROPERTIES
    {
      ( activeMode = ( ActiveMode_FIND | ActiveMode_AUD |
                    ActiveMode_BOTH ) ;
      | enabled = ( true | false ) ;
      | editAction = ( EditAction_ACCEPT | EditAction_RESET |
                    EditAction_SUSPEND) ;
    }*
  ] ]
  {
  <NXJ-statements>
  }
```

表 4-3 開発者定義コマンドの構文要素

要素	説明
<b>COMMAND</b> <command-name>	コマンド宣言の文です。コマンド名には Unify NXJ のコマンド名は使用できません。コマンド名のリストについては、Javadoc の NXJCommand ファウンデーションクラスを参照してください。Javadoc を開くには、アプリケーションデザイナのメニューバーで、ヘルプ >Javadoc を選択します。
[ <parameter-declaration parameter> ]	パラメータ宣言文です。parameter-declaration は、オブジェクトかオブジェクトのサブクラスでなければなりません。
<b>throws</b> <exception-list>	Java メソッドでスローされる例外を宣言する標準的な例外リストです。
<b>activeMode</b> = (ActiveMode_FIND   ActiveMode_AUD   ActiveMode_BOTH )	コマンドがサポートされるデータビューのモードです。activeMode のデフォルトは both です。
<b>enabled</b> = (true   false)	コマンドが利用できるかどうかを表すフラグです。enabled のデフォルトは true です。
<b>editAction</b> = ( EditAction_ACCEPT   EditAction_RESET   EditAction_SUSPEND)	実行時にコマンドを実行する前に、データ入力を受け付けられるかどうかを指定します。次のようなオプションがあります。 _ACCEPT：コマンドを実行する前にインタラクティブサーバーで入力データを受け付け、処理します。 _RESET：インタラクティブサーバーは入力データを無視し、データが入力される前の状態に戻してから、コマンドを実行します。 _SUSEPEND：インタラクティブサーバーは、入力操作を中断したままの状態でもコマンドを実行します。 editAction のデフォルトは EditAction_ACCEPT です。
<NXJ-statements>	コマンドが呼び出されたときに実行する NXJ プログラミング言語文の本文です。

コマンド宣言は実行時に同じ名前の Java クラスに変わり、この名前によってフォームクラスのコマンドが特定されます。このクラスは Unify NXJ ファウンデーションクラス (com.unify.NXJ.mgr.NXJCommand) を継承します。このクラスで利用できるプロパティとメソッドについては、NXJ フォームプロセッシング API を参照してください。

---

以下の例では、“XReport”というデータビュー用の、“authorize”というコマンドを定義する方法が示されています。

FORM X

```
{
  COMMAND authorize
  {
    AUTHORIZED = session.currentDate;
    session.queueCommand("ADD_UPDATE", "XReport");
  }
}
```

このコマンドでは、まず AUTHORIZED フィールドコントロールの値を現在の日付に設定し、次にデータベースにレコードを格納（更新）して、経費報告書が承認されています。authorize コマンドは、以下のようにフォームのボタンにマップさせることができます。フォームにボタンコントロールを置いた後、Command プロパティのドロップダウンリストから“authorize”を選択することで、ボタンの Command プロパティに、authorize コマンドを設定します。

以下の例では、Numeric のパラメータを使って、パラメータの値を表示する単純なコマンドを示しています。

FORM Y

```
{
  COMMAND showValue (Object inputValue)
  {
    if (inputValue != null)
    {
      session.messageBoxPrompt("The value is: " +inputValue);
    }
  }
}
```

showValue コマンドが呼び出されるときは、必ずパラメータが含まれていなければなりません。例えば、フォームスクリプトの他のセクションからコマンドを呼び出すには以下の構文を使います。

```
id=24;
session.queueCommand(showValue,id);
```

---

指定されたパラメータのデータタイプは、コマンド定義で宣言されたパラメータのデータタイプと互換のあるものでなければなりません。showValue コマンドが呼び出されてパラメータが指定されていない場合は、null の値が渡されます。

この例では、特定のレコードに移動するコマンドを示しています。

```
package Forms;

FORM Form2
{
NXJLeafNode mymenu;

COMMAND goToRecord(String recNumber)
{
    int recNo = Integer.parseInt(recNumber);
    Form2.positionToRecord(recNo);
}

BEFORE APPLICATION
{
    mymenu = new NXJLeafNode();
    mymenu.title = "Go To Record";
    mymenu.command = "goToRecord";
    mymenu.commandParameter = "1";
    mymenu.toolTip = "Please enter a record number";
    Form2.mainTree.treeRoot.add(mymenu);
    recNumber = "1";
}
NXJ TREE mainTree
{
}
FIELD recNumber
{
    WHEN VALUE CHANGES
    {
        mymenu.commandParameter = recNumber.toString();
    }
}
FIELD COUNTRY_ISO_CODE
{
}
FIELD CITY_NAME
{
}
FIELD CITY_ID
```

---

```
{
}
}
```

パラメータは、複雑な Java オブジェクトを受け渡すのにも使用されます。

```
// Passing a complex Java object
//Customer.java

public class Customer {
    public String name;
    public String address;
    public String city;
}

// using the object
Customer myCustomer = new Customer();
BEFORE FORM
{
    myCustomer.name = "UNIFY";
    myCustomer.address = "my home";
    myCustomer.city = "Sacramento";
    session.queueCommand(addressCheck,myCustomer);
}
COMMAND addressCheck(Customer savCus)
{
    session.displayToMessageBox(savCus.name + " - " + savCus.address +
    ", " + savCus.city);
}
```

## Form イベントセクション

「概要」の章で説明したように、イベントセクションはイベント発生時にだけ実行されるビジネスルールのセットです。FORM セクションで使用できるイベントセクションは、フォームや選択セットの処理に関連付けられています。

フォーム固有のイベントセクションの構文は以下のとおりです。

```
<event-section-name>
[ throws <Java-exception-list> ]
{
```

```

<NXJ-statements>
}

```

表 4-4 イベント セクションの構文要素

要素	説明
<event-section-name>	<a href="#">52 ページの表 4-5</a> と <a href="#">57 ページの表 4-8</a> にあるイベントセクション名のどれかを使用します。
<Java-exception-list>	Java メソッドでスローされる例外を宣言する標準的な例外リストです。省略した場合、標準的な Java メソッド宣言とは異なり、イベントセクションでは任意の例外をスローできます (“java.lang.Exception” をスローするように宣言されている場合と同様)。  イベントセクションが “throws” 節を持つ場合、指定した例外のみスローすることができます。どちらの場合も、イベントセクションからスローされる例外は Unify NXJ インタラクションサーバで扱われます。
<NXJ-statements>	イベント発生時に実行する NXJ プログラミング言語文の本文です。文には埋込み型 SQL を含めることができます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

イベントセクションは、それが発行されるクラスのメソッドになります。例えば、AFTER FIND イベントのイベントセクションがあれば、このイベントセクションは Make 操作中に afterFind() という Java メソッドに変換されます。イベントセクションが空の場合は、ファウンデーションクラスから継承されたメソッドになります。

次の例では、フォームの任意の場所で使用できる変数を BEFORE FORM イベントセクションを使って初期化する方法が示されています。この場合、XReport フォームの FROM\_DATE フィールドコントロールと TO\_DATE フィールドコントロールの clear-to-add 式が現在の日付で初期化されています。

```

FORM XReport
{
  BEFORE FORM
  {
    FROM_DATE.clearAddExp = session.currentDate;
    TO_DATE.clearAddExp = session.currentDate + 7;
  }
  ...
}

```

---

以下の表には、フォームイベントセクション名と実行時期がそれぞれ示されています。

表 4-5 フォームイベントセクション (1 / 2)

イベントセクション名	実行時期
BEFORE APPLICATION	エントリフォームの初期化で、最初のステップとして実行します。このセクションでは、ユーザ認証、セキュリティ制約の実施、またはアプリケーション全体で使用される変数の初期化などを実行します。 このイベントセクションを実行するタイミングについては、 <a href="#">82 ページの「アプリケーションの開始」</a> を参照してください。
BEFORE FORM	フォーム初期化中に実行されます。このセクションで、フォームの各インスタンスを初期化します。共通のフォーム初期化作業は、変数値の設定、clear-to-add および clear-to-find 式の設定が含まれます。このイベントセクションを実行するタイミングについては、 <a href="#">85 ページの「フォームの初期化」</a> を参照してください。
ON NEXT FORM	NEXT_FORM コマンドが実行される時に実行されます。このセクションでは、すべての必須項目が入力されているかどうかの確認、フォームの状態の確認などを実行します。 このイベントセクションを実行するタイミングについては、 <a href="#">88 ページの「次フォームに移動」</a> を参照してください。
evaluateNextFormCondition	NEXT_FORM 操作中に実行されます。このメソッドは、次フォームリスト中の各条件付きエントリに対して実行されます。次フォームの選択メニューに、条件付フォームを含めるかどうかを判断するために、このメソッドは true か false を返さなければなりません。 このイベントセクションを実行するタイミングについては、 <a href="#">88 ページの「次フォームに移動」</a> を参照してください。
AFTER FORM RETURN	PREVIOUS_FORM または CANCEL_FORM 操作中に実行されます。 このイベントセクションを実行するタイミングについては、 <a href="#">91 ページの「前フォームに移動」</a> と <a href="#">95 ページの「ズームフォームの取り消し」</a> を参照してください。

表 4-5 フォームイベントセクション (2 / 2)

イベントセクション名	実行時期
AFTER ZOOM	ズームフォームから始められた PREVIOUS_FORM 操作中に実行されます。このセクションでは、すべての必須項目が入力されているかどうかの確認、フォームの状態の確認、zoomReturnExpressions プロパティが正しく設定されているかどうかの確認などを実行します。 このイベントセクションを実行するタイミングについては、 <a href="#">91 ページの「前フォームに移動」</a> を参照してください。
ON PREVIOUS FORM	PREVIOUS_FORM 操作中に実行されます。このセクションでは、すべての必須項目が入力されているかどうかの確認、フォームの状態の確認などを実行します。 このイベントセクションを実行するタイミングについては、 <a href="#">91 ページの「前フォームに移動」</a> を参照してください。
ON EXIT	EXIT 操作中に実行されます。このセクションでは、アプリケーションを終了する前に必要な、カレントフォームに関連した処理のクリーンアップを実行します。 このイベントセクションを実行するタイミングについては、 <a href="#">83 ページの「アプリケーションの終了」</a> を参照してください。
AFTER APPLICATION	アプリケーションを終了する時に実行されます。このセクションでは、カウントの更新、メッセージの表示、統計の概要の表示など、アプリケーションの終了時に行う操作を実行します。 このイベントセクションを実行するタイミングについては、 <a href="#">83 ページの「アプリケーションの終了」</a> を参照してください。

## DATA VIEW セクション

データビューは、独自の選択セットを持つフォームの領域です。多くの場合、データビューはフォームとは別のターゲットテーブルに関連付けられています。フォーム自体が、本質的にはデータビューです。

フォームにはデータビューを追加することもできます。その場合は、フォームのレイアウトにデータビューを配置して作成します。データビューは、フォームスクリプトの DATA VIEW セクションで定義されます。フォームには複数のデータビューがありますが、データビュー同士をネストさせることはできません。

---

DATA VIEW セクションの構文は以下のとおりです。

```
DATA VIEW <data-view-name>
  [ extends <data-view-parent-class> ]
{
  ( <NXJ-variable-declaration>
  | <data-view-event-section>
  | <field-declaration>
  | <method-declaration>
  ) *
}
```

表 4-6 DATA VIEW セクションの構文要素

要素	説明
DATA VIEW <data-view-name>	データビュー宣言文。データビュー作成時にフォームスクリプトに自動的に追加されます。
<data-view-parent-class>	親クラスにはかかわらず、名前は“sources/Classes”ディレクトリに関連付けられる。
<NXJ-variable-declaration>	Java 単一値変数宣言、または複数値変数宣言です。 <a href="#">19 ページの「NXJ 変数」</a> を参照してください。
<data-view-event-section>	<a href="#">55 ページの「Data View イベントセクション」</a> を参照してください。
<field-declaration>	<a href="#">60 ページの「FIELD セクション」</a> を参照してください。
<method-declaration>	標準の Java 構文に準拠するメソッド宣言で、メソッド本文に NXJ プログラミング言語文を含めることができます。文には埋込み型 SQL を含めることができます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

DATA VIEW 宣言は同じ名前の Java 内部クラスとインスタンス変数に変わり、この名前によってフォームクラスのデータビューが特定されます。生成された Java クラスは Unify NXJ ファウンデーションクラス (**com.unify.NXJ.mgr.NXJDataView**) を継承します。データビューで宣言されたフィールドは、そのデータビュークラス内の内部クラスになります。このクラスで利用できるプロパティとメソッドについては、アプリケーションデザイナのメニューバーで、ヘルプ > Javadoc を選択して Unify NXJ フォームプロセッシング API を参照して下さい。

---

データビューにイベントセクションやメソッド定義を追加する場合は、フォームの中括弧 ( { } ) ではなくデータビューの中括弧内 ( { } ) に置きます。

```
FORM XReport
{
  ON CLEAR TO ADD ----- フォームの
  {                               ON CLEAR TO ADD
    REPORT_ID = getNextNum("XREPORT");
  }
  DATA VIEW ROFIDA_EXPENSE
  {
    ON CLEAR TO ADD ----- データビューの
    {                               ON CLEAR TO ADD
      ITEM = recordCount;
    }
  }
}
...
```

このブロックの外にあるフィールドやイベントセクションは、フォームに属するものと見なされます。これは重要なポイントで、データビューのイベントセクションも正規のフォームのイベントセクションなので、データビューのイベントセクションを間違った場所に置くとコンパイラで検出できないことがあります。

## Data View イベントセクション

データビューのイベントセクションは、選択セットのレコードの処理に関連付けられています。

データビューイベントセクション要素の構文は、以下のとおりです。

```
<event-section-name>
[ throws <Java-exception-list> ]
{
  <NXJ-statements>
}
```

表 4-7 Data View イベントセクションの構文要素

要素	説明
<event-section-name>	この次に示す <a href="#">表 4-8</a> のイベントセクション名のどれかを使用します。
<Java-exception-list>	Java メソッドでスローされる例外を宣言する標準的な例外リストです。省略した場合、標準的な Java メソッド宣言とは異なり、イベントセクションでは任意の例外をスローできます (“java.lang.Exception” をスローするように宣言されている場合と同様)。  イベントセクションが “throws” 節を持つ場合は、指定した例外のみスローすることができます。どちらの場合も、イベントセクションからスローされる例外は Unify NXJ インタラクションサーバで扱われます。
<NXJ-statements>	イベント発生時に実行する NXJ プログラミング言語文の本文です。文には埋込み型 SQL を含めることができます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

以下の例は、ON CLEAR TO ADD イベントセクションが、レコードが追加される毎にビジネスルール (recordCount) を起動する方法を示しています。

```
DATA VIEW ROFIDA_EXPENSE
{
  ON CLEAR TO ADD
  {
    ITEM = recordCount;
  }
  ...
}
```

---

以下の表は、データビューイベントセクション名と、それがいつ実行されるかを説明しています。

表 4-8 データビュー イベントセクション (1 / 3)

イベントセクション名	実行時期
ON CLEAR TO FIND	CLEAR_TO_ADD コマンドが発行される時に実行され ます。検索条件を設定するために、このセクションを使う ことができます。 このイベントセクションを実行するタイミングについ ては、 <a href="#">98 ページの「Clear-to-find 操作」</a> を参照してくださ い。
BEFORE FIND	FIND 操作の最初のステップとして実行されます。ユー ザが指定した検索条件を強化するために、このセクショ ンを使うことができます。 このイベントセクションを実行するタイミングについ ては、 <a href="#">96 ページの「レコードの検索」</a> を参照してくださ い。
ON FIND	検索条件に一致した各ターゲットテーブル行に対して FIND 操作中に実行されます。各レコード単位で処理を 実行したり、選択セットに対象行を含めないようにした りするために、このセクションを使用することができま す。 このイベントセクションを実行するタイミングについ ては、 <a href="#">96 ページの「レコードの検索」</a> を参照してくださ い。
AFTER FIND	FIND 操作が完了した後で実行されます。検索の状態の 確認、合計の更新、有効性のチェックを行うために、こ のセクションを使用することができます。 このイベントセクションを実行するタイミングについ ては、 <a href="#">96 ページの「レコードの検索」</a> を参照してくださ い。
ON CLEAR TO ADD	CLEAR_TO_ADD 操作が実行された後に実行されます。 変数のデフォルト値をセットするためにこのセクショ ンを使用することができます。 このイベントセクションを実行するタイミングについ ては、 <a href="#">105 ページの「Clear-to-add 操作」</a> を参照してくださ い。

---

表 4-8 データビュー イベントセクション (2 / 3)

イベントセクション名	実行時期
BEFORE ADD	<p>ADD_UPDATE コマンドが、新規レコードに対して発行される時に実行されます。新規レコードの値の確認、表示されないターゲット値の計算、検査記録の作成などに、このセクションを使用することができます。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">102 ページの「レコードの追加と更新」</a>を参照してください。</p>
AFTER ADD	<p>ADD 操作が完了した後に実行されます。追加操作の状態を確認、追加された行数のカウンタを増加したり、合計を更新するために、このセクションを使用することができます。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">102 ページの「レコードの追加と更新」</a>を参照してください。</p>
BEFORE UPDATE	<p>ADD_UPDATE コマンドが、既存のレコードに対して発行される時に実行されます。更新した値の確認、表示されないターゲット値の計算、検査記録の作成、追加のセキュリティ制約の提供や、合計を初期化するために、このセクションを使用することができます。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">102 ページの「レコードの追加と更新」</a>を参照してください。</p>
AFTER UPDATE	<p>UPDATE 操作が完了した後に実行されます。更新操作の状態を確認、更新された行数のカウンタを増加したり、合計を更新するために、このセクションを使用することができます。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">102 ページの「レコードの追加と更新」</a>を参照してください。</p>
BEFORE DELETE	<p>DELETE コマンドが発行される時に実行されます。検査記録の作成、追加のセキュリティ制約の提供や、合計を初期化するために、このセクションを使用することができます。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">107 ページの「レコードの削除」</a>を参照してください。</p>

表 4-8 データビュー イベントセクション (3 / 3)

イベントセクション名	実行時期
AFTER DELETE	<p>DELETE 操作が完了した後に実行されます。削除操作の状態の確認、削除された行数のカウンタを増加したり、合計を更新するために、このセクションを使用することができます。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">107 ページの「レコードの削除」</a>を参照してください。</p>
ON NEXT RECORD	<p>NEXT_RECORD、NEXT_SET、LAST_RECORD コマンドが発行された時に実行されます。但し、この表の下にある警告を参照してください。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">110 ページの「次レコード に移動」</a>を参照してください。</p>
ON PREVIOUS RECORD	<p>PREVIOUS_RECORD、PREVIOUS_SET、FIRST_RECORD コマンドが発行された時に実行されません。但し、この表の下にある警告を参照してください。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">112 ページの「前レコードに移動」</a>を参照してください。</p>
BEFORE RECORD	<p>新規レコードがカレントになる時に実行されます。</p> <p>このイベントセクションを実行するタイミングについては、<a href="#">110 ページの「次レコード に移動」</a>と <a href="#">112 ページの「前レコードに移動」</a>を参照してください。</p>

**警告** – Batch Record Changes プロパティが false の場合は、ON NEXTRECORD と ON PREVIOUS RECORD イベントセクションは対応するイベント（次レコード操作、前レコード操作）が起きたときでも、実行されません。これらのイベントセクションが実行されるのは、次の 2 つのケースだけです。(1) ユーザがカレント行を変更した。(2) データビュースクリプトのセクションに BEFORE RECORD イベントセクションが存在する。

---

## FIELD セクション

FIELD セクションは、フォーム上の任意のフィールドコントロールのイベントセクションを指定するために使用します。FIELD セクションの構文は、以下のとおりです。

