








# 4D v15 R5 - アップグレード

-  デザインモード
-  ランゲージ
-  4D Write Pro
-  4D Server
-  4D Developer Edition 64-bit版
-  最適化

## デザインモード

 アップグレード時のインデックスの再構築

## アップグレード時のインデックスの再構築














---

4D v15 R5では新しいバージョンのICUライブラリが組み込まれています。このライブラリは、文字列比較に加えて、様々な言語特性に則ったテキストキーワードを管理するために、4Dアプリケーション全体で使用されます。より詳細な情報については、[ICU Web site](#)を参照して下さい。

結果として、4Dデータベースを4D v15 R5へとアップグレードすると、4Dテキストとキーワードのインデックスが再構築されます。このオペレーションはデータベースが最初に開かれたときに自動的に実行されます(しばらく時間がかかる場合があります)。

**注:** その一方で、データベースを4D v15 R4以前にダウングレードしても、テキストとキーワードのインデックスは自動的に再構築されます。

## ランゲージ

-  プロセス間のメッセージ通信
-  CALL FORM
-  CALL WORKER
-  Command name
-  Current client authentication
-  Current process name
-  DELETE FOLDER
-  Get current printer, SET CURRENT PRINTER
-  GET PRINT OPTION, SET PRINT OPTION
-  KILL WORKER
-  OBJECT SET FORMAT, OBJECT Get format
-  PROCESS PROPERTIES
-  廃止されたコマンド

### 概要

4D v15 R5 では、簡単かつ強力なプロセス間および、プロセス-フォーム間通信の方法を新しく導入しました。この機能は非同期のメッセージシステムに基づいており、プロセスやフォームを呼び出して、呼び出し先のコンテキストで任意のメソッドを指定パラメーターとともに実行させることができます。

- **ワーカープロセス:** プロセス間通信は**ワーカー**という新しい種類のプロセスでもって管理することができます。新しい **CALL WORKER** コマンドによって、あらゆるプロセスはワーカープロセスを "雇用" ことができ、ワーカーのコンテキストにおいて任意のプロジェクトメソッドを指定のパラメーターで実行させることができます。つまり、情報の共有が可能です。
- **フォーム:** 新しい **CALL FORM** コマンドを使えば、フォームのコンテキストにおいて、任意のプロジェクトメソッドを指定のパラメーターで実行させることができます。

これらの新機能は、4D のプロセス間通信における次のニーズに対応します:

- コオペラティブおよびプリエンティブ・プロセスの両方に対応しているため、インタープロセス変数が使えないプリエンティブ・プロセスにおけるプロセス間通信に最適です。
- 煩雑になりやすいセマフォの代替手法として使用できます。

**CALL WORKER** および **CALL FORM** のコマンドは主に、64-bit版で提供されているプリエンティブ・プロセスのコンテキストにおけるプロセス間通信のために開発されましたが、これらは32-bit版でも提供されており、コオペラティブ・プロセスにおいても同様に使用することができます。

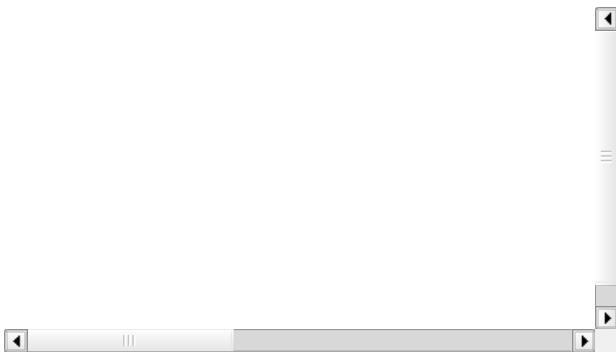
### ワーカーの使用

**ワーカー** は、プロジェクトメソッドの実行をプロセスに依頼するのに使われます。ワーカーには次のものが付随します:

- ワーカーを特定する一意の名称
- 関連プロセス (特定の時点において存在しない場合もあります)
- メッセージボックス
- 初期メソッド (任意)

**CALL WORKER** コマンドを使って、プロジェクトメソッドの実行をワーカーに依頼します。初めてワーカーを使用するとき、ワーカーはメッセージボックスとともに作成され、関連プロセスを実行します。ワーカープロセスが終了してもメッセージボックスは開いたままで、新しいメッセージを受け取ると新しくワーカープロセスを開始します。

この一連の流れをアニメーションで表しました:



**New process** コマンドとは異なり、ワーカープロセスはメソッドの実行後も生きています。つまり、ワーカーが実行するすべてのメソッドは同一のプロセス内でおこなわれ、すべてのプロセス情報 (プロセス変数、カレントレコード、カレントセレクション、など) が保持されます。続けて実行されるメソッドはこれらの情報を共有することになるため、プロセス間の通信が可能になります。ワーカーのメッセージボックスは連続した呼び出しを非同期的に扱います。

**CALL WORKER** はメソッド名と引数をカプセル化し、メッセージとしてワーカーに受け渡します。ワーカープロセスがまだ存在していなければ開始され、メッセージボックスに格納されたメッセージを実行します。したがって、大体的場合において

**CALL WORKER** は、ワーカーが受け取ったメソッドを実行するより先に終了します (非同期的な実行)。そのため **CALL WORKER** は戻り値を返しません。実行後の情報をワーカーから返してもらうには、**CALL WORKER** を利用してワーカーの呼び出し元に情報を渡す必要があります (コールバック)。これには、呼び出し元のプロセスもワーカーである必要があります。

**New process** コマンドで作成されたプロセスはメッセージボックスを持たないため、**CALL WORKER** によってワーカーとして使うことはできません。そのため、同プロセスがワーカーをコールすることは可能でも、**CALL WORKER** によるワーカーからのコールバックを受けられないことに留意が必要です。

ワーカープロセスは、ストアードプロシージャを使って 4D Server 上に作成することもできます。例えば、**CALL WORKER** コマンドを実行するメソッドを **Execute on server** コマンドから実行します。

ワーカープロセスを閉じるには **KILL WORKER** コマンドをコールします。これによってワーカーのメッセージボックスは空にされ、関連プロセスはメッセージの処理を停止し、現在のタスク完了後に実行を終了します。



ワーカープロセスを開始するために指定したメソッドがワーカーの初期メソッドになります。 *method* パラメーターに空の文字列を受け渡した場合には、**CALL WORKER** によってこの初期メソッドが再度実行されます。

ユーザーおよびアプリケーションモードで 4D データベースを開く際に作成されるメインプロセスはワーカーです。したがって、**CALL WORKER** で呼び出すことができます。メインプロセスの名称は 4D の使用言語により異なりますが、プロセス番号は常に 1 です。**CALL WORKER** でメインプロセスを呼び出す場合には、プロセス番号を使うのが便利でしょう。