```
FIELD <field-name>
{
  ( <Java-variable-declaration>
  | <field-event-section>
  | <method-declaration>
  )*
}
```

表 4-9 FIELD セクションの構文要素

要素	説明
<b>FIELD</b> <field-name>	フィールド宣言文。フィールドコントロール作成時にフォームスクリプトに自動的に追加されます。
<data-view-parent-class>	親クラスにはかかわらず、名前は“sources/Classes”ディレクトリに関連付けられる。
<Java-variable-declaration>	Java 変数宣言。
<field-event-section>	<a href="#">61 ページの「Field イベントセクション」</a> を参照してください。
<method-declaration>	標準の Java 構文に準拠するメソッド宣言で、メソッド本文に NXJ プログラミング言語文を含めることができます。文には埋込み型 SQL を含めることができます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

FIELD 宣言は同じ名前の Java クラスとインスタンス変数に変わり、この名前によってフォームやデータビューのフィールドクラスが特定されます。このクラスは Unify NXJ ファウンデーションインタフェース (`com.unify.NXJ.mgr.datatypes.NullableField`) を継承します。

---

## Field イベントセクション

FIELD イベントセクションの構文は、以下のとおりです。

```
<event-section-name>  
  [ throws <Java-exception-list> ]  
{  
  <NXJ-statements>  
}
```

表 4-10 FIELD イベントセクションの構文要素

要素	説明
<event-section-name>	この次に示す <a href="#">表 4-11</a> のイベントセクション名のどれかを使用します。
<Java-exception-list>	Java メソッドでスローされる例外を宣言する標準的な例外リストです。省略した場合、標準的な Java メソッド宣言とは異なり、イベントセクションでは任意の例外をスローできます (“java.lang.Exception” をスローするように宣言されている場合と同様)。  イベントセクションが “throws” 節を持つ場合、指定した例外のみスローすることができます。どちらの場合も、イベントセクションからスローされる例外は NXJ インタラクションサーバで扱われます。
<NXJ-statements>	イベント発生時に実行する NXJ プログラミング言語文の本文です。文には埋込み型 SQL を含めることができます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

---

イベントセクションは、それが発行されるクラスのメソッドになります。例えば、BEFORE FIELD イベントのイベントセクションがあれば、このイベントセクションは Make 操作中に beforeField() という Java メソッドに変換されます。

表 4-11 フィールドイベントセクション

イベントセクション名	実行時期
INIT FIELD	フォームが初期化される時に実行されます。フィールドの初期化が行えるセクションです。 このイベントセクションを実行するタイミングについては、 <a href="#">85 ページの「フォームの初期化」</a> を参照してください。
BEFORE FIELD	フィールドがカレントになる時に実行されます（データビューが AUD モードの場合）。ローカルフィールドの初期化が行えるセクションです。 このイベントセクションを実行するタイミングについては、 <a href="#">114 ページの「フィールド位置」</a> を参照してください。
ON DATA ACCEPT	ユーザによる新規入力の値を受け入れるために実行されます。ただし、FIND モードではユーザは、検索条件を入力するため、AUD モードでのみ実行されます。ユーザの入力を検証して、その受け入れや、拒否を行うことができるセクションです。 このイベントセクションを実行するタイミングについては、 <a href="#">120 ページの「ユーザ入力の処理」</a> を参照してください。
AFTER FIELD	フォーカスがフィールドから離れて移動する時に実行されます（データビューが AUD モードの場合）。 このイベントセクションを実行するタイミングについては、 <a href="#">114 ページの「フィールド位置」</a> を参照してください。
WHEN VALUE CHANGES	フィールド変数の値が変わるときに実行されます。 このイベントセクションを実行するタイミングについては、 <a href="#">120 ページの「ユーザ入力の処理」</a> を参照してください。

---

以下の例では、ON DATA ACCEPT イベントセクションを使って、指定した制限に対するユーザ入力を検証しています。

```
...
FIELD XAMOUNT
{
  ON DATA ACCEPT
  throws java.sql.SQLException,
  com.unify.NXJ.mgr.datatypes.NXJNullValueException,
  com.unify.NXJ.mgr.dataConnection.NXJDataConnectionException
  {
    NullableAmount limit = null;
    EXEC SQL SELECT LIMIT FROM ROFIDA.LIMIT
      WHERE EMP_ID = :EMP_ID AND CAT_ID = :CATEGORY
      INTO limit;
    if ( (! ( limit.isNull() ) ) && ( XAMOUNT > limit ) )
    {
      session.displayToMessageBox(
        "Amount exceeds limit for this employee.");
      rejectOperation();
    }
  }
}
```

## Control セクション

Control セクションは、ファイル選択コントロールのイベント処理を変更するために使用されます。Control セクションは、イベント処理動作のないコントロールを宣言するために使用することもできます。レイアウトに実在するコントロールはすべて、コントロールとして宣言する必要があります。これらの宣言型のコントロールセクションが必要になります。次のコントロールは宣言型の Control セクションを持つことができます。

- Boxes
- Buttons and image buttons
- Components
- Dynamic text
- Images
- Inline frames
- JavaScript
- Labels
- Lines

---

Links  
Navigation bar  
Styled text  
Tab set

ファイル選択コントロール以外のコントロールはイベントセクションを持つことはできないので、データビューおよびフィールドとは異なり、これらの宣言はフォームスクリプトに自動的に追加されません。コントロールの1つにアクセスするには、コードの名前でコントロールを参照します。例えば、

```
label1.foregroundColor = `red`;
```

Event-Handling コントロールの構文は、以下のとおりです。

```
<control-type> <control-name>  
{  
  ( <Java-variable-declaration>  
    | <control-event-section>  
    | <method-declaration>  
  )*  
}
```

単一の宣言型のコントロールセクションの構文は、以下のとおりです。

```
(<declarative-control-type> <control-name> ;
```

表 4-12 Control セクションの構文要素

要素	説明
<control-type>	コントロールのタイプ。以下のものでなければなりません。 NXJ FILE CHOOSER
<control-parent-class>	親クラスにはかかわらず、名前は “sources/Classes” ディレクトリに関連付けられる。 ボックス、タブ、ボタン、ダイナミックテキスト、イメージボタン、ラベル、スタイルテキストボックス、線、リンクコントロールだけが親クラスを持つことができます。

表 4-12 Control セクションの構文要素

要素	説明
<declarative-control-type>	単一の宣言型のコントロールのコントロールタイプ。以下の内の1つでなければなりません。 BOX NXJ BUTTON NXJ DYNAMIC TEXT NXJ IFRAME NXJ IMAGE NXJ IMAGE BUTTON NXJ JAVASCRIPT NXJ LABEL NXJ LINE NXJ LINK NXJ NAVIGATION BAR NXJ STYLED TEXT BOX TAB SET
<control-name>	コントロール名。フォームのコントロール名に一致しなければなりません。
<Java-variable-declaration>	Java 変数宣言。
<control-event-section>	次の <a href="#">65 ページの「Control イベントセクション」</a> を参照してください。
<method-declaration>	標準の Java 構文に準拠したメソッド宣言ですが、メソッド本文に NXJ プログラミング言語文を含めることができます。文には埋込み型 SQL を含めることができます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

## Control イベントセクション

Control イベントセクションの構文は、以下のとおりです。

```
<event-section-name>
[throws <Java-exception-list>]
{
  <NXJ-statements>
}
```

表 4-13 Control イベントセクションの構文要素

要素	説明
<event-section-name>	ON UPLOAD または WHEN UPLOAD COMPLETES
<NXJ-statements>	イベント発生時に実行する NXJ プログラミング言語文の本文です。文には埋込み型 SQL を含めることができます。 <a href="#">68 ページの「SQL 文の埋め込み」</a> を参照してください。

ファイルのアップロードに関する詳細は、『テキストデータとバイナリデータの使い方』を参照してください。

## 推奨するイベントセクションの順

FORM、DATA VIEW、FIELD セクションに追加するイベントセクションは任意の順に使用できます。チーム開発を行なう場合、これらのイベントセクションの順序を決めておけば、チームのメンバ全員がスクリプトを合理的に使用できて便利です。以下に、インタラクションサーバがイベントセクションを実行する順序とほぼ同じイベントセクションの順序を示します。

### フォームレベル イベントセクション

BEFORE APPLICATION  
BEFORE FORM  
ON CLEAR TO FIND  
BEFORE FIND  
ON FIND  
AFTER FIND  
ON CLEAR TO ADD  
BEFORE ADD  
AFTER ADD  
BEFORE UPDATE  
AFTER UPDATE

---

BEFORE DELETE  
AFTER DELETE  
ON NEXT RECORD  
ON PREVIOUS RECORD  
BEFORE RECORD  
ON NEXT FORM  
AFTER FORM RETURN  
AFTER ZOOM  
ON PREVIOUS FORM  
ON EXIT  
AFTER APPLICATION

データビュー イベントセクション

ON CLEAR TO FIND  
BEFORE FIND  
ON FIND  
AFTER FIND  
ON CLEAR TO ADD  
BEFORE ADD  
AFTER ADD  
BEFORE UPDATE  
AFTER UPDATE  
BEFORE DELETE  
AFTER DELETE  
ON NEXT RECORD  
ON PREVIOUS RECORD  
BEFORE RECORD

---

## フィールドレベル イベントセクション

INIT FIELD  
BEFORE FIELD  
ON DATA ACCEPT  
AFTER FIELD  
WHEN VALUE CHANGES

## ファイル選択 イベントセクション

ON UPLOAD  
WHEN UPLOAD COMPLETE

## SQL 文の埋め込み

このセクションでは、フォームスクリプトの埋め込み SQL 文の構文について説明します。SQL 文は <NXJ statements> や <method declaration> であれば、どのスクリプト要素にも埋め込むことができます。

埋め込み SQL 文の構文には、以下の 3 つのカテゴリがあります。

- シンプル埋め込み SQL、例えば UPDATE、DELETE、INSERT
- SELECT 文
- ロック文

Oracle、DataServer、Infomix データベースを使用している場合、NXJ アプリケーションからストアドプロシージャを呼び出して準備し、実行するために埋込み型 SQL を利用することができます。詳細については、『ストアドプロシージャの呼び出し方』を参照してください。

## シンプル埋め込み SQL 文

多くの場合、埋め込み SQL 文では、以下のような簡単な SQL 文構文が使用されます。

---

```
EXEC SQL [ USING CONNECTION <connection-name> ]  
<SQL-statement> ;
```

表 4-14 埋め込み SQL の構文要素

要素	説明
USING CONNECTION <connection-name> <SQL-statement>	SQL 文が実行されるコネクションを識別するコネクション名です。次の「 <a href="#">コネクション</a> 」を参照してください。 コネクションに関連付けられたデータベースと JDBC ドライバがサポートする SQL 文です。文の終了時にはセミコロンが必要です。 SQL 文には動的パラメータが使用できます。 <a href="#">70 ページの「動的パラメータ」</a> を参照してください。

## コネクション

コネクション名には、プロジェクトで定義したコネクションを使用します。コネクション名は識別子として指定します。以下はその例です。

```
USING CONNECTION tutorial
```

USING CONNECTION の節が省略されると、以下のように、文はデフォルトのコネクションで実行されます。

- 埋め込み SQL 文がフォームレベル イベントセクション（例：ON NEXT FORM）にある場合は、フォームのコネクションを使用。
- 埋め込み SQL 文がデータビュー イベントセクション（例：ON FIND）にある場合は、データビューのコネクションを使用。
- 埋め込み SQL 文がフィールドレベル イベントセクション（例：ON DATA ACCEPT）にある場合は、フィールドが属するデータビューのデータコネクションを使用。

フォームは暗黙的なデータビューとして動作するので注意が必要です。

COMMIT 文や ROLLBACK 文はアプリケーションのトランザクション全体で機能するので、これらの文には USING CONNECTION 節は使用できません。

プロジェクトでコネクションを定義する方法については、『Unify NXJ 開発者ガイド』の第 3 章「プロジェクト」を参照して下さい。

---

## 動的パラメータ

Unify NXJ 変数や Java 変数は、変数名の先頭にコロン “:” を付けると SQL 文から参照できます。SQL 文では、このような変数リファレンスは動的パラメータとして扱われます。

### 例

埋め込み UPDATE 文の例を次に示します。

```
private int setNextNum(String type, int newNum)
    throws java.sql.SQLException
    com.unify.NXJ.mgr.datatypes.NXJNullValueException,
    com.unify.NXJ.mgr.dataConnection.NXJDataConnectionException
{
    EXEC SQL USING CONNECTION tutorial
    UPDATE ROFIDA.NEXT_NUMBER SET NUM = :newNum
    WHERE NTYPE = :type ;
    return session.status;
}
```

この例では 2 つの動的パラメータが使用されています。1 つは NUM に新しい更新値を提供する “:newNum” で、もう 1 つは更新するレコードを識別するためのキーを提供する “:type” です。

## 埋込み型 SELECT 文

埋込み型 SQL SELECT 文の基本構文は、以下のとおりです。

```
EXEC SQL [ USING CONNECTION <connection-name> ]
    SELECT <SQL-SELECT-body>
    INTO <variable-list> ;
```

選択された行それぞれで NXJ プログラミング言語文のブロックを実行するには、以下の構文を使用します。

```

EXEC SQL [ USING CONNECTION <connection-name> ]
  SELECT <SQL-SELECT-body>
    INTO <variable-list>
    EXECUTING
    {
    <Java-statements>
    }

```

Java ループ文の “continue” と “break” は、ループを実行したり中止したりするために、EXECUTING ブロックで使用することができます。

表 4-15 埋込み型 SELECT の構文要素

要素	説明
USING CONNECTION <connection-name>	文が実行されるコネクション名です。 <a href="#">69 ページの「コネクション」</a> を参照してください。
<SQL-SELECT-body>	SQL SELECT 文の本文です。シンプル埋め込み SQL 文の場合と同様、ここに変数を埋め込むことができます。前述の <a href="#">70 ページの「動的パラメータ」</a> を参照してください。
INTO <variable-list>	コマで区切られた変数のリスト。SQL SELECT 文の列の値にそれぞれが対応しています。マッピングされた列の値のデータタイプは、対応する “INTO” 変数のタイプと一致する必要があります。変数の前にはコロンは付けません。INTO 変数は選択されたレコードの列の値を含めるために、毎回リセットされます。
<Java-statements>	SELECT 文でレコードが選択されるたびに実行される NXJ プログラミング言語文の本文です。

SELECT 文にオプションの EXECUTING ブロックがなければ、“INTO” 変数には選択条件に一致する第 1 レコードの値が含まれます。

Unify NXJ 変数をパラメータか、ターゲットの取り出しに使用すれば、Null 値は自動的に処理されます。基本型に NULL 値を選択すると NXJNullValueException が発生します。

以下の NXJ プログラミング言語文では、埋込み型 SELECT 文を使って、指定した日の上位 5 位の映画の配列を作成しています。

---

```

String[] getTopFive(NullableDateVariable weekendDate)
    throws java.sql.SQLException
        com.unify.NXJ.mgr.datatypes.NXJNullValueException,
        com.unify.NXJ.mgr.dataConnection.NXJDataConnectionException
    {
        String[]topFive= new String[5];
        int count = 0;
        int movieId;
        StringmovieTitle=null;
        NullableAmount movieSales;

        EXEC SQL
            SELECT moviegross.mov_id, mov_wkend_gross_sales, mov_title
            FROM movie.moviegross, movie.movielist
            WHERE mov_wkend_dt = :weekendDate
                AND moviegross.mov_id = movielist.mov_id
            ORDER BY mov_wkend_gross_sales ASC
            INTO movieId, movieTitle, movieSales
            EXECUTING
            {
                topFive[count] = movieTitle;
                ++count;
                if ( count >= 5 )
                    {
                        break;
                    }
            }

        return topFive;
    }

```

Oracle データベース使用時には、組み込み SQL SELECT 文が、予期せずに Oracle Sequence を増加させてしまう場合があることに注意して下さい。

他のデータベースでは、組み込み SQL を準備 (prepare) するだけで使用する列リストが利用可能になりますが、Oracle では必要な情報が取得できないために、実際に SQL 文を実行する必要があります。

SELECT 文は、通常データベース中のいかなるデータも変更しませんが、Oracle Sequence (Sequence はデータベースの特殊なオブジェクトで、自動的に、次の番号などのユニークな識別子を生成するために使用されます) を参照する SELECT 文だけは、問題になります。

---

SELECT 文が sequence を参照している場合、その SELECT 文を実行すると、sequence の値が増加してしまいます。したがって、アプリケーションデザイナーは、sequence を参照する SELECT 文のコンパイルチェックを行いません。sequence を参照しているかどうかは、“name.nextval” というパターン (name は識別子) が SELECT 文に含まれているかどうかで判断します (“nextval” に続く識別子は sequence とみなします)。

しかしながら、アプリケーションデザイナーが SELECT 文中の sequence を判別しないようにすることも可能です。例えば、Oracle sequence を参照するテーブルを作成し、そのテーブルをフォームコードから参照するようにします。これにより、アプリケーションデザイナーは、sequence を直接参照しませんので、Oracle Sequence が不必要に増加してしまうことを防ぐことができます。

## 埋込み型ロック文

埋込み型ロック文は、ターゲットテーブル データベースタイプによってサポートされるように、ターゲットテーブルの1つ以上の行、あるいはターゲットテーブル全体をロックすることを可能にします。

ロックを管理するために埋込み型ロック文を使用しない場合、ロックは Unify NXJ が自動で組み込んだ動作によって管理されます。埋め込まれた対話型の操作は、挿入、更新、削除操作が実行されるときに、排他的なロックを自動的に獲得します。ロックは、トランザクションがコミットされるか、ロールバックされると解除されます。ロック文は、異なるタイプのロックを獲得したり、あるいは変更の複雑なセットを形成する前に全ての必要なロックを獲得することを確実にするための機能を提供します。

埋め込み SQL ロック文の構文は、以下のとおりです。

```
EXEC SQL [ USING CONNECTION <connection-name> ]
  LOCK TABLE <table-name>
  IN ( SHARE | UPDATE | EXCLUSIVE ) MODE
  [WHERE <where-clause>];
```

表 4-16 埋込み型ロック文の構文要素

要素	説明
USING CONNECTION <connection-name>	SQL 文が実行されるコネクションを識別するコネクション名です。 <a href="#">69 ページの「コネクション」</a> を参照してください。

表 4-16 埋込み型ロック文の構文要素

要素	説明
<table-name>	ロックが要求されるテーブル。コネクションの中のテーブルでなければなりません。
<where-clause>	ロックが適用される行の選択基準を定義する WHERE 句。WHERE 句が省略される場合は、ロックはテーブル全体に適用されます。

ロックモードのタイプは、以下のとおりです。

- レコードの SHARE ロックは、他のトランザクションがそのレコードを読むことを許可します。また、別のトランザクションより、そのレコードへのロックの更新を許可します。すべての共有ロックが解除されるまで、そのレコード上への排他的ロックを許可しません。
- UPDATE ロックは、これからレコードを変更する意図を示します。他のトランザクションがこのレコード読むことは許可しますが、UPDATE ロックが消失されるまで他のトランザクションがそのレコードの UPDATE や排他的ロックを許可しません。
- 排他的ロックは、そのレコードを変更する前にレコードにかけられます。排他的なロックが削除されるまで、他のトランザクションよりそのレコードを読むことや、どのタイプのロックを獲得することも許可しません。

例えば、以下の NXJ プログラミング言語文は、CATEGORY テーブルのすべての行に UPDATE ロックをするコマンドを定義します。

```
COMMAND tableUpdate
{
EXEC SQL LOCK TABLE CATEGORY IN UPDATE MODE;
if ( session.status == StatusCode.SS_NORM )
    session.displayToMessageBox("Table successfully locked.");
else if ( session.status == StatusCode.SS_NIMP )
    session.displayToMessageBox("Lock type not supported.");
else
    session.displayToMessageBox("Table not locked due to: " +
        session.status);
}
```

Unify NXJ がサポートするすべてのデータベースタイプが、ロックモードの 3 つのタイプすべてをサポートするとは限りません。また、すべてのデータベースタイプが、行ベースのロックを許可するとは限りません。テーブルベースのロックをサポートするだけの場合もあります。表 4-17 は、この機能に関する要約です。

表 4-17 サポートするロックの要約

データベース タイプ	SHARE ロック	UPDATE ロック	EXCLUSIVE ロック
MS/SQL Server	行とテーブル	行とテーブル	行とテーブル
Oracle	テーブルのみ	行とテーブル	テーブルのみ

これらのどのロックも、一旦レコード、もしくはテーブルにかけられると、トランザクションの終わりまでかけられたままです。

## エラー処理

このセクションでは、NXJ アプリケーションでエラーを処理する方法を説明します。標準 Java 例外を使う方法、対話型の操作でエラーをテストする方法、埋め込み SQL 文でエラーを処理する方法があります。

ファウンデーションクラスでメソッドを呼び出すと、例外がスローされエラーが報告されます。これらの例外をキャッチする必要はありませんが、Java では例外がスローされるとそのブロックのコードは実行されなくなるので注意が必要です。イベントセクションからスローされた例外は NXJ インタラクションサーバで自動的に処理されます。例外は Java の “try-catch” 文を使ってキャッチされ、処理されません。

多くの場合、対応する対話型の操作が正しく実行されたかどうかをイベントセクションのコードで確認する必要があります。そのためには、対話型の操作の後にコードで `session.status` プロパティを確認します。

`status` プロパティは、どのようなエラーが発生したかを明らかにするためのデータベース固有の方法です。操作が正しく実行された場合、`status` プロパティはゼロ (`StatusCode.SS_NORM`) です。正しく実行されなかった場合、`status` プロパティには `StatusCode` インタフェースで説明されているエラーコードが設定されます。(アプリケーションデザイナのメニューバーで、ヘルプ > Javadoc を選択して、Javadoc の `StatusCode` インタフェースを参照して下さい。) `status` プロパティは埋め

込み SQL 文によってリセットされるので、埋め込み SQL 文を実行する前にテストすることが必要です。以下のイベントセクションを使用すると示された操作をテストできます。

表 4-18 イベントセクションと操作

イベントセクション	テストする対話型の操作
AFTER FIND	FIND
AFTER ADD	ADD_UPDATE
AFTER UPDATE	ADD_UPDATE
AFTER DELETE	DELETE

埋め込み SQL 文は、行の操作が起きたかどうかをレポートし、エラーが発生している場合は例外をスローして `session.status` を設定します。行が一件も検索されていなかったり、INSERT、UPDATE、DELETE によって何も起きなかったりした場合、`status` プロパティは `StatusCode.SS_NOREC` に設定されます。それ以外の場合は `status` プロパティはゼロ (`StatusCode.SS_NORM`) に設定されます。

埋め込み SQL 文のエラー状態は、例外とともにレポートされます。文が使用する接続の確立に問題がある場合、`NXJDataConnectionException` がスローされることがあります。選択した値が NULL で、対応する INTO 変数が Java 基本型の場合、`NXJNullValueException` がスローされます。データベース固有の問題が文とともにレポートされる場合は、`java.sql.SQLException` がスローされます。このデータベース固有の例外は、`NXJDataConnection` クラスの `mapToStatusCode()` メソッドを使って、対象としているデータベース固有の `StatusCode` 値にマップできます。

`NXJDataConnectionException` と `java.sql.SQLException` は、ランタイム例外ではありません。このため、自分のフォームスクリプトで埋め込み SQL 文を Java メソッドに置く場合は、このような例外をキャッチするかスローすることを宣言する必要があります。

以下の例では、レコードが検索されたかどうかを確認する `session.status` と、SQL 例外を解釈する `mapToStatusCode()` メソッドの両方が使用されています。

```
BEFORE RECORD
{
    try
    {
        String mgrFirstName=null;
        String mgrLastName=null;
        EXEC SQL SELECT FIRST_NAME, LAST_NAME
            FROM ROFIDA.EMPLOYEE
```

---