## ワーカープロセスの識別

**PROCESS PROPERTIES** コマンドを使った場合、メインプロセス以外のすべてのワーカープロセスは Worker process (5) をプロセスの種別として返します (メインプロセスは Main process (-1) です)。

ランタイムエクスプローラーおよび 4D Server の管理画面においては、アイコンからもワーカープロセスを識別することができます:

プロセスタイプ	アイコン
プリエンプティブ・ワーカープロセス	
コオペラティブ・ワーカープロセス	

注: ほかの既存プロセスのアイコンも 4D v15 R5 で更新されています。

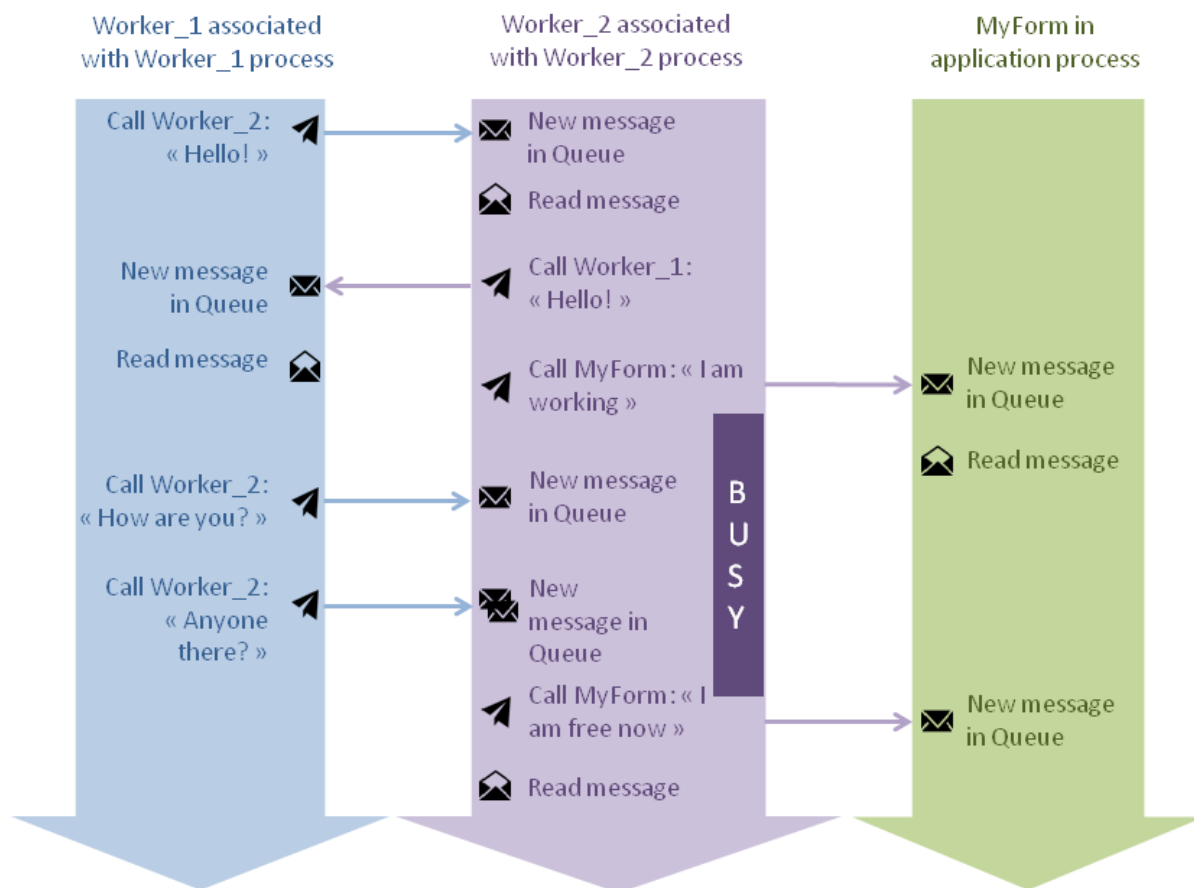
## フォームの呼び出し

**CALL FORM** コマンドを使って、ウィンドウに表示されたフォームのコンテキストにおいて、プロジェクトメソッドを実行させることができます。この場合、どのプロセスが そのフォームを持っているかは、問題になりません。この呼び出しはコオペラティブおよびプリエンプティブ・プロセスの両方に対応しているため、この機能をプリエンプティブ・プロセスにおけるプロセス間通信に利用することができます。

ウィンドウ内で実行されるすべてのフォームに、メッセージボックスが実装されました。メッセージボックスの中身はウィンドウがフォームを表示したとき (On Load フォームイベントの後) に処理されます。**CALL WORKER** と同様に、**CALL FORM** はメソッド名と引数をカプセル化し、メッセージの形でフォームのメッセージボックスに受け渡します。フォームは自身のプロセスにおいて、そのメッセージを実行します。

## 使用の原理

二つのワーカーと一つのフォーム間の通信の流れを図に表しました。通信の内容は文字列です:



1. Worker\_1 が Worker\_2 を呼び出し、"Hello!" を渡します
2. Worker\_2 がメッセージを受信し、内容を確認します。Worker\_2 が Worker\_1 をコールバックして、"Hello!" を返します。
3. Worker\_2 は次にフォーム MyForm を呼び出し、"I work" を渡します。Worker\_2 はそのまま時間の掛かる処理を始め、ステータスが 'busy' になります。
4. Worker\_1 が Worker\_2 を二回呼び出しますが、非同期的なメッセージ・システムのため、メッセージはキューに入ります。これらは Worker\_2 が継続中の処理を終了し、MyForm を呼び出したあとに処理されます。

CALL FORM ( window ; method {; param}{; param2 ; ... ; paramN} )

引数	型		説明
window	WinRef	→	ウィンドウ参照番号
method	テキスト	→	フォームで実行させるプロジェクトメソッド名
param	式	→	メソッドに渡す引数

## 説明

### テーマ: フォーム

**CALL FORM** コマンドは、*window* に指定したウィンドウに表示されているフォームのコンテキストにおいて、任意の *param* パラメーターを使った *method* の実行を要求します。どのプロセスがそのフォームを持っているかは、問題になりません。

ワーカーを利用したプロセス間通信 ([プロセス間のメッセージ通信](#) 参照) の機能は、ワーカーが持つメッセージボックスに基づいて設計されていますが、ウィンドウも同様にメッセージボックスを持っており、ウィンドウがフォームを表示した後に (On Load フォームイベントの後) に使用することができます。**CALL FORM** はメソッド名と引数をカプセル化し、メッセージの形でウィンドウが持つメッセージボックスに受け渡します。フォームは自身のプロセスにおいて、そのメッセージを実行します。