```
        WHERE EMP_ID = :MANAGER
    INTO mgrFirstName, mgrLastName ;
    if ( session.status == StatusCode.SS_NOREC )
    {
        ManagerField = "no manager";
    }
    else
    {
        ManagerField = mgrFirstName + ' ' + mgrLastName;
    }
}
catch ( java.sql.SQLException sqlEx )
{
    ManagerField = "Error "
        + session.getConnection(connectionName)
            .mapToStatusCode(sqlEx)
        + " on manager lookup";
}
}
```

NXJ プログラミング言語を使用して、カスタムエラー処理を定義することもできます。Unify NXJ Javadoc で、NXJExceptionHandler インタフェースを参照してください。



この章では、NXJ インタラクショナルサーバがアプリケーションを実行し、さらに NXJ プログラミング言語フォームスクリプトを実行する方法を説明します。

この章を読めば、アプリケーションに必要なビジネスルールを実行できるイベントがわかります。また、デバッグのためのアプリケーション実行フローを理解することもできます。

すでに説明されたように、NXJ アプリケーションはイベント駆動で実行されます。イベントが発生すると、NXJ インタラクショナルサーバは一定のアクションを実行します。アクションには、そのイベントのフォームスクリプトで指定したイベントセクションの実行も含まれます。

## NXJ のイベント

NXJ インタラクショナルサーバは NXJ プログラミング言語スクリプトを記述する目的にあわせて、NXJ で定義されているイベントだけを認識します。

表 5-1 NXJ のイベント (1 / 2)

カテゴリ	イベント
アプリケーションレベルの処理	アプリケーションの開始
	アプリケーションの終了
フォームナビゲーション	フォームの初期化
	次フォームに移動
	前フォームに移動
	ズームフォームに移動
	ズームフォームの取り消し

表 5-1 NXJ のイベント (2 / 2)

カテゴリ	イベント
データベース操作	レコードの検索 レコードの追加 レコードの更新 レコードの削除
レコードナビゲーション	選択セットの次レコードに移動 選択セットの前レコードに移動
フィールドナビゲーション	フォーム上のフィールドの位置決め フォームの前フィールドに移動 フォームの次フィールドに移動
開発者定義コマンドの処理	開発者定義コマンドの実行
ユーザインタラクション	Yes/No ダイアログを使用

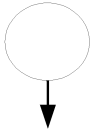
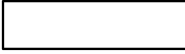




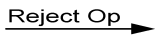
以下のセクションでは、それぞれのイベント、イベントをトリガするアクション、そのイベントの結果として実行されるスクリプトのイベントアクションについて説明します。

---

## 実行シーケンスのダイアグラム

この章では、イベントごとにランタイム実行フローを示すダイアグラムがあります。ダイアグラムはプログラミングのフローチャートに似ており、以下のような記号を使用します。

表 5-2 実行フローのダイアグラムで使用する記号

記号	説明
	操作の開始
	実行される NXJ プログラミング言語 イベントセクション
	NXJ インタラクションサーバによる判断
	別の操作の呼び出し
	NXJインタラクションサーバウィンドウ の表示内容
	NXJ インタラクションサーバでの重要な 処理ステップ
	rejectOperation メソッド実行時のパス

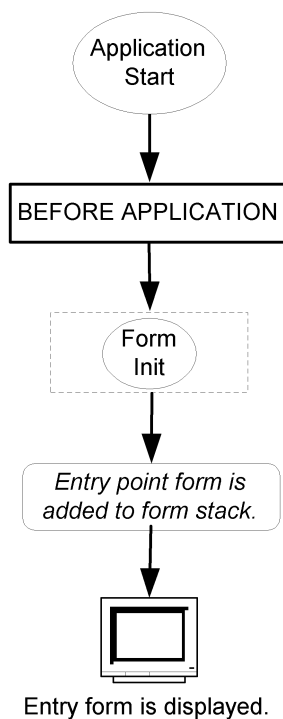
---

---

## アプリケーションの開始

エントリーポイントとして指定されたフォームが NXJ インタラクショナルサーバにアクセスされると、アプリケーションは開始します。通常、ユーザがコントロールセンタビューのエントリーフォームに対応するメニュー項目をクリックすると、エントリーポイントにアクセスします。

アプリケーション開始操作には、以下の実行シーケンスがあります。



- エントリーポイントフォームスクリプトの BEFORE APPLICATION イベントセクションが実行されます。
- エントリーポイントフォームが初期化されます。[85 ページの「フォームの初期化」](#)を参照してください。

- 
- エントリポイントフォームがフォームスタックに追加され、ユーザに表示されます。

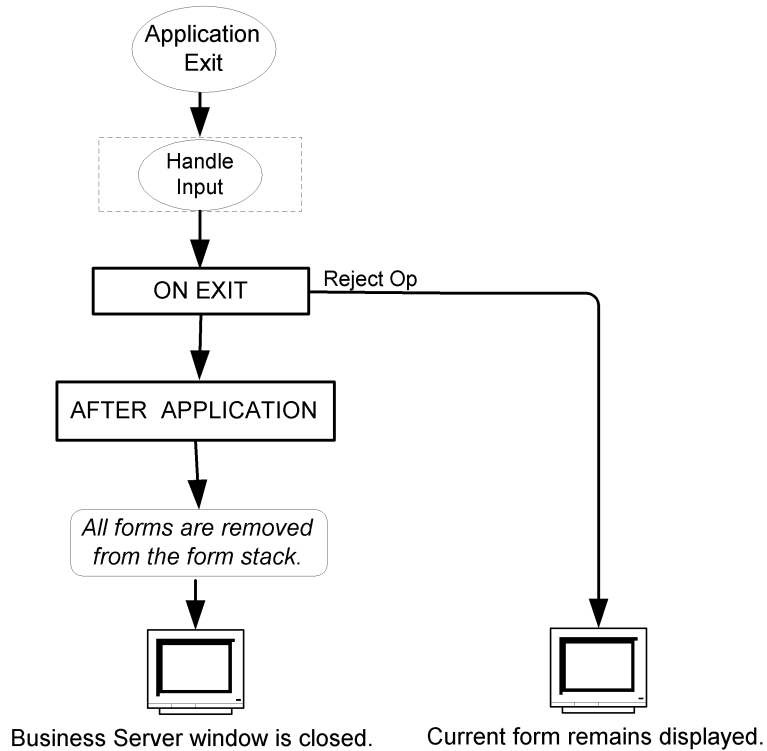
## アプリケーションの終了

アプリケーションは、以下のいずれかの場合に終了します。

- ユーザがデフォルトの Unify NXJ ツールバーで、**終了** ボタンをクリック
  - ユーザが **次フォーム選択** ダイアログで、“Exit the Application ” を選択
  - エントリフォームで PREVIOUS\_FORM 操作を開始し、エントリフォームのスク립トで ON PREVIOUS FORM イベントセクションを実行（拒否されなかった）
  - Internet Explorer では、Web ブラウザウィンドウが閉じる
- Netscape Navigator および Mozilla では、Web ブラウザを閉じても終了処理のトリガにはなりません。



アプリケーション終了操作には、以下の実行シーケンスがあります。

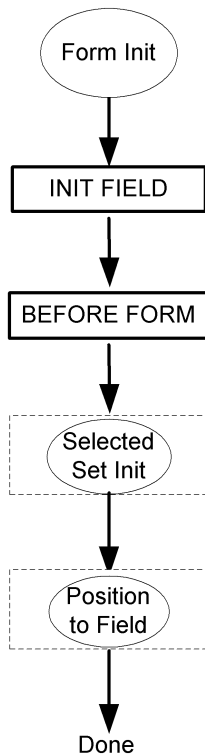


- カレントフォームへのユーザ入力処理が完了します。 [120 ページの「ユーザ入力の処理」](#)を参照してください。
- ON EXIT イベントセクションが実行されます。
- エントリフォームスクリプトの AFTER APPLICATION イベントセクションが実行されます。
- すべてのフォームがフォームスタックから削除されます。

NXJ インタクションサーバ ウィンドウが閉じられます。

## フォームの初期化

NXJ インタラクショナルサーバでフォームにアクセスしたりフォームを開くと、フォームが初期化されます。フォームの初期化操作には、以下の実行シーケンスがあります。



- フォームのフィールドが初期化されます。つまり、フォームスクリプトの INIT FIELD イベントセクションがすべて実行されます（データビューのフィールドも含まれます）。

NXJ インタラクショナルサーバは、フォームスクリプトに表示される順番通りに INIT FIELD イベントセクションを実行するとは限らないので注意が必要です。

- フォームスクリプトの BEFORE FORM イベントセクションが実行されます。

- 
- フォームの選択セットが初期化されます。マスタを持たないフォームのデータビューの選択セットも、やはり初期化されます。[86 ページの「選択セットの初期化」](#)を参照してください。

選択セットに関連のないその他の変数はすべて null 値に初期化されます。

- カーソルのフォーカスは、フォームの第 1 フィールドに設定されます。[114 ページの「フィールド位置」](#)を参照してください。

デフォルトでは、第 1 フィールドはフォームの左上ですが、アプリケーションデザイナーでは別のフィールドを第 1 フィールドに指定することもできます。また、プログラム上は INIT FIELD イベントセクションでフォームの firstField プロパティを設定することで指定できます。

フォームを初期化したら、次にフォームのフィールドをフィールドレベルで処理するか、次フォームに移動します。

## 選択セットの初期化

選択セットとは、検索操作時に検索条件に一致するターゲットテーブルの行のセットです。検索条件がない場合は、ターゲットテーブルのすべての行が選択されます。

選択セット操作の最後に、データビューは次のいずれかのモードになります。

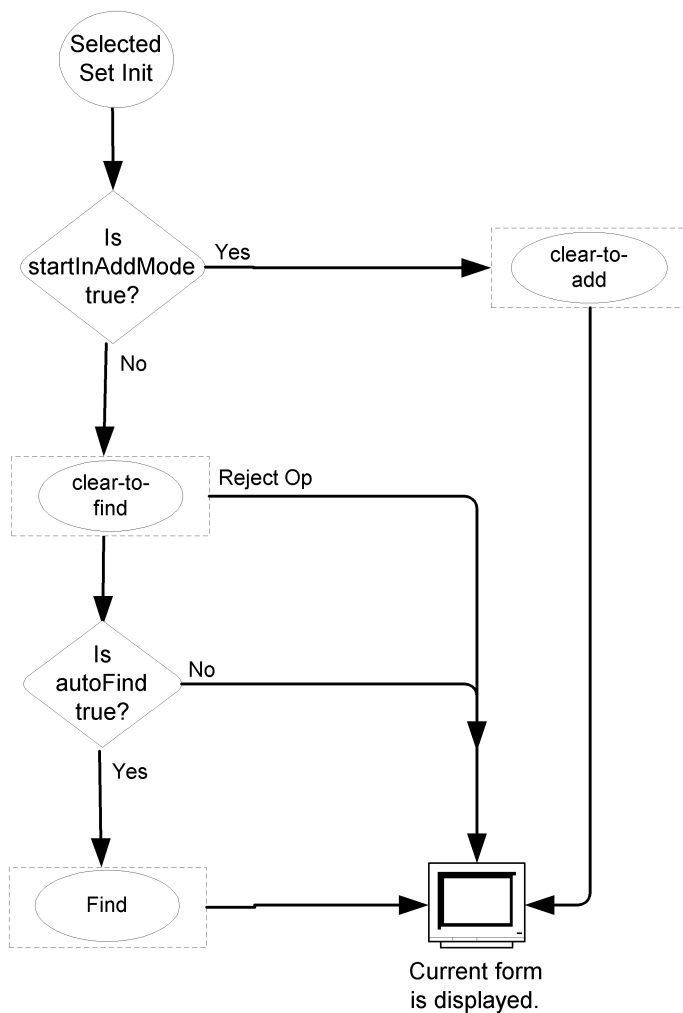
- FIND モード

データビューが FIND モードの場合、ユーザは検索条件を指定して対話型の検索操作、つまり、ターゲットテーブルでデータベース検索を行います。

- 追加 / 更新 / 削除 (AUD) モード

データビューが AUD モードの場合、ユーザは追加、更新、削除の操作をインタラクティブに実行できます。これらの操作はターゲットテーブルで実行されます。

選択セットの初期化には、以下の実行シーケンスがあります。



- startInAddMode プロパティがテストされます。
- startInAddMode プロパティが true であれば、データビューは AUD モードになり、clear-to-add 操作が実行されます。[105 ページの「Clear-to-add 操作」](#)を参照してください。

- 
- startInAddMode プロパティが false であれば、データビューは FIND モードになり、clear-to-find 操作が実行されます。[98 ページの「Clear-to-find 操作」](#)を参照してください。

- autoFind プロパティの値がテストされます。

- autoFind プロパティが true であれば、FIND 操作が実行されます。[96 ページの「レコードの検索」](#)を参照してください。

FIND 操作によって選択セットの行が 1 行または複数行返された場合は、データビューは AUD モードになります。

選択セットの第 1 レコードがカレントになります。

行が返されなければ、データビューは FIND モードのままです。

- autoFind プロパティが false の場合、操作が完了します。

フォームに他のデータビューがあれば、やはりこの方法で初期化されますが、そのデータビューがマスタ / 詳細関係の詳細部分の場合は、この方法では初期化されません。その場合は、以下のようになります。

- 詳細データビューは、マスタのデータビューでレコードがカレントになるたびに初期化されます。
- 詳細データビューは、マスタのデータビューが FIND モードにある場合や使用不可の場合は使用できません。

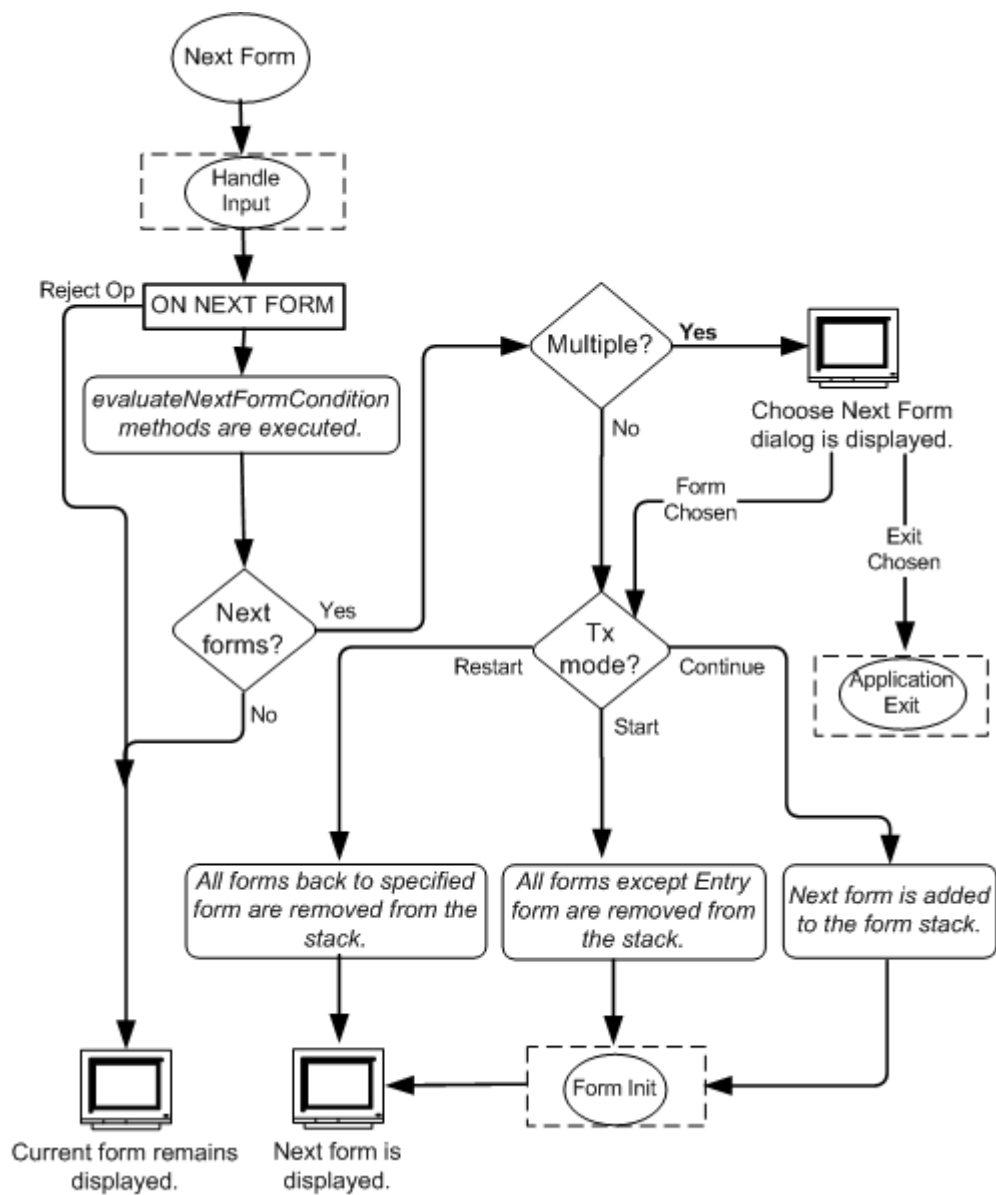
## 次フォームに移動

以下のいずれかのアクションが発生すると、NXJ インタラクションサーバは次フォームに移動します。

- ユーザがデフォルトの Unify NXJ ツールバーで、**進む** ボタンをクリック
- フォームスクリプトで session.queueCommand (“NEXT\_FORM”) メソッドなどの NEXT\_FORM コマンドを実行
- session.queueNextForm メソッドを実行



次フォーム操作には、以下の実行シーケンスがあります。



- 
- カレントフォームへのユーザ入力処理が処理されます。[120 ページの「ユーザ入力の処理」](#)を参照してください。
  - このフォームスクリプトの ON NEXT FORM イベントセクションが実行されます。
  - NXJ インタラクショナルサーバは nextFormList プロパティを評価します。  
次フォームに条件がある場合、フォームスクリプトの `evaluateNextFormCondition()` メソッドが呼び出されます。このメソッドは、次フォームに条件がある場合に必ず呼び出されます。  
  
メソッドが true を返すと対応するフォームが次フォームリストに追加されます。このメソッドが false を返すとフォームは追加されません。  
  
NXJ インタラクショナルサーバにより、次フォームリストのすべてのフォームがアプリケーションに存在するかどうかを確認されます。
  - 次フォームがない場合、次フォーム操作は拒否されます。NXJ インタラクショナルサーバでエラーメッセージが表示され、カレントフォームがそのまま残ります。
  - 複数の次フォーム定義がある場合、次フォーム選択ダイアログが表示され、ユーザに次フォームの指定を求めます。
  - 次フォームが 1 つだけの場合は、次フォーム選択ダイアログは表示されません。次フォームとして、このフォームが自動的に選択されます。
  - NXJ インタラクショナルサーバにより、フォームのトランザクションモードが評価され、フォームスタックが適切に変更されます。  
  
トランザクションモードは nextFormList プロパティで指定します。
  - NXJ インタラクショナルサーバで実行する次フォームが決まると、フォームが初期化されます。[85 ページの「フォームの初期化」](#)を参照してください。
  - 次フォームが表示されます。
  - nextFormList プロパティで、終了操作が次フォームに指定されることがあります。この場合は、次フォーム選択ダイアログに “Exit the Application” と表示されます。ユーザがこの項目を選択すると、アプリケーションが終了します。[83 ページの「アプリケーションの終了」](#)を参照してください。

nextFormList プロパティの詳細については、『Unify NXJ 開発者ガイド』の第 5 章「フォーム」を参照して下さい。evaluateNextFormCondition() メソッドの詳細については、NXJ Javadoc を参照して下さい。

---

## 前フォームに移動

次のいずれかのアクションが発生すると、NXJ インタラクションサーバは前フォームに移動します。

- ユーザがデフォルトの Unify NXJ ツールバーで、**戻る** ボタンをクリック
- フォームスクリプトで `session.queueCommand` (“PREVIOUS\_FORM”) メソッドなどの PREVIOUS\_FORM コマンドを実行

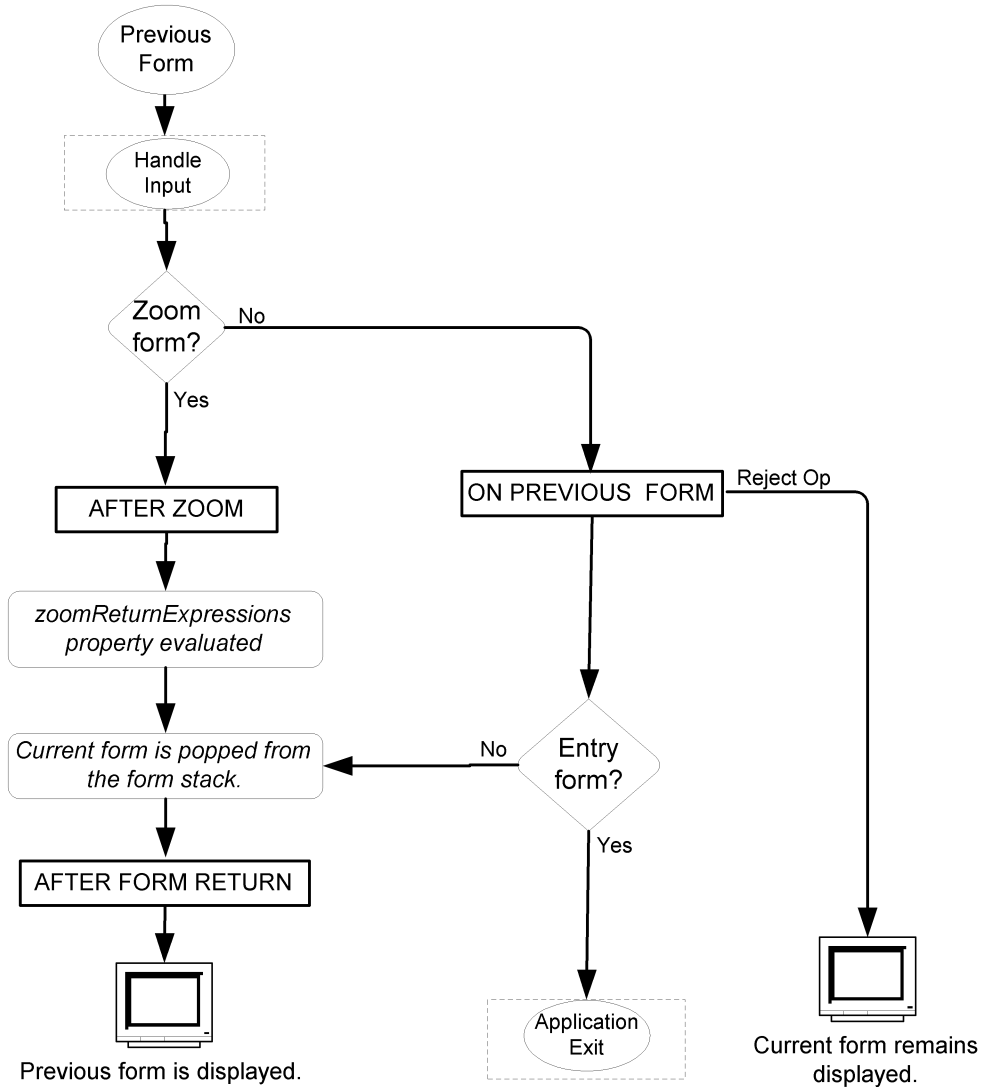


---

**注** - ズームフォームから呼び出し元のフォームに移動しても前フォーム操作とはみなされません。[95 ページの「ズームフォームの取り消し」](#)を参照してください。

---

前フォーム操作には、以下の実行シーケンスがあります。



- 
- カレントフォームへのユーザ入力処理が処理されます。[120 ページの「ユーザ入力の処理」](#)を参照してください。
  - カレントフォームがズームフォームの場合、AFTER ZOOM イベントセクションが実行されます。`zoomReturnExpressions` プロパティが定義されている場合、評価されて戻されます。このプロパティが定義されていない場合、現在の行のプライマリーキーの値が戻されます。戻された値は、`zoomReturnValuesInto` プロパティに格納されます。
  - ズームフォームではない場合、ON PREVIOUS FORM イベントセクションが実行されます。
  - カレントフォームがエントリフォームの場合、アプリケーションが終了します。[83 ページの「アプリケーションの終了」](#)を参照してください。
  - カレントフォームが非アクティブ化され、フォームスタックから削除されます。フォームで定義したローカル変数はすべて、アプリケーションで他のフォームの外部リファレンスとしてアクセスすることはできなくなります。
  - カレントフォームスクリプトの AFTER FORM RETURN イベントセクションが実行されます。
  - 前フォームが表示され、カレントフォームになります。

このフォームでは、ズーム戻り値を受け取ったフィールドに次のフィールドがあれば、そのフィールドがアクティブになります。それ以外の場合、カーソルのフォーカスは、フォームの最初のフィールドに位置決めされます。

[114 ページの「フィールド位置」](#)を参照してください。

## ズームフォームに移動

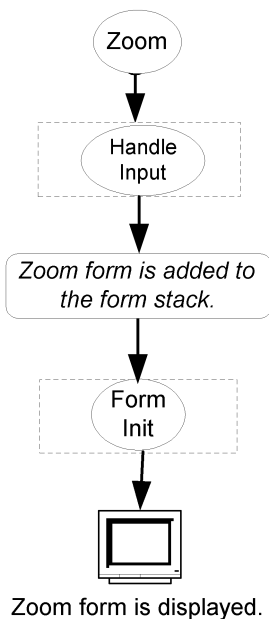
次のいずれかのアクションが発生すると、NXJ インタラクションサーバはズームフォームに移動します。

- ユーザがデフォルトの Unify NXJ ツールバーで、**ズーム** ボタンをクリック
- ズーム可能フィールドがカレントの時のみ、このボタンは使用可能になります。
- ズームが有効なフィールドが選択されており、`autoZoom` プロパティが `true` に設定されている
  - ズームが有効なフィールドが選択されている場合に、フォームスクリプトで `session.queueCommand` (“ZOOM”) メソッドなどの ZOOM コマンドを実行



---

ズーム操作には、以下の実行シーケンスがあります。



- カレントフォームへのユーザ入力処理が完了します。[120 ページの「ユーザ入力の処理」](#)を参照してください。
- フォームスタックにズームフォームが追加されます。
- ズームフォームが初期化されます。[85 ページの「フォームの初期化」](#)を参照してください。
- ズームフォームが表示されます。

ズームフォームの定義については、『Unify NXJ 開発者ガイド』の第 5 章「フォーム」を参照して下さい。

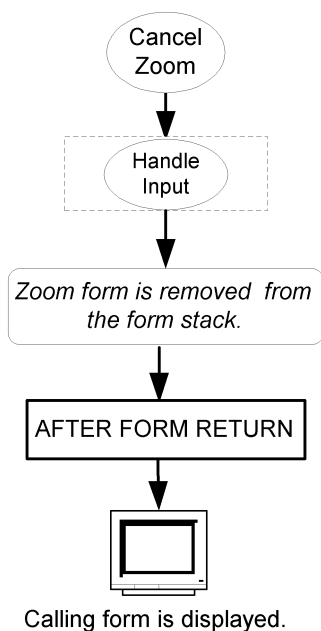
## ズームフォームの取り消し

次のいずれかのアクションが発生すると、NXJ インタラクションサーバによってズームフォームが取り消されます。

- ユーザがデフォルトの Unify NXJ ツールバーで、**取消** ボタンをクリック
- `session.queueCommand` (“CANCEL\_ZOOM”) メソッドなどのフォーム スクリプトで CANCEL\_ZOOM コマンドを実行



キャンセルズーム操作には、以下の実行シーケンスがあります。



- カレントフォームへのユーザ入力処理されます。 [120 ページの「ユーザ入力の処理」](#)を参照してください。
- フォームスタックからズームフォームが削除されます。
- 呼び出しフォームのフォームスクリプトにある AFTER FORM RETURN イベントセクションが実行されます。

- 
- 呼び出しフォームの実行が再開されます。

このフォームのズームフィールドはアクティブになります。

---

**注** – ズームフォームを使うと、呼び出しフォームのフィールドに返す値を前フォーム操作の実行中に選択できます。取り消し操作を行うと、値は返されません。

---

## レコードの検索

次のいずれかのアクションが発生すると、NXJ インタラクションサーバによって検索操作が開始されます。

- ユーザがデフォルトの Unify NXJ ツールバーで、**検索** ボタンをクリック
- フォームが初期化され、そのフォームプロパティ `autoFind` は、`true` に設定、`startInAddMode` プロパティは、`false` に設定されている
- `session.queueCommand` (“FIND”) メソッドなどのフォームスクリプトで `FIND` コマンドを実行

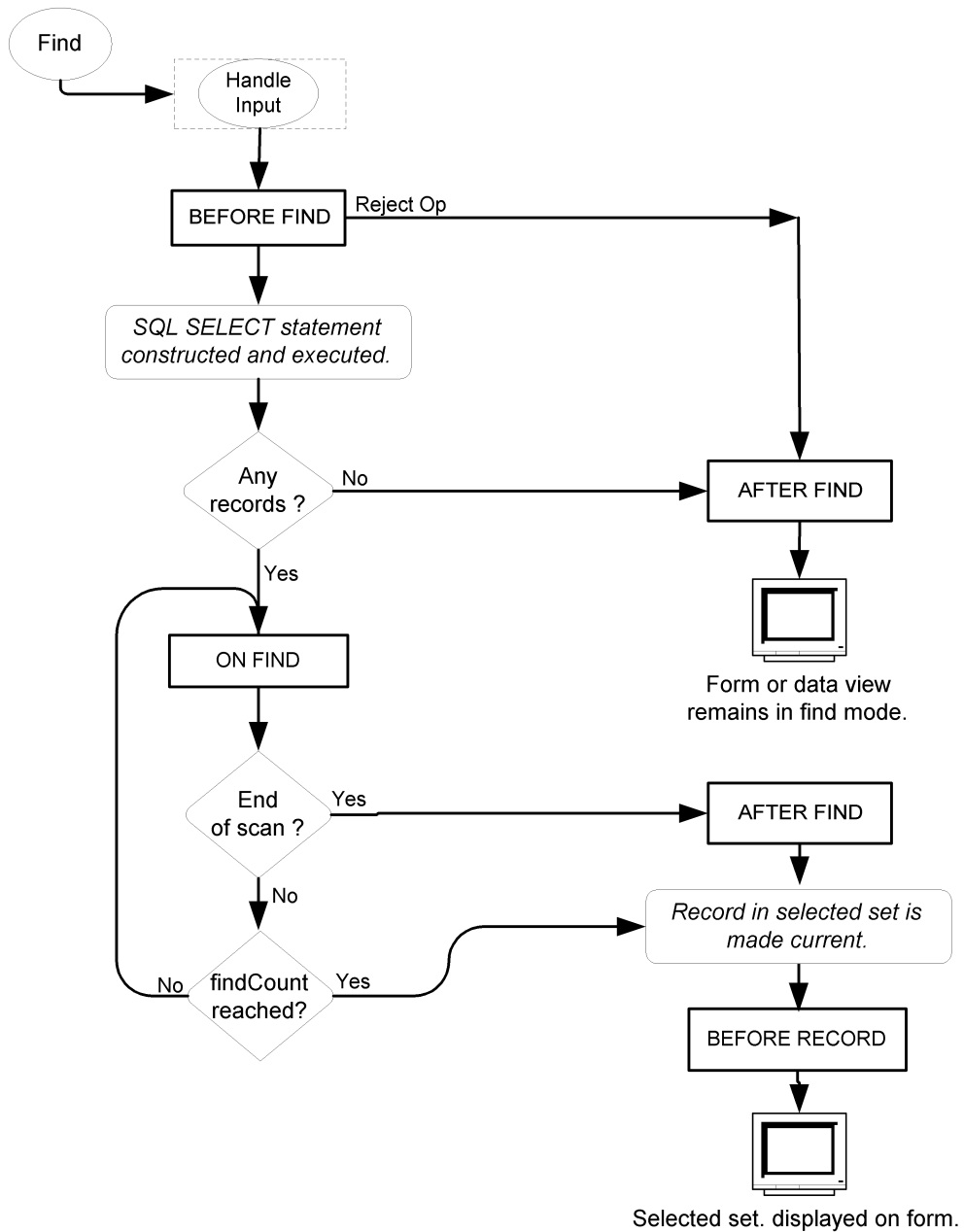


---

**注** – フォームスクリプトの埋込み型 SQL `SELECT` 文は、検索操作とは見なされません。

---

検索操作には、以下の実行シーケンスがあります。



- 
- カレントフォームへのユーザ入力処理が処理されます。[120 ページの「ユーザ入力の処理」](#)を参照してください。
  - フォームスクリプトの BEFORE FIND イベントセクションが実行されます。
  - SQL 文が作成され、フォームのターゲットテーブルに対して発行されます。[100 ページの「SQL Select 文の構成」](#)を参照してください。

findAllowed プロパティは、検索を許すためには true を設定していなければなりません。

- 対象レコードに対して、ON FIND イベントセクションが各レコード毎に 1 回実行されます。ターゲットフィールド / 変数は、ON FIND が実行されている間に代入されます。
- findCount プロパティは、大量検索操作に割り込むかどうかを決めるために検査されます。
- データ表示モードは、AUD モードに切り替えます。選択セットの第 1 レコードが、フォームのカレントレコードになります。
- 選択セットが作成されない場合、つまりレコードが見つからないまたは、全てのレコードを拒否したときは、データ表示モードは、検索モードのまま残ります。
- スキャンが完了すると、フォームスクリプトの AFTER FIND イベントセクションが実行されます。

検索操作が、選択セットを返すと、ユーザは、AUD モードで有効なユーザコマンドのどれでも実行する事ができます。それ以外では、フォームは検索モードが残り、ユーザは、別の検索操作を発行することができます。検索条件は、フォームに表示されて残ります。

## Clear-to-find 操作

clear-to-find 操作により、対話型の検索フォームが用意されます。検索モードにフォームが設定された場合、NXJ インタラクションサーバは clear-to-find 操作を自動的に開始します。

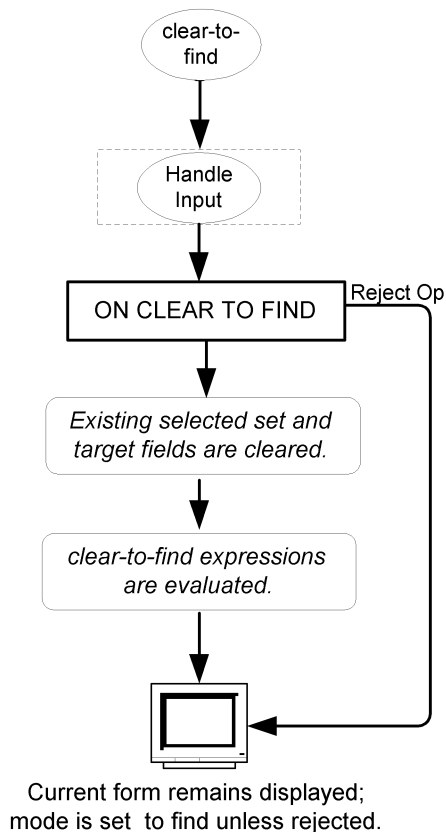


次のいずれかのアクションが発生すると、clear-to-find 操作が開始されます。

- ユーザがデフォルトの Unify NXJ ツールバーで、**クリア** ボタンをクリック

- 
- `session.queueCommand` (“CLEAR\_TO\_FIND”) メソッドなどのフォームスクリプトで `CLEAR_TO_FIND` コマンドを実行

clear-to-find 操作には、以下の実行シーケンスがあります。



- カレントフォームへのユーザ入力処理が完了します。 [120 ページの「ユーザ入力の処理」](#) を参照してください。
- ON CLEAR TO FIND イベントセクションが実行されます。
- 既存の選択セットがクリアされ、すべてのターゲットフィールドがクリアされます。

非ターゲットフィールドと検索不可フィールド、および関連する詳細データビューは無効にされます。

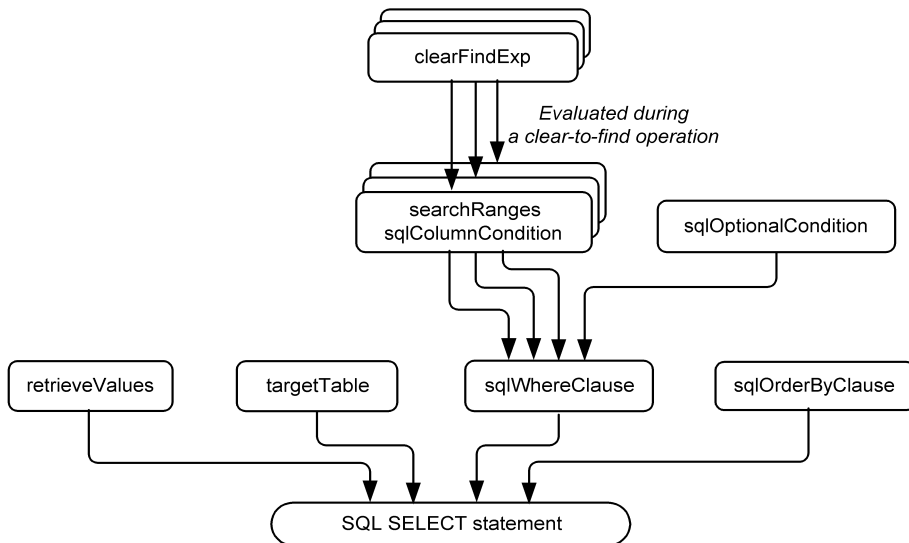
- 各フィールドの `clearFindExp` プロパティで指定した `clear-to-find` 式が評価され、結果がフォームに表示されます。

検索モードのフォームが初期化されると、ユーザは検索操作を準備するフォームに新しい検索条件の入力や、既存の条件の変更を行うことができます。

## SQL Select 文の構成

NXJ インタクションサーバは、いくつかのプロパティを評価することによる検索操作のターゲットテーブルに対して発行する SQL Select 文を作成します。

以下の図に、SQL Select 文の構成に必要なプロパティを示します。



---

以下の表では、SQL SELECT 文に関連するプロパティを説明しています。

表 5-3 SQL SELECT 文の作成に関連するプロパティ

プロパティ	説明
clearFindExp	フォームの各ターゲットフィールドは、このプロパティの値を持っています。ターゲットカラムの最初の検索条件を指定します。最初の検索条件は、検索操作を始める前にユーザが変更することができます。
searchRanges	このプロパティの値は、clear-to-find 操作中に clearFindExp プロパティに基づいて算出されます。
sqlColumnCondition	searchRanges プロパティと同じ情報の SQL に基づく説明です。
sqlOptionalCondition	フォームやデータビューは、アプリケーションデザイナーの Sql optional Condition プロパティで指定するようにこのプロパティに値を設定することができます。
explicitSearchMode	各ターゲットフィールドは、検索条件が、解読される方法をこのプロパティに fuzzy か exact を設定することによって指定することができます。
sqlWhereClause	NXJ インタラクションサーバにより構築されて、このプロパティに Where 句が格納されます。
sqlOrderByClause	フォームやデータビューは、アプリケーションデザイナーの Sql Order By Clause プロパティに指定するようにこのプロパティに値を設定することができます。
targetTable	アプリケーションデザイナーのフォームやデータビューの Target Table プロパティによって指定されるような SELECT 文に対して発行するターゲットテーブルの名前です。

---

---

## レコードの追加と更新

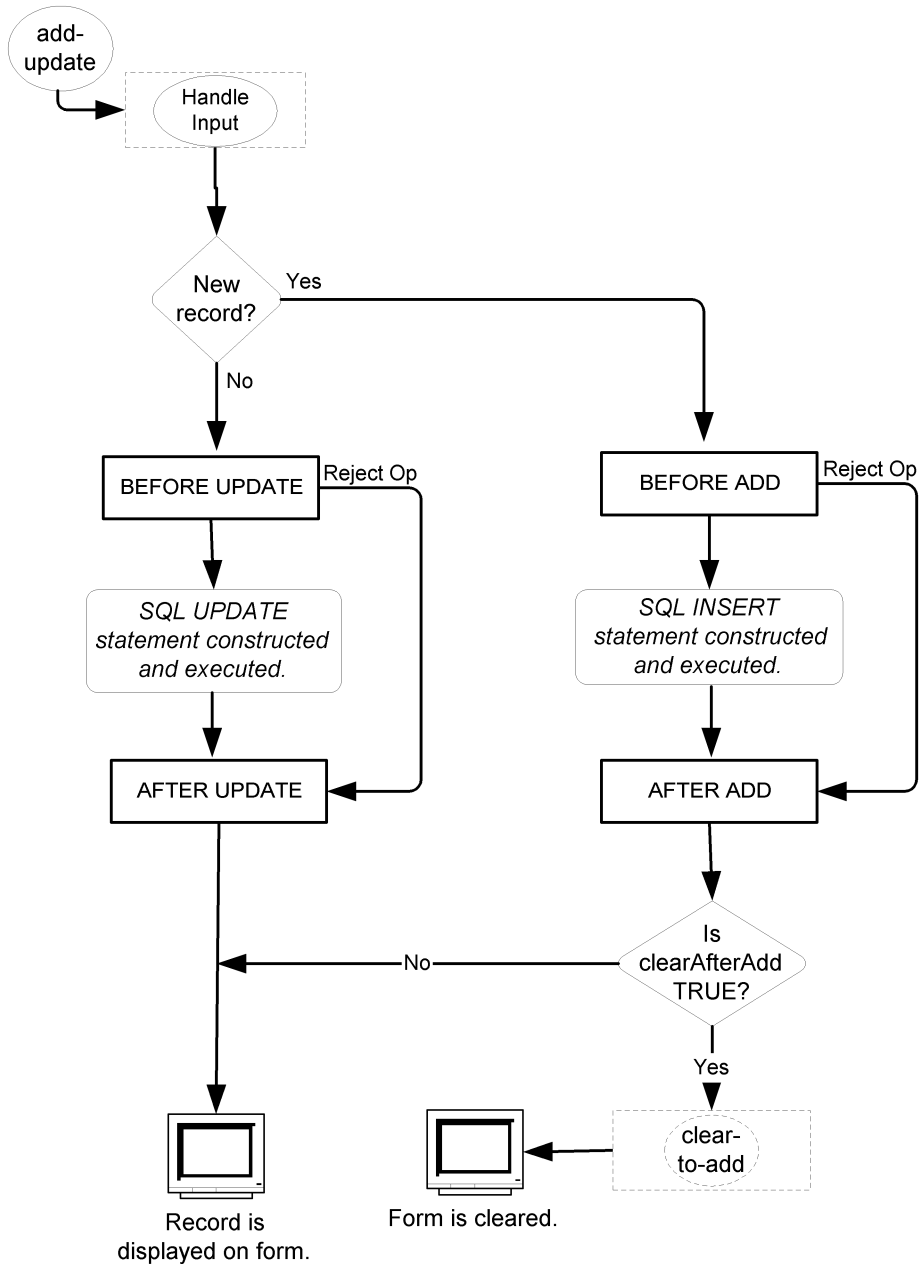
追加操作と更新操作の手順は似ています。ここでは両方の手順について説明します。レコードを含むデータビューに Batch Record Changes プロパティが設定されている場合は、更新は一括処理となり、ユーザが繰り返し領域の外をクリックしたとき、フォームナビゲーションコマンドを処理したとき、あるいは、選択セット内で新しいレコードグループを表示するようなナビゲーションを行ったときに処理が起動されます。

以下のいずれかのアクションが発生すると、NXJ インタラクションサーバによって追加 / 更新操作が開始されます。

- ユーザがデフォルトの Unify NXJ ツールバーで、**更新** ボタンをクリック
- 繰り返し領域内に表示されているレコードの場合は、ある行の値を変更した後にその行以外の場所をクリック
- `session.queueCommand (“ADD_UPDATE”)` メソッドを実行した場合のように、フォームスクリプトで `ADD_UPDATE` コマンドを実行



フォームの追加 / 更新操作には、以下の実行シーケンスがあります。



- 
- カレントフォームへのユーザ入力処理が処理されます。[120 ページの「ユーザ入力の処理」](#)を参照してください。
  - NXJ インタクションサーバは、カレントレコードが、新規であるか既存であるを判断し、そしてそれに応じて追加または、更新操作を続けます。
  - 更新操作の場合、以下のアクションが発生します。
    - BEFORE UPDATE イベントセクションが実行されます。
    - 対応する SQL UPDATE 文が構成され、ターゲットテーブルに対して発行されます。  
updateAllowed プロパティは、更新を許すために true に設定されていなければなりません。
    - たとえ準備された SQL UPDATE 文が、正常に完了しない場合でも、AFTER UPDATE イベントセクションは、実行されます。session.status プロパティをチェックしてエラーを確認することができます。
  - 追加操作の場合、以下のアクションが発生します。
    - BEFORE ADD イベントセクションが実行されます。
    - 対応する SQL INSERT 文が構成され、ターゲットテーブルに対して発行されます。  
insertAllowed プロパティは、挿入を許すために true にセットされていなければなりません。排他的ロックが、新しい行に設定され、これはトランザクションが終了するまで保持されます。
    - たとえ準備された SQL Insert 文が、正常に完了しない場合でも、AFTER ADD イベントセクションは、実行されます。session.status プロパティをチェックしてエラーを確認することができます。
    - clearAfterAdd プロパティが評価されます。true の場合、clear-to-add 操作が実行されます。[105 ページの「Clear-to-add 操作」](#)を参照してください。false の場合、position-to-field 操作が実行されます。[114 ページの「フィールド位置」](#)を参照してください。

追加操作が成功すると、新しいレコードが選択セットの最後に追加されそして、レコードは、ターゲットテーブル行として追加されます。追加されたレコードは、新しいカレントレコードになります。

NXJ インタクションサーバは、カレントフィールドで実行を再開します。カレントフィールドで stopForInput プロパティが true に設定されていない場合、NXJ インタクションサーバは次のフィールド操作を開始します。[116 ページの「次フィールドに移動」](#)を参照してください。

---

## Clear-to-add 操作

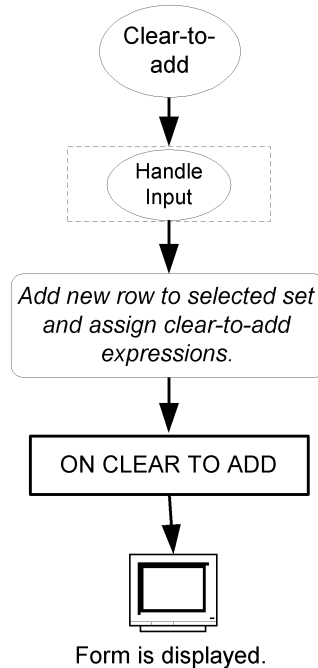
clear-to-add 操作は、新規レコードの追加のためにフォームとデータビューを準備します。clear-to-add 操作は、選択セットの初期化または追加 / 更新操作の一部として実行されることがよくあります。また、以下の動作のうちひとつが発生した時にも、clear-to-add 操作が実行されます。

- ユーザがデフォルトの Unify NXJ ツールバーで、**新規** ボタンをクリック
- フォームが初期化され、startInAddMode プロパティは true に設定されている
- 追加操作が実行され、フォームの clearAfterAdd プロパティは、true に設定されている
- session.queueCommand (“CLEAR\_TO\_ADD”) メソッドなどのフォームスクリプトで CLEAR\_TO\_ADD コマンドを実行
- 選択セットの最終レコードを削除



---

clear-to-add 操作には、以下の実行シーケンスがあります。



- カレントフォームへのユーザ入力処理されます。 [120 ページの「ユーザ入力の処理」](#)を参照してください。
- 選択セットに空の新規レコードが追加されます。フォームフィールドがすべてクリアされます。
- データ表示モードが AUD になります。
- clear-to-add 式が評価され、デフォルトのフィールドと変数の値が取得されます。

新規レコードが選択セットに追加される時に、clearAddExp プロパティは、関連するフィールドのデフォルト値を指定する NXJ プログラミング言語式になります。このプロパティがフィールドに設定されていなければ、NXJ インタクションサーバは、変数をプロジェクトの “application.properties” ファイルに指定されているデータタイプごとの値に初期化します。デフォルトで、

---

null 値設定可能な変数、あるいは必要に応じて、Java 値がゼロ、false、null である場合、ファイルのすべてのデータタイプに指定される値は NULL です。

- clear-to-add 操作が、フォームスクリプトの `session.queueCommand("CLEAR_TO_ADD")` によって始められていなければ、ON CLEAR TO ADD イベントセクションは、実行されます。
- position-to-field 操作が実行されます。 [114 ページの「フィールド位置」](#) を参照してください。

この時点で、ユーザは新規レコードの必要なフィールドに値を入力し、ツールバーの **更新** ボタンをクリックしてレコードを保存します。この新規レコードを保存する前にユーザが別のレコードに移動すると、新規レコードの値は破棄されます。

“application.properties” ファイルについての詳細は、『開発者ガイド』の第 3 章を参照してください。

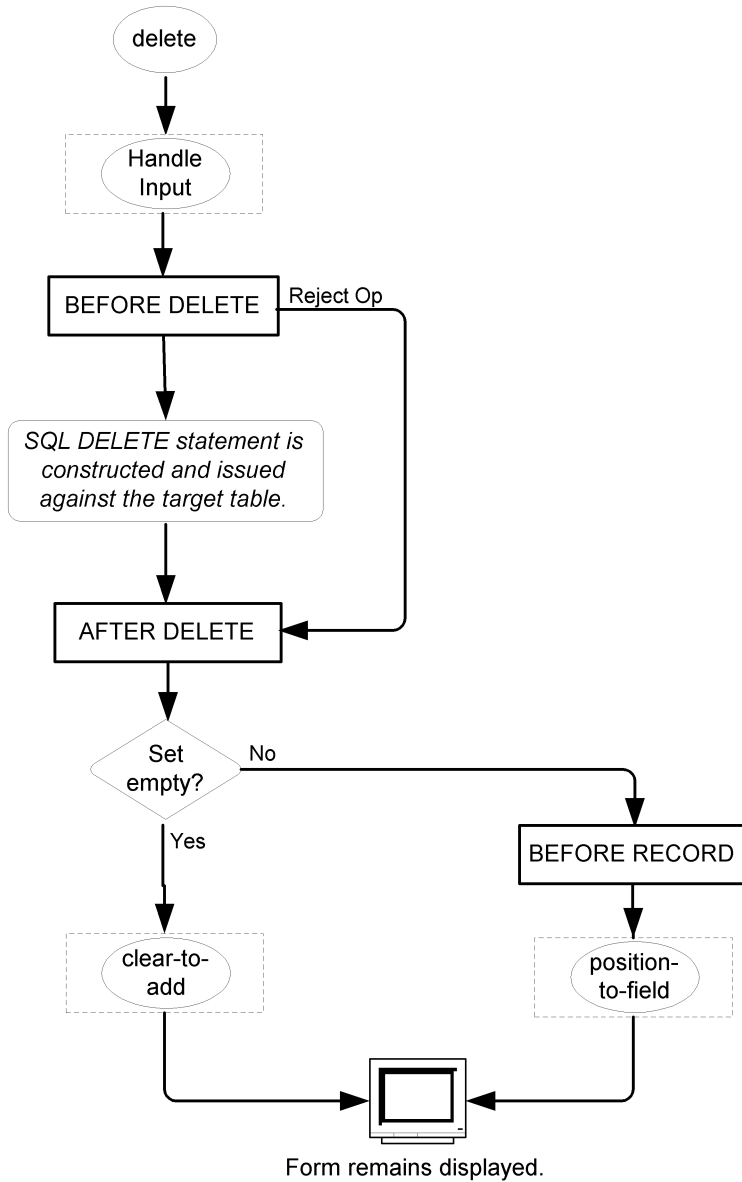
## レコードの削除

次のいずれかのイベントが発生すると、NXJ インタクションサーバは、選択セットからカレントレコードを削除します。

- ユーザがデフォルトの Unify NXJ ツールバーで、**削除** ボタンをクリック
- DELETE コマンドが、`session.queueCommand("DELETE")` メソッドからのようなフォームスクリプト内の実行



削除操作には、以下の実行シーケンスがあります。



- 
- カレントフォームへのユーザ入力処理が処理されます。[120 ページの「ユーザ入力の処理」](#)を参照してください。
  - BEFORE DELETE イベントセクションが実行されます。
  - 対応する SQL DELETE 文が構成され、ターゲットテーブルに対して発行されます。

deleteAllowed プロパティは、削除を許すために true に設定されてなければなりません。




その上に、NXJ インタラクショナルサーバは、データベース行の排他的ロックを削除しているときに要求します。この排他的ロックは、カレントトランザクションがコミットされるまで保持されます。排他的ロックが獲得できない場合、操作は拒否されます。

- 削除操作が成功すると、NXJ インタラクショナルサーバは、選択セットの次レコードに移動します。NXJ インタラクショナルサーバは選択セットの次レコードに移動します。[110 ページの「次レコードに移動」](#)を参照してください。
- AFTER DELETE セクションは、削除操作が成功するかどうかに関わらず実行されます。session.status プロパティをチェックしてエラーを確認することができます。
- 削除操作後に選択セットに残っているレコードがなければ、clear-to-add 操作が実行されます。[105 ページの「Clear-to-add 操作」](#)を参照してください。
- 選択セットに 1 つのレコードが存在すれば、レコードに対するカレントフォームスクリプトの BEFORE RECORD イベントセクションが、実行されてその後、position-to-field 操作が実行されます。[114 ページの「フィールド位置」](#)を参照してください。

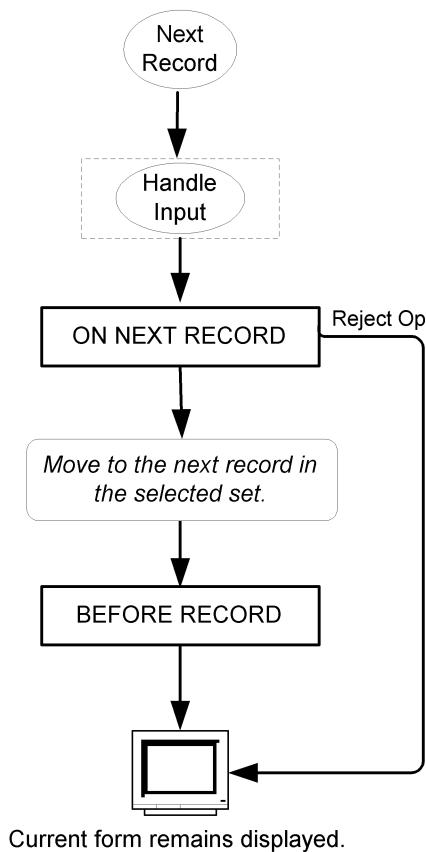
---

## 次レコード に移動

以下のいずれかのアクションが発生すると、NXJ インタラクショナルサーバは選択セットの次レコード に移動します。

- ユーザがデフォルトの Unify NXJ ツールバーで、**次レコード** ボタンをクリック  Next Record
- ユーザがデフォルトの Unify NXJ ツールバーで、**次セット** ボタンをクリック  Next Set
- ユーザがデフォルトの Unify NXJ ツールバーで、**最終レコード** ボタンをクリック  Last Record
- ユーザが行をクリックするか、キーボードの矢印キーを使用するかによって、繰り返し領域のカレント行をナビゲートする
- session.queueCommand (“NEXT\_RECORD”) メソッドからのように、NEXT\_RECORD、LAST\_RECORD または NEXT\_SET コマンドが、フォームスクリプトで実行される
- 選択セットのカレントレコードを削除

Batch Record Changes プロパティは、次レコード操作が起きたときにどのイベントセクションが実行されるかに影響します。Batch Record Changes プロパティが false であれば、次レコード操作は、以下の実行シーケンスになります。



- カレントフォームへのユーザ入力処理が実行されます。 [120 ページの「ユーザ入力の処理」](#)を参照してください。
- カレントレコードに変更が加えられている場合には、カレントデータベーススクリプトの ON NEXT RECORD イベントセクションが実行されます。変更が無い場合には、イベントセクションは実行されません。
- 選択セットの次のレコードがカレントになります。

- 
- カレントデータビュースクリプトの BEFORE RECORD イベントセクションが実行されます。

Batch Record Changes プロパティが true の場合、次レコード操作はカレントレコードに変更があったかどうかにかかわらず、カレントフォームスクリプトの ON NEXT RECORD イベントセクションが常に実行されることを除けば上に説明された通りにまとめて実行されます。

## 前レコードに移動

以下のいずれかのアクションが発生すると、NXJ インタラクションサーバは前レコードに移動します。

- ユーザがデフォルトの Unify NXJ ツールバーで、**前レコード** ボタンをクリック
- ユーザがデフォルトの Unify NXJ ツールバーで、**前セット** ボタンをクリック
- ユーザがデフォルトの Unify NXJ ツールバーで、**先頭レコード** ボタンをクリック
- ユーザは、行をクリックするか、キーボードの矢印キーを使用することで、繰り返し領域のカレント行の上の行で操作
- session.queueCommand (“PREVIOUS\_RECORD”) メソッドからのように、PREVIOUS\_RECORD、FIRST\_RECORD または PREVIOUS\_SET コマンドがフォームスクリプトで実行される



Previous Record

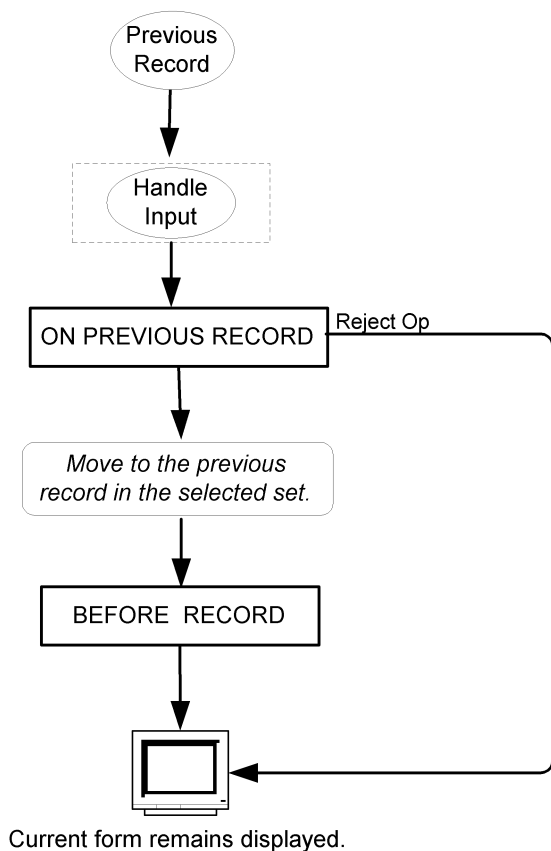


Previous Set



First Record

Batch Record Changes プロパティは、前レコード操作が起きたときにどのイベントセクションが実行されるかに影響します。Batch Record Changes プロパティが false であれば、前レコード操作は、以下の実行シーケンスになります。



- カレントフォームへのユーザ入力処理が実行されます。 [120 ページの「ユーザ入力の処理」](#)を参照してください。
- カレントレコードに変更が加えられている場合には、カレントフォームスクリプトの ON PREVIOUS RECORD イベントセクションが実行されます。変更が無い場合には、イベントセクションは実行されません。
- 選択セットの前のレコードがカレントになります。

- 
- カレントフォームスクリプトの BEFORE RECORD イベントセクションが実行されます。

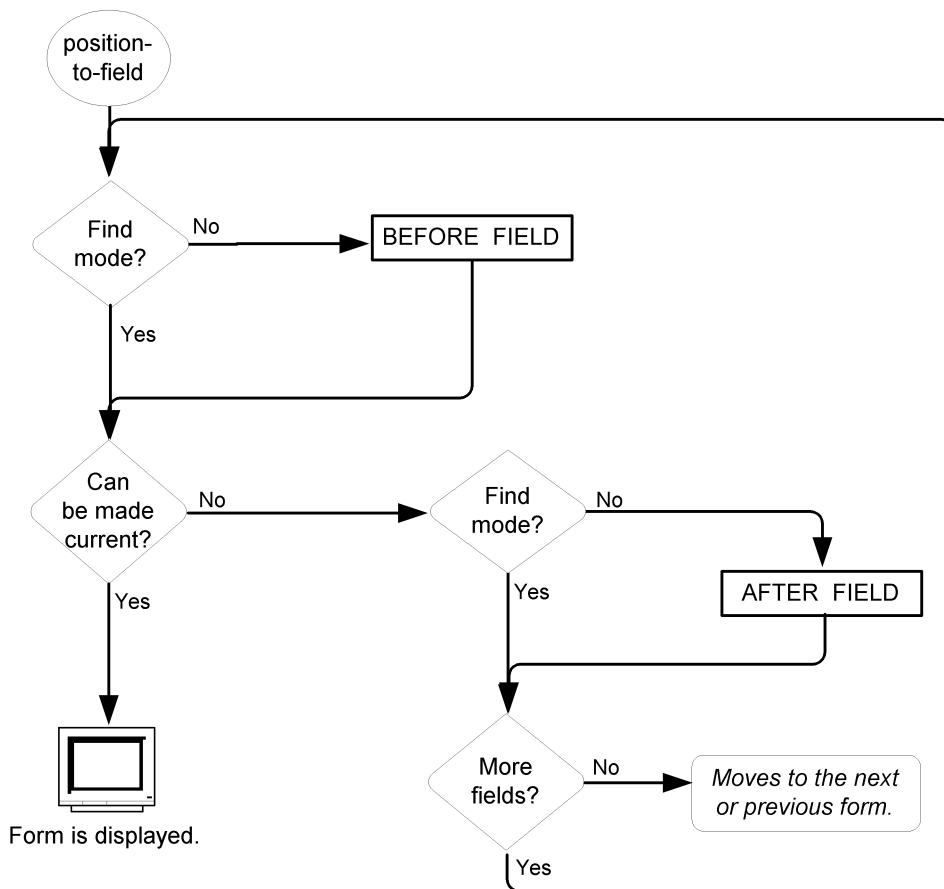
Batch Record Changes プロパティが true であれば、次レコード操作はカレントレコードに変更があったかどうかにかかわらず、カレントフォームスクリプトの ON PREVIOUS RECORD イベントセクションが常に実行されることを除けば上に説明された通りにまとめて実行されます。

## フィールド位置

NXJ インタラクションサーバは、フォーム上にフィールドをカレントにしたフィールドを配置します。フォームのカレントフィールドにはフォーカスがあり、ユーザ入力やユーザのアクションを受け付けます。

position-to-field 操作は、フォーム初期化や次フィールド操作のような他の実行操作から起動されます。

position-to-field 操作には、以下の実行シーケンスがあります。



- 対象となるフォームへのユーザ入力処理が処理されます。[120 ページの「ユーザ入力の処理」](#)を参照してください。
- フィールドのデータビューが、検索モードであるか確認されます。データビューが検索モードではない場合、BEFORE FIELD イベントセクションが実行されます。
- フィールドは、以下のようなカレントになることができるかどうかを決めるためにテストされます。
  - フィールドは、stopForInput フィールドプロパティに true に設定されている。

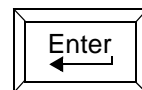
- 
- ・フィールドが属するデータビューは、無効になっていない。
  - ・フィールドは、ターゲットフィールドである。データビューが、検索モードの時、非ターゲットフィールドは、スキップされます。
  - ・フィールドがカレントになることができれば、それをカレントにして、position-to-field 操作は、完了します。
  - ・フィールドがカレントになれない、そしてフィールドのデータビューが検索モードではない場合、AFTER FIELD イベントセクションが実行されます。
  - ・どれがカレントにすることができるか確かめるため、フォーム上のフィールドは、テストされます。
- フィールドが、カレントになれない場合、次フォーム操作が実行されます。次フォーム操作が失敗した場合、前フォーム操作が実行されます。

## 次フィールドに移動

フォームにはフィールド順があり、次フィールド操作や前フィールド操作によってアクセスされる順番が決まっています。次フィールド操作により、カレントフィールドから次のフィールドに移動します。

以下のいずれかのアクションがカレントフォームに発生すると、次フィールド操作が実行されます。

- ・ユーザがキーボードの Enter/Return キーを押下。ただし、プロジェクトのカレント look & feel 定義でデフォルトになっている NEXT\_FIELD とは異なるコマンドが Enter/Return キーにマップされている場合を除く
- ・session.queueCommand (“NEXT\_FIELD”) メソッドからのように、NEXT\_FIELD コマンドがフォームスクリプトで実行される



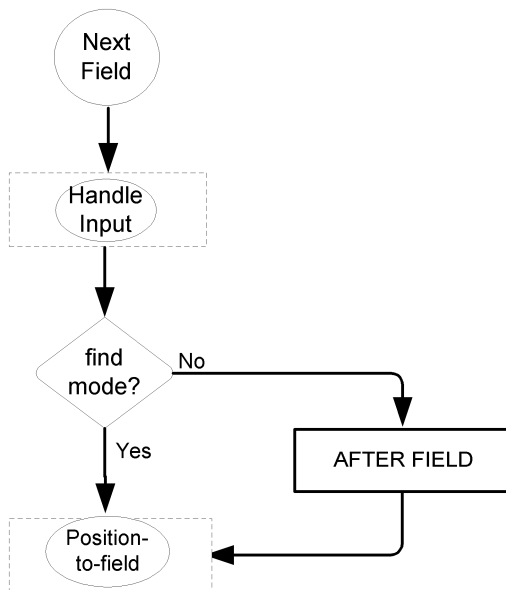
---

**注** – デフォルトでは、フィールドレベル イベントセクションの実行は、まとめて一度に複数のフィールドに起こります。フィールドの Immediate プロパティに True を設定することにより、そのフィールドに対してユーザが操作をした場合には、そのフィールドのイベントセクションを単独で実行するように指定することができます。このセクションの実行シーケンスの図は、フィールドレベル イベントセクションの即時の実行を表示しています。

---

---

次フィールド操作には、以下の実行シーケンスがあります。



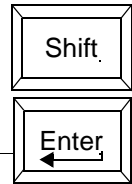
- カレントフィールドの置かれているデータビューが検索モードであるか確認します。
- データビューが検索モードでなければ、NXJ インタクションサーバはフォームスクリプトの AFTER FIELD イベントセクションを実行します。
- NXJ インタクションサーバは position-to-field 操作を実行し、次フィールドにフォーカスを移動します。[114 ページの「フィールド位置」](#)を参照してください。

## 前フィールドに移動

フィールドには次フィールドと同様、前フィールドがあります。前フィールド操作により、フォーカスをカレントフィールドから前のフィールドに移動します。

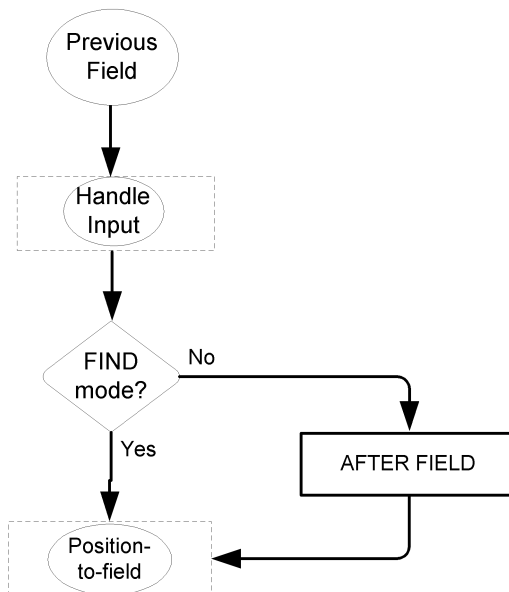
以下のいずれかのアクションが発生すると、前フィールド操作が実行されます。

- ユーザがキーボードで Shift+Enter キーの組み合わせを押下
- session.queueCommand (“PREVIOUS\_FIELD”) メソッドからのように、PREVIOUS\_FIELD コマンドがフォームスクリプトで実行される



注 - デフォルトでは、フィールドレベル イベントセクションの実行は、まとめて一度に複数のフィールドに起こります。フィールドの Immediate プロパティに True を設定することにより、そのフィールドに対してユーザが操作をした場合には、そのフィールドのイベントセクションを単独で実行するように指定することができます。このセクションの実行シーケンスの図は、フィールドレベル イベントセクションの即時の実行を表示しています。

前フィールド操作には、以下の実行シーケンスがあります。



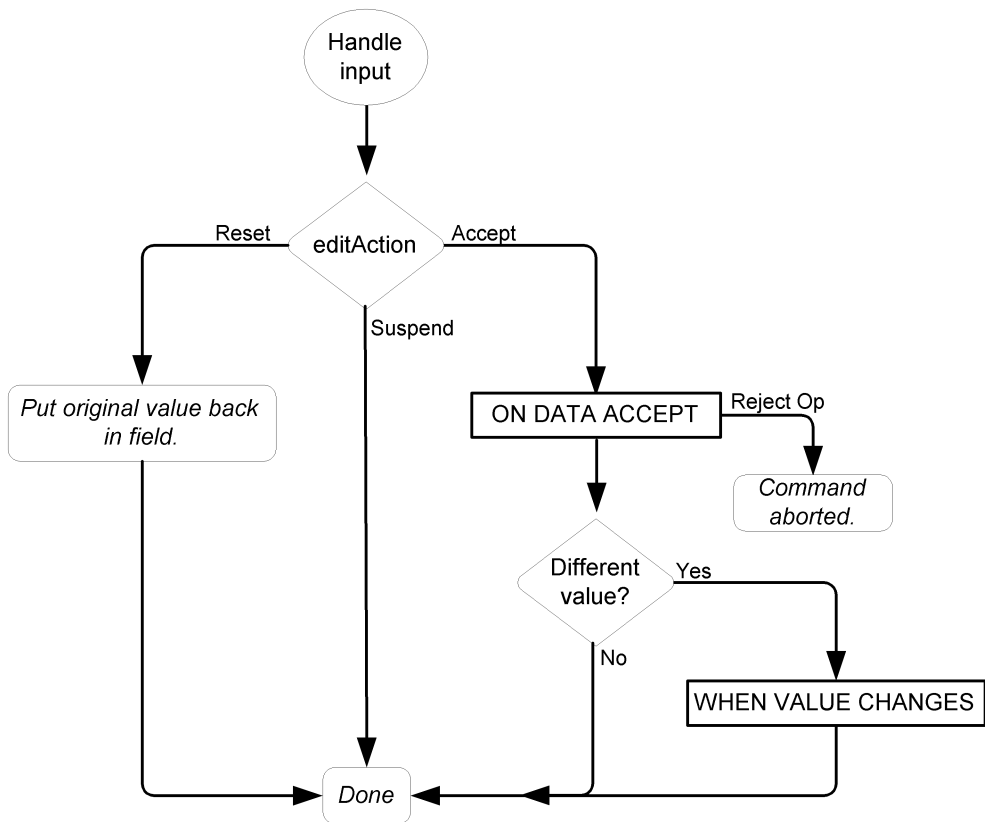
- カレントフィールドの置かれているデータビューが検索モードであるか確認します。

- 
- データビューが検索モードでなければ、NXJ インタラクションサーバはフォームスクリプトの AFTER FIELD イベントセクションを実行します。
  - NXJ インタラクションサーバは position-to-field 操作を実行し、前フィールドにフォーカスを移動します。[114 ページの「フィールド位置」](#)を参照してください。

## ユーザ入力の処理

いくつかの操作では、ユーザは、フォームのカレントフィールドに対して入力を行うことができます。フィールドの Immediate プロパティが設定されている場合、このユーザ入力は以下で説明されるように処理されます。Immediate プロパティが設定されていない場合は、フィールドレベル操作は、次レコードナビゲーションコマンドやフォームナビゲーションコマンドが実行されたときのような特定のチェックポイントで一括処理されます。繰り返し領域にあるレコードの場合には、これらのチェックポイントは Batch Record Changes プロパティの設定に依存します。

ユーザ入力処理操作には、以下の実行シーケンスがあります。



- 関連するコマンドの editAction プロパティ (NXJCommand オブジェクト) が確認されます。

NXJ システムコマンドの editAction プロパティのデフォルトの設定は次の通りです。

表 5-4 EditAction プロパティの設定

システムコマンド	editAction デフォルト設定
ADD_UPDATE	EditAction_ACCEPT
CANCEL_ZOOM	
CLEAR_FIELD	
FIND	
NEXT_FIELD	
NEXT_FORM	
FIRST_RECORD	
LAST_RECORD	
NEXT_RECORD	
NEXT_SET	
PREVIOUS_FIELD	
PREVIOUS_RECORD	
PREVIOUS_SET	
CLEAR_TO_ADD	EditAction_RESET
CLEAR_TO_FIND	
DELETE	
EXIT	
PREVIOUS_FORM	
ZOOM	EditAction_SUSPEND

システムコマンドの editAction プロパティの設定は、以下のように変更することができます。