*window* には呼び出すフォームを表示しているウィンドウの参照番号を渡します。

*method* には、*window* の親プロセスのコンテキストで実行するプロジェクトメソッド名を受け渡します。

*param* パラメーターに値を受け渡すことで、一つ以上の引数をメソッドに受け渡すことができます。引数の渡し方は、サブルーチンを使う場合と同じです ([メソッドに引数を渡す](#) 参照)。フォームのコンテキストで実行を開始する際に、メソッドはこれらの引数を \$1, \$2, などの値として受け取ります。配列はメソッドのパラメーターに渡せないことに留意してください。また、**CALL FORM** コマンドを使うにあたっては、次のことに留意してください:

- テーブルやフィールドへのポインターが使えます
- ローカル変数やプロセス変数を筆頭とした、変数へのポインターの利用は、プロセスメソッドによってアクセスした時点で未定義の可能性があるため、推奨されません。
- 呼び出し先のフォームが呼び出し元とは別のプロセスに所属している場合に、オブジェクト型の引数を受け渡すと、4D は呼び出し先プロセスにそのオブジェクトのコピーを作成します。

## 例題 1

メイン・フォームに設置されたボタンをクリックすると、背景色とメッセージの異なる二つのフォームウィンドウが同時に開くようにします。また、この子ウィンドウのメッセージは、後から違う内容を送信して表示を変更できるようにします。

まず、メイン・フォームのボタンに設定するオブジェクトメソッドです:

```
// 子ウィンドウを作成し、フォームを表示させます
// 一つ目の子ウィンドウ
formRef1:=Open form window("FormMessage";Palette form window;On the left)
SET WINDOW TITLE("MyForm1";formRef1)
DIALOG("FormMessage";*)
SHOW WINDOW(formRef1)

// 二つ目の子ウィンドウ
formRef2:=Open form window("FormMessage";Palette form window;On the left+500)
SET WINDOW TITLE("MyForm2";formRef2)
DIALOG("FormMessage";*)
SHOW WINDOW(formRef2)

// 背景色の指定
CALL FORM(formRef1;"doSetColor";0x00E6F2FF)
CALL FORM(formRef2;"doSetColor";0x00F2E6FF)
```

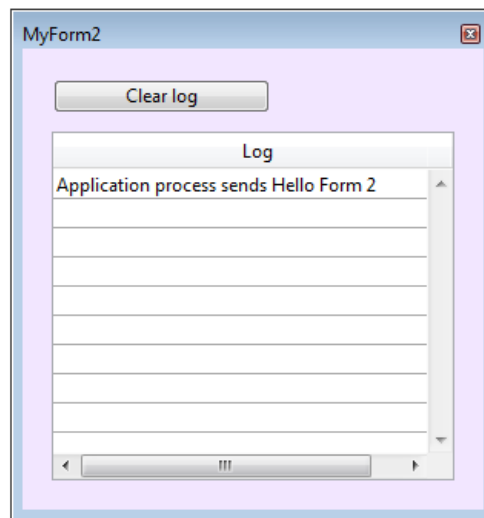
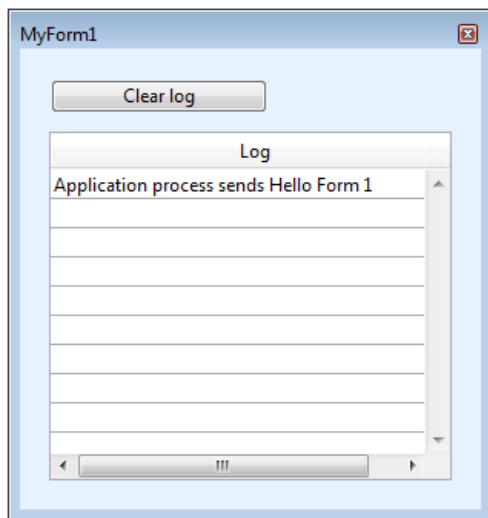


```
// メッセージの表示
CALL FORM(formRef1;"doAddMessage";Current process name;"Hello Form 1")
CALL FORM(formRef2;"doAddMessage";Current process name;"Hello Form 2")
```

doAddMessage メソッドは "FormMessage" フォームのリストボックスに行を追加します:

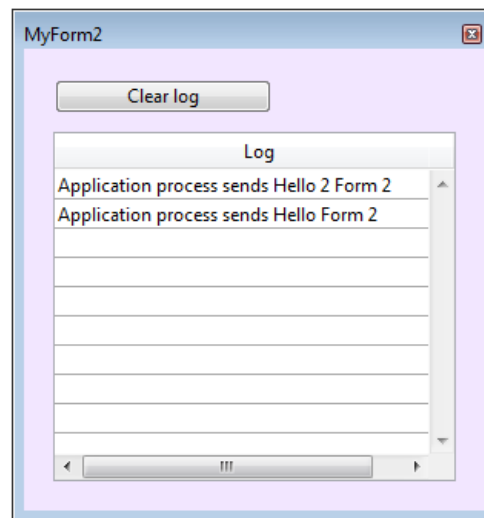
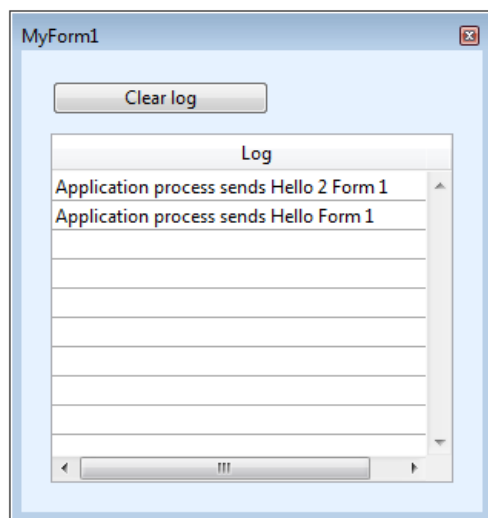
```
C_TEXT($1) // コール元のプロセス名
C_TEXT($2) // 表示するメッセージ
// $2 からメッセージを取得し、リストボックスにメッセージを記録します
$P:=OBJECT Get pointer(Object named;"Column1")
INSERT IN ARRAY($P->1)
$P->{1}:= $1+" sends "+$2
```

ランタイムでは次のような結果になります:



CALL FORM コマンドを繰り返し実行することで、メッセージを追加していくことができます:

```
CALL FORM(formRef1;"doAddMessage";Current process name;"Hello 2 Form 1")
CALL FORM(formRef2;"doAddMessage";Current process name;"Hello 2 Form 2")
```