```
BEFORE_FORM
{
NEXT_RECORD.editAction=NXJCommand.EditAction_RESET;
}
```

開発者定義コマンドでは、editAction プロパティはコマンドをコーディングするときに指定できます。editAction プロパティのデフォルト値は EditAction\_ACCEPT です。上に示すようにフォームスクリプトの設定を変更することができます。

- editAction の値が “EditAction\_ACCEPT” の場合、NXJ インタラクションサーバはユーザのデータ入力を受け付けて処理します。

- フォームスクリプトの ON DATA ACCEPT イベントセクションが実行されます。
- 値がフィールドの値と異なる場合は、フォームスクリプトの WHEN VALUE CHANGES イベントセクションが実行されます。
- editAction の値が “EditAction\_RESET” の場合、前の値があれば、NXJ インタラクションサーバは、ユーザが入力する前の値に戻します。
- editAction の値が “EditAction\_SUSPEND” の場合、NXJ インタラクションサーバはユーザの入力を一時保存し、入力処理操作が完了します。ただし、ユーザの入力は後で処理されます。

『Unify NXJ 開発者ガイド』の第 6 章「コントロール」で「イベントセクションの即時実行」を参照してください。

## ユーザ入力のプロンプト

ダイアログボックスのプロンプトでアプリケーションユーザに条件を提示する NXJ プログラミング言語文を使用することができます。ユーザのプロンプトに対する応答に基づいて、1 つ以上の NXJ プログラミング言語文は実行されます。

例えば、フォームが、選択セットを表示しているとき、アプリケーションユーザが、レコードの 1 つを変更しているかもしれません。このようなときに右のように表示するダイアログボックスで条件を提示するために NXJ プログラミング言語文を以下の様に使用します。