## 例題 2

CALL FORM コマンドを利用することで、プロセス変数を使わずにフォームにカスタム設定 (規定値など) を受け渡すことができます:

```
$win:=Open form window("form")
CALL FORM($win;"configure";param1;param2)
DIALOG("form")
```

```
CALL WORKER ( process ; method {; param}{; param2 ; ... ; paramN} )
```

引数	型	説明
process	倍長整数, テキスト	→ プロセス番号、またはプロセス名
method	テキスト	→ プロセスで実行させるプロジェクトメソッド名
param	式	→ メソッドに渡す引数

## 説明

**テーマ:** プロセス (コミュニケーション)

**CALL WORKER** コマンドは、*process* に受け渡した名称または ID のワーカープロセスを作成、または呼び出して、任意の *param* パラメーターを使った *method* の実行を要求します。

**CALL WORKER** コマンドは *params* をカプセル化し、メッセージの形でワーカーが持つメッセージボックスに受け渡します。ワーカープロセスについての詳細は [プロセス間のメッセージ通信](#) を参照ください。

*process* パラメーターに指定するプロセス名またはプロセス番号により、ワーカーは特定されます:

- プロセス番号で指定したプロセスが存在しない場合、またはそのプロセスが **CALL WORKER** によって 4D が作成したものでない場合 (例えばメインのアプリケーションプロセスなど)、**CALL WORKER** は何もしません。
- プロセス名で指定したプロセスが存在しない場合には、新規のワーカープロセスが作成されます。

ワーカープロセスはランタイムエクスプローラーのプロセスリストに表示されます。また、**PROCESS PROPERTIES** コマンドはワーカープロセスも対象に実行できます。

*method* には、ワーカープロセスのコンテキストで実行するプロジェクトメソッド名を受け渡します。*method* には空の文字列を渡すこともでき、その場合ワーカーはプロセス開始時に指定されていたワーカーメソッドを実行します。プロセス開始時にメソッドが指定されていない場合には、何もしません。

**注:** メインアプリケーション・プロセスはワーカーですが、初期プロセスを持たないため **CALL WORKER (1;"/)** は何もしません。

*param* パラメーターに値を受け渡すことで、一つ以上の引数をプロセスメソッドに受け渡すことができます。引数の渡し方は、サブルーチンを使う場合と同じです ([サブルーチン](#) 参照)。プロセスのコンテキストで実行を開始する際に、プロセスメソッドはこれらの引数を \$1, \$2, などの値として受け取ります。配列はメソッドのパラメーターに渡せないことに留意してください。また、**CALL WORKER** コマンドを使うにあたっては、次のことに留意してください:

- テーブルやフィールドへのポインターが使えます
- ローカル変数やプロセス変数を筆頭とした、変数へのポインターの利用は、プロセスメソッドによってアクセスした時点で未定義の可能性があるため、推奨されません。
- ワーカーが **CALL WORKER** の呼び出し元とは別のプロセスの場合に、オブジェクト型の引数を受け渡すと、4D は呼び出し先プロセスにそのオブジェクトのコピーを作成します。

ワーカープロセスは、アプリケーションが終了するか、あるいは **KILL WORKER** をコールされるまで存続します。必要ないワーカーにはこのコマンドを使い、メモリを解放するのがよいでしょう。

## 例題

フォーム上に、選択年の統計などの算出をおこなうボタンを設置します。

ボタンはワーカープロセスを作成、あるいは呼び出します。演算はワーカーがおこなうため、ユーザーはフォームの操作を続行できます。

ボタンのメソッドは次のとおりです:

```
// ワーカー vWorkerName をコールし、実行メソッドと引数を指定します
C_LONGINT (vYear)
vYear:=2015 // この値はフォーム上でユーザーが選択したかもしれません
CALL WORKER ("myWorker"; "workerMethod"; vYear; Current form window)
```

*workerMethod* のコードは次のとおりです:

```
// ワーカーが実行するメソッドです
// プリエンプティブでもコオペラティブでも可能です
C_LONGINT ($1) // 選択年を取得します
C_LONGINT ($2) // ウィンドウの参照を取得します
C_OBJECT (vStatResults) // 演算の結果を格納する変数です
... // 統計データを算出します
// 終了後、ワーカーはフォームをコールして、結果を渡します
// vStatResults でフォーム上に結果を表示します
CALL FORM ($2;"displayStats";vStatResults)
```

## Command name

Command name ( `command {; info {; theme}}` ) -> 戻り値

引数	型	詳細
<code>command</code>	倍長整数	-> コマンド番号
<code>info</code>	倍長整数	<- コマンドのスレッドセーフティプロパティ
<code>theme</code>	テキスト	<- コマンドのランゲージのテーマ
戻り値	文字列	<- ローカライズされたコマンド名

## 説明

テーマ: ランゲージ

**Command name** コマンドは拡張され、64-bitアプリケーションでプリエンティブプロセスを使用したい場合に特に有用な追加の情報を提供するようになりました ([プリエンティブ4Dプロセス](#) を参照ください)。

二つの任意の引数を使用できるようになりました:

- `info`: コマンドのプロパティ。返される値は *bit field* であり、現在は最初の桁 (bit 0) のみが有効で、コマンドがスレッドセーフであった場合に 1 に設定されます (0 は非スレッドセーフを意味します)。プリエンティブプロセスではスレッドセーフのコマンドのみ使用する事ができます。
- `theme`: コマンドの 4D ランゲージテーマ名を返します。

## 例題

番号を渡したコマンドがスレッドセーフである場合には **true** を、それ以外の場合には **false** を返すメソッドを作成します。

```
// Is_Thread_Safe プロジェクトメソッド
// Is_Thread_Safe(numCom) -> boolean

C_LONGINT ($1;$threadsafe)
C_TEXT ($name)
C_BOOLEAN ($0)
$name:=Command name ($1;$threadsafe;$theme)
If ($threadsafe ?? 0) // 最初の桁が 1 に設定されていた場合
    $0:=True
Else
    $0:=False
End if
```

これを使用して、例えば "SAVE RECORD" コマンド (53) に対して、以下のように書くことができます:

```
$isSafe:=Is_Thread_Safe(53)
// trueを返します
```

## Current client authentication

Current client authentication {( domain ; protocol )} -> 戻り値

引数	型	説明
domain	テキスト	← ドメイン名
protocol	テキスト	← "Kerberos"、"NTLM"、または空の文字列
戻り値	テキスト	→ Windows が返すセッションユーザーのログイン名

### 説明

#### テーマ: システム環境

新しい **Current client authentication** コマンドは Windows の Active Directory サーバーにクライアントの認証を要求し、成功した場合には同クライアントの Windows ログイン名を返します。認証に失敗した場合は、空の文字列が返されません。

このコマンドの使用は、Windows 環境で 4D Server に SSO (Single Sign On) 機能を実装している場合に限りです。詳細については [Windowsでのシングルサインオン\(SSO\)](#) を参照ください。

通常はクライアントとサーバーの両方が同じ Active Directory で管理されている必要がありますが、[SSOのための必須要件](#) でも説明されているように、違う設定にも対応することができます。

Windows セッションのログインに基づいてクライアントにアクセス権を付与するには、戻り値のログイン名を 4Dの識別モジュールに受け渡します。"デフォルトユーザー"を設定することで、4D Server のログインダイアログが表示されないようにしておけば、ユーザーが ID等を再入力する必要のないインターフェースを実装することができます (例題参照)。

このコマンドは次のテキストパラメーターを返すことができます (任意):

- *domain*: クライアントが所属するドメイン名
- *protocol*: Windows がユーザー認証に使用しているプロトコル名。値は "Kerberos" または "NTLM" のいずれかです。認証が失敗した場合には、空の文字列 ("") が返されます。

ドメイン名やプロトコルによってアクセスを制限したい場合には、接続の可否を判断するのにこれらのパラメーターを利用することができます。

#### 認証のセキュリティレベル

認証のセキュリティレベル (ログインユーザーの信用度) は使用されたユーザー識別方法によります。**Current client authentication** コマンドパラメーターに返される値を使い、クライアントが何に基づいてログインしているのかを確認して、そこからセキュリティレベルを評価することができます:

Login (戻り値)	Domain	Protocol	コメント
空文字列	空文字列	空文字列	ログインユーザーの認証情報を取得できませんでした
空以外	空文字列	NTLM	戻り値はローカルマシンで定義されたローカルIDです
空以外	空以外	NTLM	戻り値は <i>domain</i> パラメーターに返されたドメインで NTLM プロトコルの認証を受けた ID です。この場合、セキュリティを向上するにはドメインを検証する必要があります。つまり、予想されるドメインでの認証をユーザーが受けているかを確認します。
空以外	空以外	Kerberos	戻り値は予想されるドメインでケルベロスプロトコルの認証を受けた ID です。この設定は一番高いセキュリティレベルを提供します。

これらの条件についての詳細は [SSOのための必須要件](#) を参照ください。

### 例題

4D のユーザーとグループの機能に基づいたアクセス管理のシステムを持つ 4D Server があります。Windows の 4Dリモー

トユーザーが、別途ログイン操作を行わなくても適切なアクセス権でもってこのサーバーに直接接続できるよう、アプリケーションを設定します:

1. データベース設定の "セキュリティ" ページで、デフォルトユーザーを設定します:

Default User:

この設定を行うと、サーバーに接続するリモートの 4D に対してパスワードダイアログが表示されず、すべてのクライアントは "Bob" としてログインします。

2. On Server Open Connection データベースメソッドに次のコードを追加し、Active Directory でユーザー認証を確認します:

```
// On Server Open Connection データベースメソッド
C_LONGINT ($0;$1;$2;$3)
$login:=Current client authentication($domain;$protocol)
If($login # "") // ログイン名が返された場合
// カスタムの認証メソッドをコールします
    $0:=CheckCredentials($login)
// 成功の場合に 0、エラーの場合には -1 を返すようにします
Else
    $0:=-1 // ログイン名が空文字列の場合は接続を拒否します
End if
```

**注:** この例で紹介した基本シナリオは、実際のアプリケーションに応用する必要があります。例えば次のような実装に基づいた、カスタムの 4D ユーザー認証メソッド (例では **CheckCredentials**) が考えられます:

- Active Directory の情報を 4D のユーザーとグループ名に複製して自動的にマッピング
- 戻り値をカスタムの [user] テーブルと照合
- LDAP を使ってユーザー情報を取得

## ⚙️ Current process name

Current process name -> 戻り値

引数	型	説明
戻り値	テキスト	カレントプロセス名

### 説明

---

**テーマ:** プロセス

**Current process name** コマンドは、このコマンドを呼び出したプロセスのプロセス名を返します。

このコマンドはワーカープロセスのコンテキストにおいて有用です ([プロセス間のメッセージ通信](#) の章を参照)。これを使用すると汎用的なコードを書く際に呼び出すワーカープロセスを特定することができます。

### 例題

---

ワーカーを呼び出し、呼び出し元のプロセス名を引数として渡します:

```
CALL WORKER(1;"myMessage";Current process name;"Start :"+String(vMax))
```

## DELETED FOLDER

DELETE FOLDER ( folder {; *deleteOption* } )

引数	型	説明
folder	String	-> 削除するフォルダーの名称またはフルパス
<i>deleteOption</i>	Longint	-> フォルダー削除オプション

### 説明

テーマ: システムドキュメント

**DELETE FOLDER** コマンドで任意の *deleteOption* パラメーターを使うことで、空ではないフォルダーを削除することができますようになりました。**注:** *deleteOption* パラメーターを省略した場合、このコマンドはこれまでのバージョンと同じ動作をします。

*deleteOption* パラメーターには、**System Documents** テーマに含まれる次の新しい定数を受け渡すことができます:

定数	値	説明
Delete only if empty	0	フォルダーが空の場合のみ削除します
Delete with contents	1	フォルダーが空でなくても、格納されている要素ごと削除します

- Delete only if empty (0) を受け渡した場合:

- *folder* パラメーターに指定したフォルダーは、空の場合に限り削除されます。空でない場合には削除処理は行わず、エラー\*が発生します (これまでのバージョンと同様の動作です)。
- 指定したフォルダーが存在しない場合にもエラー\*\* が生成されます (後述の **注記** 参照)。

- Delete with contents (1) を受け渡した場合:

- *folder* パラメーターに指定したフォルダーは、格納されている要素ごと削除されます。  
**警告:** フォルダーや格納要素がロックされていたり、読み取り専用を設定されていても、カレントユーザーが処理に必要な権限を持っていれば削除されます。
- フォルダーや格納要素を削除できない場合、最初の削除不能な要素を検知した時点で処理は中断され、エラー\*\*\* が返されます。この場合、フォルダーは一部削除済みの可能性があります。処理が中断された場合には、**GET LAST ERROR STACK** を使って、問題となったファイルの名称とパスを取得できます。
- *folder* パラメーターに指定したフォルダーが存在しない場合、このコマンドは処理を行わず、エラーも発生しません。