```
ON NEXT RECORD
{
  if ( ! isCurrentRecordStored() )
  {
    if ( session.messageBoxPrompt("save changes?") )
      updateCurrentRecord();
  }
}
```

ユーザがカレントレコードを離れる別の方法があるので、同じ NXJ プログラミング言語が、ON PREVIOUS RECORD イベントセクションに追加されなければなりません。

---

ON NEXT RECORD イベントセクションがカレントレコードに対して実行される  
とき、ダイアログボックスは表示されます。ユーザが、プロンプトに  
応答できるように NXJ インタラクショナルサーバは、実行を休止  
します。応答した後に、アプリケーションは、**messageBoxPrompt**  
メソッドが実行されたポイントから実行を継続します。

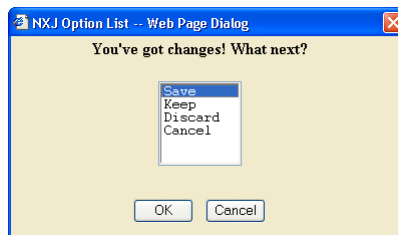
NXJSession ファウンデーションクラスは、プロンプトを定義するために以下のメソッドを提供します。

- **messageBoxPrompt()**  
“OK” と Cancel デフォルトボタンを持ったダイアログボックスを提供します。  
ユーザが“OK” ボタンを押下すると、メソッドは true を返します。プロンプト  
メッセージは、表示される文字列を表します。
- **optionListPrompt()**, **optionButtonPrompt()**, **optionPrompt()**  
ダイアログボックスに表すメッセージ選択肢をカスタマイズすることができます。  
状況に応じて、タイムアウト値が設定できます。
- **convertStringToHtml()**

これらのメソッドの詳しい説明については、Javadoc を参照してください。

このプロンプトは、**optionListPrompt** メソッドを使う事によって複数の応答を与えるためのカスタマイズがなされています。

このメソッドは、ユーザに選択肢に示したい  
いずれかの1つを選択する事ができるリスト  
スタイルプロンプトで選択肢を提供しています。  
このメソッドは、ユーザによって選択された  
選択肢のインデックスを返します。



ユーザは、リストボックスの選択肢でダブルクリックするか、  
選択肢を選んで、“OK” ボタンをクリックするかのどちらかによって  
選択肢を選べます。

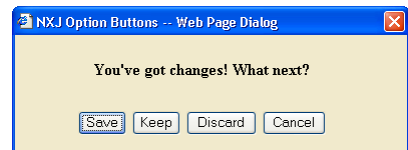
---

以下は、NXJプログラミング言語文の例です。