\* -47: ファイルが既に開かれている、あるいはフォルダが空ではありません。

\*\* -120: 存在しないディレクトリを指定するパス名を使用してファイルにアクセスしようとしてしました。

\*\*\* -54: ロックされたファイルを書き込みのために開こうとしてしました。(Win) または

-45: ファイルがロックされている、あるいはパス名が不正です。(Mac)

**注:** Delete only if empty(0) オプション指定時 (あるいは省略時) に *folder* パラメーターにファイルパス、空の文字列、または存在しないフォルダーを受け渡した場合には、エラーコード -120が生成されます (存在しないディレクトリを指定するパス名を使用してファイルにアクセスしようとしてしました)。これまでのバージョンではエラーコード -43 が生成されていました (ファイルが見つかりません)。



## 📄 Get current printer, SET CURRENT PRINTER

Get current printer -> Function result

SET CURRENT PRINTER ( printerName )

### 説明

テーマ: 印刷

**64-bit のみ:** ここで紹介する新機能は 4D Developer Edition v15 R5 64-bit でのみ提供されています ([印刷アーキテクチャー\(新デザイン\)](#) 参照)。

4D Developer Edition v15 R5 64-bit において、これらのコマンドは OS X および Windows 10 環境の PDF プリンターで使用される、新しいプリンター名の値を扱います。

**SET CURRENT PRINTER** コマンドの *printerName* パラメーターには、汎用 PDF プリンターを指定するための新しい定数を受け渡すことができます。この定数は "Print Options" テーマで提供されています:

定数	型	値	説明
Generic			- OS X: カレントプリンターをデフォルトドライバーに設定します。このドライバーは <b>PRINTERS LIST</b> コマンドで返されません。あらかじめ <b>SET PRINT OPTION</b> で PDF ドキュメントパスを指定しておく必要があることに注意が必要です。指定がないと、エラー 3107 が返されます。
PDF driver	text	_4d_pdf_printer	- Windows: カレントプリンターを Windows の PDF ドライバー ("Microsoft Print to PDF") に設定します。このオプションは PDF オプションがインストールされた Windows 10 でのみ使用可能です。PDF ドライバーが提供されていない場合や、そのほかの Windows のバージョンでは、このコマンドはなにもせず、OK変数は 0 に設定されます。

このオプションが有効な場合、**Get current printer** コマンドの戻り値は "\_4d\_pdf\_printer" または実際の PDF ドライバーの名称です。

## GET PRINT OPTION, SET PRINT OPTION

GET PRINT OPTION ( *option* ; value1 {; value2} )

SET PRINT OPTION ( *option* ; value1 {; value2} )

### 説明

#### テーマ: 印刷

4D v15 R5 のいくつかの変更点は、これらのコマンドが扱うプリントオプションに影響します。変更点は次の実装に関わります (詳細についてはこれらの章を参照ください):

- 4D Developer Edition 64-bit ([印刷アーキテクチャー\(新デザイン\)](#) 参照)
- 4D Write Pro ([4D Write Proドキュメントの印刷](#) 参照)

オプション (定数)	OS	4D v15 R5 では	補足
2 (Orientation option)	Windows と OS X	4D Developer Edition v15 R5 64-bit で更新	プリントジョブ内でコールすることができ、同じプリントジョブのなかで方向を縦 / 横に変更可能です
8 (Color option)	Windows のみ	4D Developer Edition v15 R5 64-bit で廃止	廃止されました
13 (Mac spool file format option)	OS X のみ	4D Developer Edition v15 R5 64-bit で廃止	<b>SET CURRENT PRINTER</b> オプションに置き換えられました
15 (Page range option)	Windows と OS X	新規、4D Write Pro でのみサポート	印刷範囲の最初のページを <i>value1</i> (デフォルト値は 1) に、任意で最後のページを <i>value2</i> (デフォルト値は最後のページを示す -1) に指定します

```
KILL WORKER {( process )}
```

引数	型	説明
process	倍長整数, テキスト	→ 終了させるプロセスの番号または名称 (省略の場合はカレントプロセス)

## 説明

**テーマ:** プロセス (コミュニケーション)

**KILL WORKER** コマンドは *process* に指定した番号、または名称のワーカープロセスにメッセージを送信し、現在の処理が完了次第、未処理のメッセージすべて無視して実行を終了するよう命令します。

このコマンドの対象は、メッセージボックスを持つワーカープロセスに限られます。詳細については [プロセス間のメッセージ通信](#) を参照ください。

*process* には実行を終了させるプロセスの番号または名称を受け渡します。指定のプロセスが存在しない場合、**KILL WORKER** は何もしません。

**KILL WORKER** のパラメーターを省略した場合には、現在実行中のワーカーにコマンドが適用されます。つまり、**KILL WORKER** (current process) と同じ結果になります。

**CALL WORKER** によって作成されたワーカーではないプロセス (例えばメインプロセス) を終了しようとした場合には、**KILL WORKER** コマンドはワーカーのメッセージボックスを空にしますが、これによってそのワーカーは終了しません。つまり、**KILL WORKER** (1) は何もしません。

## 例題

例えばフォームなどで次のようなコードを実行し、ワーカーの終了をトリガーします:

```
CALL WORKER (vWorkerName; "theWorker"; "end")
```

ワーカーメソッド (*theWorker*) の例です:

```
//theWorker メソッド
C_TEXT($1) //パラメーター

Case of
: ($1="call") // ワーカーをコールした場合
... // 処理用のコード
: ($1="end") // ワーカーの終了を指示した場合
    KILL WORKER
End case
```

## OBJECT SET FORMAT, OBJECT Get format

OBJECT SET FORMAT ( { \* ; } object ; displayFormat )  
OBJECT Get format ( { \* ; } object ) -> Function result

### 説明

#### テーマ: オブジェクト(フォーム)

これらのコマンドはリストボックスヘッダー内のアイコンに対して特定のサポートを提供するようになりました。

*object* 引数にリストボックスヘッダーを定義する変数名あるいはオブジェクト名を渡した場合、これらのコマンドを使用してヘッダー内のアイコンのパスと位置を設定あるいは取得できるようになりました。これらの機能は、例えばカスタマイズされた並べ替えアイコンを使用したい場合に有用です。

リストボックスヘッダー内でのアイコンを設定するためには、次のシンタックスで書かれた文字列を *displayFormat* 引数に渡します:

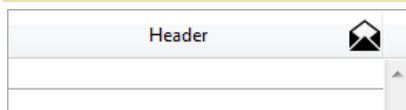
*picture;iconPos*

- *picture* = ピクチャーライブラリ、ピクチャー変数、あるいはピクチャーファイルから取得したヘッダーピクチャー:
  - ピクチャーがピクチャーライブラリから取得された場合、クエスチョンマークの後にその番号を入力します(例: "?250")。
  - ピクチャーがピクチャー変数から取得された場合、その変数名を入力します。
  - ピクチャーがデータベースの Resources フォルダ内に保存されたファイルから取得された場合、その URL を、"# {folder/} picturename" あるいは "file: {folder/} picturename" のように入力します。
- *iconPos* = ヘッダー内でのアイコンの位置。二つの値がサポートされています:
  - *iconPos* = 1: 左
  - *iconPos* = 2: 右

### 例題

データベースの Resources フォルダに "envelope\_open.png" という名前のピクチャーファイルが保存されていた場合、以下のように書くことができます:

```
vIcon:="#envelope_open.png"  
vPos:="2" // 右  
OBJECT SET FORMAT(*;"Header1";vIcon+";"+vPos)
```



## PROCESS PROPERTIES

```
PROCESS PROPERTIES ( process ; procName ; procState ; procTime {; procMode {; uniqueID {; origin}}})
```

引数	型	詳細
process	倍長整数	-> プロセス番号
procName	文字列	<- プロセス名
procState	倍長整数	<- プロセスの状態
procTime	倍長整数	<- プロセスの稼働時間 (Tick)
<b>procMode</b>	ブール   倍長整数	<- ブール値の場合: TRUE: 表示, FALSE: 非表示  倍長整数(ビットフィールド)の場合: <i>bit 0</i> = 表示状態、 <i>bit 1</i> = プリエンプティブ実行
uniqueID	倍長整数	<- ユニークなプロセス番号
<b>origin</b>	倍長整数	<- プロセスの発生源

### 説明

**PROCESS PROPERTIES** コマンドはプリエンティブモードとワーカースタイルプロセスに関連した新しい情報を返すように変更されました。

#### 新しい **procMode** の型

**procMode** (以前の名前は *procVisible*) には新しく倍長整数型の変数も使用できるようになりました。倍長整数の場合、メソッドの実行後にビットフィールドを格納しています。最初の二つのビットは以下の様な意味を持ちます:

- bit 0 は表示状態プロパティを返します。表示状態であれば 1に、非表示であれば 0に設定されています。
- bit 1 はプリエンティブモードのプロパティを返します。プロセスがプリエンティブモードで実行されている場合は 1に、コオペラティブモードで実行されている場合には 0に設定されています。

**注:** このプロパティは 64-bit版の 4Dアプリケーションでのみ有用です。64-bit版ではプロセスはプリエンティブにもコオペラティブにも実行できるからです。詳細な情報については、[プリエンティブ4Dプロセス](#) の章を参照してください。

**procMode** にブール変数を渡した場合には、以前のリリース同様にプロセスの表示・非表示に基づいて **true** または **false** を取得します。

#### **origin** の新しい値

これに加えて、**origin** には "プロセスタイプ" テーマの新しい定数が返されるようになりました:

定数	型	値	詳細
Worker process	倍長整数	5	ユーザーによって開始されたワーカースタイルプロセス

ワーカースタイルプロセスについての詳細な情報については、[プロセス間のメッセージ通信](#) の章を参照してください。

### 例題

カレントプロセスの表示状態と実行モードを知るには、以下のように書くことができます:

```
C_TEXT (vName)
C_LONGINT (vState)
C_LONGINT (vTime)
C_LONGINT (vFlags)
C_BOOLEAN (isVisible)
C_BOOLEAN (isPreemptive)
```


```
PROCESS PROPERTIES (Current process; vName; vState; vTime; vFlags)
isVisible:=vFlags?? 0 // 表示状態であれば true
isPreemptive:=vFlags?? 1 // プリエンプティブであれば 1
```


## ☰ 廃止されたコマンド

次のコマンドは 4D v15 R5 以降で改名され、4D コマンドリストには表示されなくなりました:

以前の 名前	新しい 名前	補足
C_GRAPH	<b><code>_o_C_GRAPH</code></b>	4D v14以降、グラフエリア型の変数は廃止されており、サポートされていません。代わりにピクチャー変数を使用する必要があります( <a href="#">GRAPH</a> 参照)。

# 4D Write Pro

 新しいページビューモード

 4D Write Proドキュメントの印刷



## 新しいページビューモード

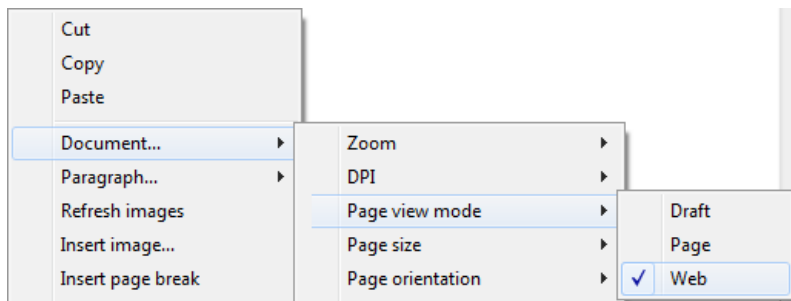
4D v15 R5以降、4D Write Proエリアではドキュメントの **ページ** ビューモードが使用できるようになります。ユーザーはこのモードで、ドキュメントがどのように印刷されるかをページとして見ることができます。

### ビューモードの選択

4D Write Proドキュメントは三種のページビューモードで表示することができます：

- **下書き**：基本的なプロパティを持つ下書きモード
- **ページ**：“印刷ビュー”モード（4D v15 R5 からの新機能）
- **Web**（デフォルト）：ページプロパティ・段落プロパティを一切持たないそのままのモード

現在の 4D Write Pro の実装では、エリアポップアップメニューを使用することでのみビューモードの選択および設定が可能です：



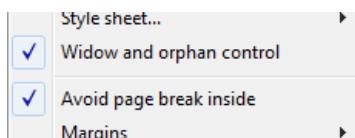
**注：**ページビューモードはドキュメントには保存されません。

### ページビュー機能

ドキュメントがページビューモードのとき、以下のドキュメントプロパティがユーザーに表示されます：

- 印刷範囲を示すページのアウトライン
- ページ幅とページの高さ（デフォルトは21x29.7 cm）
- ページの向き（デフォルト：縦向き）
- ページの余白（デフォルト：2.5cm）

これに加えて、**段落...**サブメニューでは新しい段落プロパティが利用可能です：



- **ウィドウ&オーファンコントロール**：このオプションが段落に対して設定されているとき、4D Write Pro はドキュメント内にてウィドウ（段落の最後の行がページの最上部に取り残される状態）とオーファン（段落の最初の行がページの最下部に取り残される状態）を許可しません。前者の場合には、最後の行の一つ前の行も加えて 2行がページの最上部に表示されます。後者の場合には、単一の最初の行は次のページへと送られます。
- **段落内の改ページを避ける**：このオプションが段落に対してチェックされている場合、4D Write Pro はその段落が 2枚以上のページに分割されないようにします。