```
ON NEXT RECORD
{
  if ( ! isCurrentRecordStored() )
  {
    String[] optionArray = new String[4];
    optionArray[0] = "Save";
    optionArray[1] = "Keep";
    optionArray[2] = "Discard";
    optionArray[3] = "Cancel";
    switch ( session.optionListPrompt(
      "You've got changes! What next?", optionArray, 0 ) )
    {
      case 0: // save
      case 1: // keep
        updateCurrentRecord();
        break;
      case 2: // discard
        break;
      case 3: // cancel
      case -1: // X in title bar
        rejectOperation();
        break;
    }
  }
}
```

このプロンプトは、**optionButtonPrompt** メソッドを使ってボタンとして提供されています。

例：



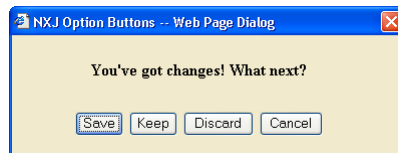
```
ON PREVIOUS RECORD
{
  if ( ! isCurrentRecordStored() )
  {
    String[] optionArray = new String[4];
    optionArray[0] = "Save";
    optionArray[1] = "Keep";
    optionArray[2] = "Discard";
    optionArray[3] = "Cancel";
    switch ( session.optionButtonPrompt(
      "You've got changes! What next?",
```

```

optionArray, 0) )
{
case 0: // save
case 1: // keep
    updateCurrentRecord();
    break;
case 2: // discard
    break;
case 3: // cancel
case -1: // X in title bar
    rejectOperation();
    break;
}
}
}

```

カスタムプロントダイアログは、**optionPrompt** メソッドを使用することで提供できます。url 引数は、openURL メソッドと同じ方法で指定します。カスタムプロンプトページを指定しなければなりません。



メソッドは、ユーザによって選択された選択肢のインデックスを戻します。ユーザが、タイトルバーの X を押下したときは、メソッドは、-1 を返します。プロンプトメッセージは、HTML マークアップ (フォーマットを目的として) を含めてもかまいません。

Unify NXJ に組み込まれたプロンプトダイアログファイルの 1 つで開始することにより、簡単にカスタムプロンプトページを作成することができます。

```

<UNIFY_HOME>%web-app%NXJOptionList.jsp
<UNIFY_HOME>%web-app%NXJOptionButtons.jsp

```

プロジェクトの Static\_Content フォルダにファイルのコピーを置いて、必要に応じてカスタマイズしてください。

NXJOptionList.jsp の JavaScript は、**optionPrompt** メソッドに渡された選択肢に基づいてリストボックスを表示します。NXJOptionButtons.jsp は、渡された選択肢によって既存のボタンを表示します。選択肢よりも多くボタンがある場合、残りのボタンは、取り除かれます。ボタンよりも多く選択肢がある場合、新しいボタンが、残りの選択肢のために作成されます。

---

以下の NXJ プログラミング言語文は、プロンプトを定義するために “myFavoriteButtonPrompt” ファイルを使用しています。

ON PREVIOUS RECORD

```
{
  if ( ! isCurrentRecordStored() )
  {
    String[] optionArray = new String[4];
    optionArray[0] = "Save";
    optionArray[1] = "Keep";
    optionArray[2] = "Discard";
    optionArray[3] = "Cancel";
    switch ( session.optionPrompt(
      "myFavoriteButtonPrompt.jsp",
      getLocalizedMessage(),
      optionArray, 0 ) )
    {
      case 0: // save
        case 1: // keep
          updateCurrentRecord();
          break;
      case 2: // discard
          break;
      case 3: // cancel
        case -1: // X in title bar
          rejectOperation();
          break;
    }
  }
}
```

プロンプトの文字列は、optionListPrompt()、optionButtonPrompt() と optionPrompt() メソッドにとって HTML として解釈します。HTML にとって適切に特定の文字 (例えば、<,>,&) を文字列でリテラルに表示するには、convertStringToHtml メソッドを使う使う必要があります。

**messageBoxPrompt** メソッドについての詳細は、NXJ Javadoc の NXJSession ファウンデーションクラスを参照してください。

# フォームスクリプトの デバッグ

## 6

NXJ Debugger は、NXJ アプリケーション用の対話型デバッグツールです。NXJ デバッガは、NXJ インタラクションサーバと動作するため、動作中の NXJ アプリケーションの実行をモニタ、管理することができます。

NXJ アプリケーションが、予想外の結果を示したとき、選択したポイントで会話的にアプリケーションの実行を一時停止するために NXJ デバッガを使用することができます。アプリケーションを停止している間に、アプリケーションデザイナーのデバッガパネルでアプリケーションのフォーム、スクリプトや変数の値を調べることができます。デバッグ目的に変数の値を作成したり出力したりする必要はありません。

例えば、NXJ デバッガで以下のようなことが実行できます。

- NXJ プログラミング言語変数の値の表示や変更
- フォーム、データビュー、コマンド、フィールドプロパティの値の表示や変更
- 実行を停止するブレークポイントの設定
- ステップや継続コマンドによる NXJ アプリケーションの実行を制御
- どのフォームが現在アクティブであるかを認識
- どのメソッドまたはイベントセクションが、最新のユーザリクエストに対して現在アクティブであるかを認識

NXJ デバッガは、NXJ チュートリアルレッスンの 8 で実際に試してみることができます。

---

## デバッグのプロセス

NXJ デバッグは アプリケーションデザイナ内で実行されます。

---

**注** - 以前にカレントのものとは異なるアプリケーションサーバと一緒に NXJ デバッグを実行している場合、デバッグサーバを停止させた後、新しいデバッグサーバを開始しなければなりません。アプリケーションサーバを開始、または停止するためには、アプリケーションサーバのツールを使用します。

---

一般的なデバッグのプロセスは、以下のとおりです。

### タスク 1: デバッグアプリケーションサーバが開始されていることを確認する

NXJ デバッグはそれ自身のアプリケーションサーバのインスタンス（コントロールセンタによって使用されるものとは、異なるアプリケーションサーバのインスタンス）を使用します。JBoss の場合のアプリケーションサーバの開始についての詳細は、『Unify NXJ 管理者ガイド』の第 2 章を参照してください。

### タスク 2: フォームがコンパイルされていることを確認する

デバッグ対象のフォームスクリプトがコンパイルされていなければ、Make コマンドを使ってコンパイルします。

### タスク 3: 最初のブレークポイントを設定する

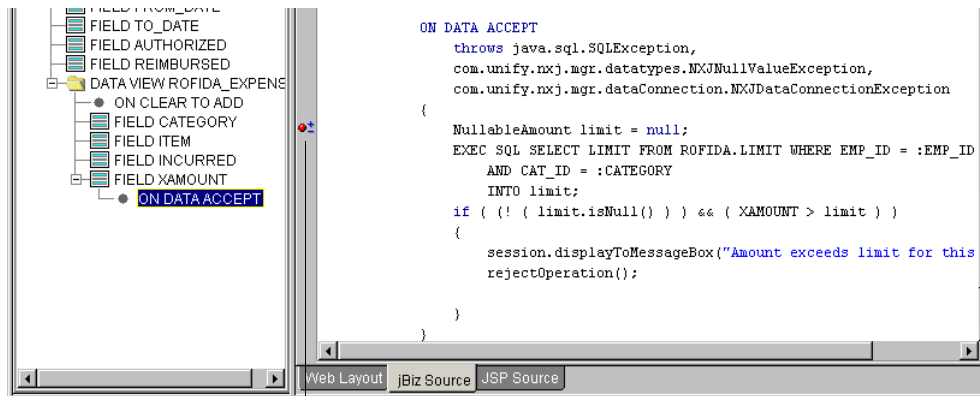
ビジネスロジックが予想外の動作をしているフォームスクリプトの場所に、ブレークポイントを設定します。ブレークポイントは、NXJ プログラミング言語スクリプトのすべての実行可能行に設定できます。

ブレークポイントを設定するには、スクリプトエディタでフォームスクリプトを開きます。ブレークポイントを設定する行にカーソルを置き、**デバッグ > ブレークポイントの ON/OFF** を選択するか、ブレークポイントの ON/OFF ボタンをクリックします。



ブレーク  
ポイントの  
ON/OFF

ブレークポイントが設定されると、スクリプトエディタ ウィンドウの左側にブレークポイントのマーカが表示されます。



ブレークポイントマーカー

## タスク 4: デバッグ > 開始 を選択して NXJ デバッガを開始

NXJ インタクションサーバは、デバッグしない場合と同じようにアプリケーションを起動します。エントリフォームが表示されるか、複数のエントリフォームがある場合はメニューが表示されます。

初めに**開始** ボタンをクリックした後、ドロップダウンリストからフォーム名を選択することで任意に、プロジェクトのシングルフォームを NXJ デバッガで実行することができます。

## タスク 5: アプリケーションの実行を調査

フォーム内を移動し、デバッグ対象となるイベントを指定します。例えば、検索操作をモニタしたい場合、関連するイベントセクションにブレークポイントを設定し、いくつかの必要な検索条件を入力し、**検索** ボタンをクリックします。

ブレークポイントが設定された行になると実行が停止します。ブレークポイントで実行が停止すると、[131 ページの「アプリケーションデータの表示」](#)で説明されている方法でアプリケーションデータを調べることができます。

**デバッグ > 継続** コマンドを選択し、さらに次のブレークポイントまで実行します。






継続

実行が停止している間、スクリプトエディタウィンドウに追加のブレークポイントを設定することができます。

また、スクリプトの実行中にアプリケーションを1ステップずつ実行することもできます。アプリケーションを1ステップずつ実行するには、以下のコマンドを使用します。

表 6-1 ステップコマンド

コマンド	目的
 ステップ イン	次の実行可能 NXJ プログラミング言語文まで実行を継続します。
 ステップ オーバー	次の実行可能 NXJ プログラミング言語文まで実行を継続しますが、この NXJ プログラミング言語文がメソッド呼び出しの場合は、メソッドに介入せずに実行されます。
 ステップ アウト	カレントのイベントセクションやメソッドの終了まで実行を継続します。

## タスク 6: デバッグ セッションの終了



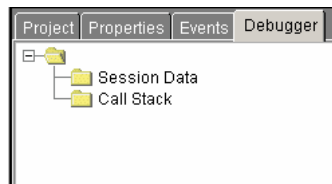
デバッグセッションは、**デバッグの中止** ボタンをクリックすると終了します。

デバックの  
中止

## アプリケーションデータの表示

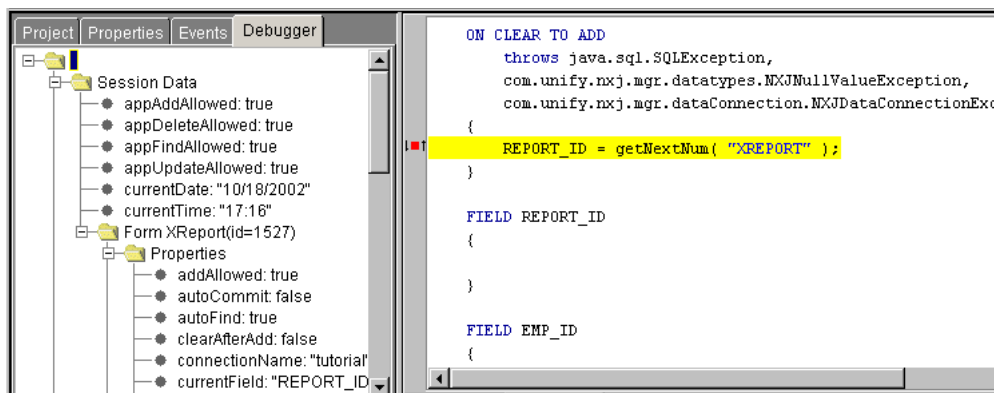
実行が停止されると、ブラウザパネルのデバッガビューに次の 2 つのフォルダが表示されます。

- Session Data
- Call Stack

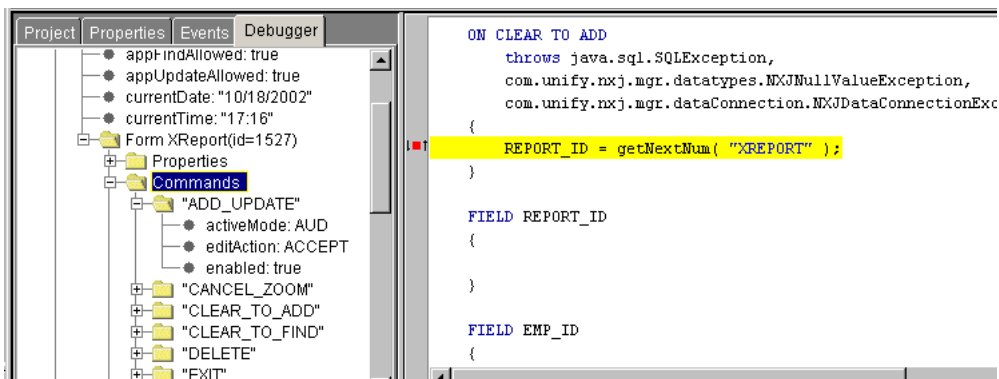


Session Data フォルダは、フォームスタックの各フォーム用のフォルダが含まれます。これはユーザーセッションで初期化されたフォームです。最初に初期化されたフォームはエントリポイントのフォームでフォームエントリの一番上であり、最後に初期化されたフォームが一番下にあります。

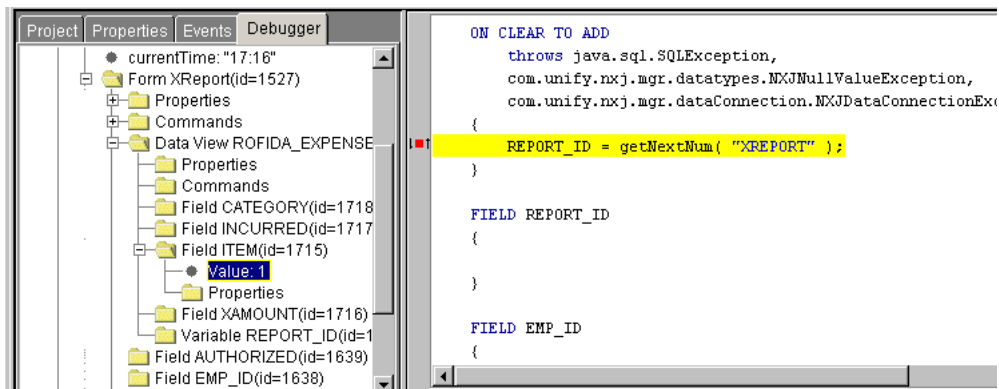
各フォームフォルダで、フォームプロパティ、コマンド、データビューやフィールドクラス、任意の開発者定義 Java あるいは NXJ 変数を見ることができます。(また、同じ名前前のフォルダにグループ化されます)。そのフォルダを開いて、そこに納められた値を見ることができます。例えば、フォーム XReport の Properties フォルダにはフォームのカレントプロパティ設定が含まれています。



Commands フォルダには、すべての定義済みコマンドと開発者定義コマンドが含まれています。



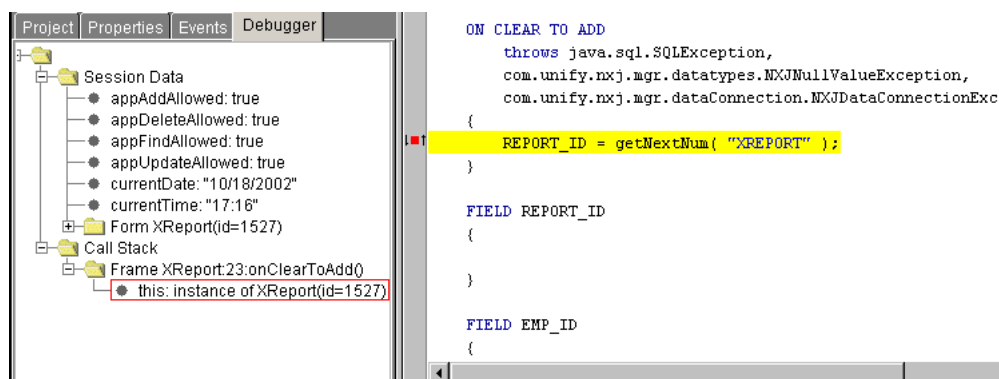
Field フォルダには、フィールドの値とフィールドプロパティの設定値が含まれています。



Call Stack フォルダは、フォームに出された最も最近のユーザリクエスト（検索、新規のような）のサードパーティ実行スタックを表示します。Call Stack の各エントリは、呼び出しフレームです。call stack は、スタック形式です。つまり、最後に呼び

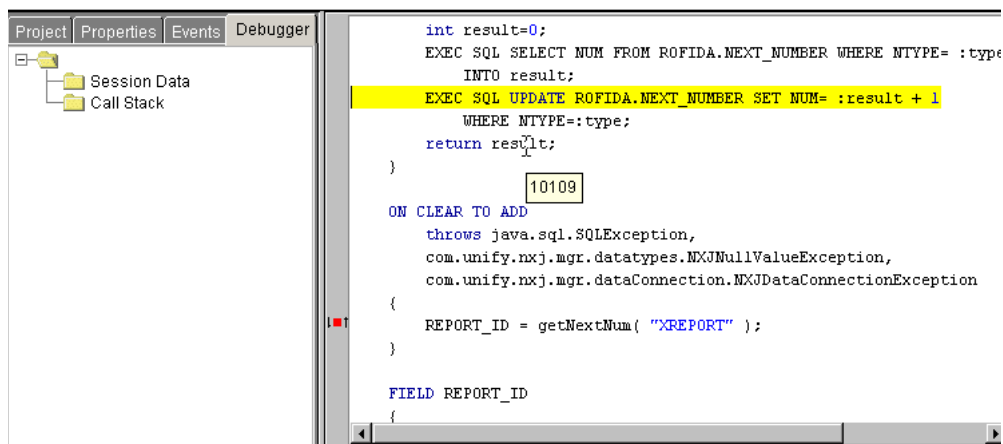
出されたメソッドが一番上に表示され、その下に呼び出し元がエントリとして表示されます。さらにその呼び出し元はその下のエントリという具合に表示されていきます。

Call Stack をクリックすると最後のリクエストの呼出しフレームがすべて表示されます。エントリには、フォーム名、行番号、実行コントロールが現在停止しているメソッド名が表示されます。フレームノードをクリックするとフレームが展開し、ローカル変数、Java インスタンス変数（static メソッド以外では“this”オブジェクト）などの変数が現れます。一番下のエントリ（エントリが1つだけの場合は一番上のエントリと同じもの）は最後にアクティブ化されたフォームのエントリで、実行が停止している行が表示されます。



編集可能なフォームのプロパティ、データビューまたはフィールドをツリーに表示して、その上から新しい値を入力することで別の値を設定することができます。さらに、Java 基本型変数 (final 宣言されていない) とフォームフィールド変数も新しい値を入力することで設定できます。Date フィールドに文字列を入力するなど間違った値を入力すると検証エラーが報告されます。

Java 基本型変数の値は、スクリプトエディタ ウィンドウでも確認できます。スクリプトエディタの変数名にカーソルを置くと、変数の値がツールチップで表示されるので、ここで確認します。



The screenshot shows a Java IDE window with a code editor. The code editor contains the following code:

```
int result=0;
EXEC SQL SELECT NUM FROM ROFIDA.NEXT_NUMBER WHERE NTTYPE= :type
      INTO result;
EXEC SQL UPDATE ROFIDA.NEXT_NUMBER SET NUM= :result + 1
      WHERE NTTYPE=:type;
return result;
}
ON CLEAR TO ADD
  throws java.sql.SQLException,
  com.unify.nxj.mgr.datatypes.NXJNullValueException,
  com.unify.nxj.mgr.dataConnection.NXJDataConnectionException
{
  REPORT_ID = getNextNum( "XREPORT" );
}
FIELD REPORT_ID
{
```

The line `return result;` is highlighted in yellow. A tooltip is displayed over the variable `result`, showing the value `10109`.

# データタイプマッピング

## A

Unify NXJ では、ターゲット変数はデータベースのデータタイプから対応する Unify NXJ データタイプに変換されます。以下の表は、サポートされる各データベースに対応するデフォルトのマッピングが示されています。

表 A-1 DB2 データタイプマッピング

DB2 データタイプ	Unify NXJ データタイプ
Decimal (n, 2)	Amount
BigInt Decimal Integer SmallInt	Numeric
Double Real	Float
Date Huge Date	Date
Time	Time
TimeStamp	DateTime
Char Varchar	String
Char for bit data Varchar for bit data Long Varchar for bit data BLOB	Binary
Long Varchar CLOB	Text
List Set Multiset	Not mapped

表 A-2 IBM Informix Dynamic Server データタイプマッピング

Informix データタイプ	Unify NXJ データタイプ
Money	Amount
Integer	Numeric
Numeric	
Serial	
Small Int	
Decimal	Float
Float	
Real	
Small Float	
Date	DateTime
DateTime	
Interval	String
NVarchar	
NChar	
char	
Varchar	
Byte	Binary
BLOB	
Text	Text
CLOB	
Set	Not mapped
List	
Multiset	

表 A-3 MS SQL データタイプマッピング

MS SQL データタイプ	Unify NXJ データタイプ
Money SmallMoney Decimal(n,2) Numeric (n,2)	Amount
Decimal Int Numeric SmallInt Tinyint BigInt	Numeric
Real Float	Float
SmallDateTime DateTime	DateTime
char Varchar	String
Bit	Bool
Binary Image Varbinary	Binary
Text	Text
Timestamp	Not mapped

表 A-4 ORACLE データタイプマッピング

ORACLE データタイプ	Unify NXJ データタイプ
NUMBER(X,2)	Amount
NUMBER(X,0) DECIMAL INTEGER, SMALLINT	Numeric
NUMBER(X,Y)	Float
DATE (date-time)	DateTime
No default conversion to TIME	Time
(Char) 1 VARCHAR, VARCHAR2 (Version 7 only) <sup>1</sup>	String
BLOB LONG RAW RAW	Binary
CLOB LONG RAW	Text

<sup>1</sup> ORACLE バージョン 6 の CHAR データタイプと ORACLE バージョン 7 の VARCHAR データタイプは同等です。

表 A-5 Sybase データタイプマッピング

Sybase データタイプ	Unify NXJ データタイプ
SmallMoney Money	Amount
SmallInt Decimal(x, 0) Int TinyInt Numeric(x, 0)	Numeric
Decimal(x, y) Numeric(x, y) Real Double Precision Float	Float
SmallDatetime Datetime	DateTime
Char nchar nvarchar Varchar	String
Bit	Bool
Text	Text
Timestamp	Not mapped
Varbinary Binary	Binary
Image	Binary (2 GB)

表 A-6 Unify DataServer データタイプマッピング

Unify DataServer データタイプ	Unify NXJ データタイプ
AMOUNT HUGE AMOUNT CURRENCY	Amount
FLOAT	Float
Numeric(x) SMALLINT INTEGER HUGE INTEGER	Numeric
DATE, HUGE DATE	Date
TIME	Time
CHAR(n)	String
TEXT	Text
BINARY	Binary