また、コンテキストメニューから以下のコマンドを使用する事もできます：

- **改ページを挿入**：カーソルの位置に改ページを追加します。テキストがいくつか選択されていた場合、それらは改ページで置換されます。
- **文書.../ページサイズ**：ページサイズの選択ができます。様々な標準のページサイズが提示されます。
- **文書.../ページの向き**：標準の向きプロパティ（縦方向 / 横方向）

**注:** ドキュメントが Web モードまたは下書きモードのときにページプロパティを変更しても見た目上は変化ありませんが、それでも設定は可能です。以下の段落プロパティは、下書きモードでも見た目上の変化があります:

- ページの高さ制限 (線が引かれます)
- 内側での改ページを避けるプロパティ
- ウィドウ&オーファンコントロール

## 4D Write Proドキュメントの印刷

4D v15 R5以降、4D Write Proに新機能が含まれるようになりました。これらの標準の機能を使用すると独立した4D Write Proドキュメントを印刷できるようになるのに加え、フォーマット、ページの向き、ページ番号と言った標準の印刷オプションを管理できるようになります。

二つの新しい専用の4D Write Proコマンドと複数の既存のコマンドを使用する事によって4D Write Proドキュメントの印刷と印刷オプションの設定が可能になります。

### 必須要件

- **64-bit版が必須:** 4D Write Proの**WP USE PAGE SETUP** と **WP PRINT**コマンドは64-bit版の4Dでのみ利用可能な新しい内部アーキテクチャーに依存しています。どちらのコマンドも、32-bit版の4Dから呼び出された場合にはエラーを返します。
- Windows 7とWindows Server 2008マシン上では、印刷機能のサポートのためにPlatform Update for Windows 7がインストールされていることを確認して下さい。

### 新しい4D Write Proコマンド

基本的に、二つの新しいコマンドが4D Write Proの印刷機能を管理します。**WP PRINT** と **WP USE PAGE SETUP**です。

- **WP PRINT**( *wpDoc {; printLayout}* ): *wpDoc*引数で定義された4D Write Proドキュメントの印刷ジョブをローンチします(あるいは**OPEN PRINTING JOB**と**CLOSE PRINTING JOB**の間に呼び出された場合にはドキュメントをカレントの印刷ジョブに追加します)。**WP PRINT**は通常**PRINT SETTINGS** あるいは **SET PRINT OPTION**4Dコマンドで定義された印刷設定を使用しますが、ページの余白に関してのみ常に4D Write Proドキュメントページ設定から取得します。**WP PRINT**は(ページサイズやページの向きといった)カレントのページ設定オプションを使用しますが、**WP USE PAGE SETUP**が呼び出されていた場合にはドキュメントのそれらの設定を使用します。
  - 任意の*printLayout*引数を使用すると印刷出力にHTML wysiwygビューを使用できるようになります。*printLayout*引数に1を渡すとHTML wysiwygを印刷し、0(省略時のデフォルト)を渡すと4D Write Proレイアウトで印刷します。  
**注:** HTML wysiwygビューでは、全てのブラウザと互換性があるわけではない4D Write Pro詳細属性は削除されます。
- **WP USE PAGE SETUP** ( *wpDoc* ): カレントのプリンターページ設定を、4D Write Proドキュメント属性のページサイズとページの向きに変更します。このコマンドは、カレントのプリンターページ設定を4D Write Proドキュメントページ設定と同期させるために、**WP PRINT**の直前に呼び出される必要があります。ページの向きとページサイズは4D Write Proエリアのコンテキストメニューを使用して定義する事ができます(新しいページビューモードの段落も参照して下さい)。  
その他の設定は**PRINT SETTINGS**4Dコマンドによって定義されます。カレントの印刷設定は4Dセッション全体に対して設定されます。

### 4D コマンドのアップデート

以下の4Dコマンドは4D Write Pro印刷機能をサポートするようにアップデートされました:

- **SET PRINT OPTION**( *option ; value1 {; value2}* )  
**GET PRINT OPTION**( *option ; value1 {; value2}* )  
4D Write Proドキュメントに対しては、全てのオプションは**WP PRINT**によってサポートされています。  
Paper option と Orientation optionに関しては、**SET PRINT OPTION**を使用してページサイズと向きを個別に設定するよりは、**WP USE PAGE SETUP**を呼び出してこれらの属性を4D Write Proドキュメントの設定と同期させる方が効率的に思えるかもしれません。


新しい印刷のoption引数が使用可能です:

- Page range option (15): *value1*引数には印刷する最初のページ番号(デフォルト値は1)を、そして任意の引数*value2*には印刷する最後のページ番号(デフォルト値はドキュメントの終わり、-1)を渡します。

以下のコマンドは4D Write Pro印刷コマンドと互換性があります:

- **PRINT SETTINGS:** カレントのプリンターに対して印刷設定を設定します。**WP PRINT**がこの後に呼び出された場合、PRINT SETTINGSダイアログで変更されたプリンター設定を使用します(ただし余白ページ設定だけは常に4D Write Proドキュメントの設定を使用します)
- **OPEN PRINTING JOB** と **CLOSE PRINTING JOB:** **WP PRINT**コマンドを**OPEN PRINTING JOB** と **CLOSE PRINTING JOB** の間に呼び出す事によって、一つあるいは複数の4D Write Proドキュメントを一つの印刷ジョブの中に挿入する事ができます。

# 4D Server

 スリープの処理

 Windowsでのシングルサインオン(SSO)

## スリープの処理

4D v15 R5以降、4D Server は接続しているクライアントマシンがスリープ状態へ切り替わった場合に、スリープの終了とともにセッションを復元します。

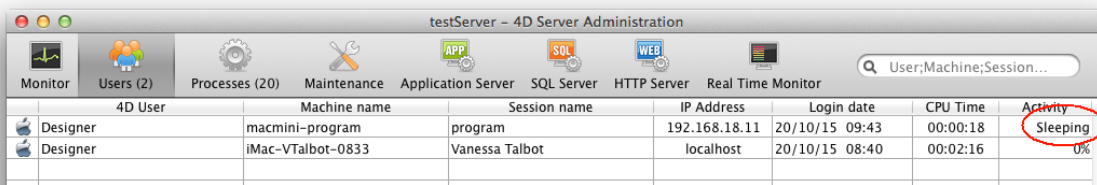
例えば次のような場合が考えられます: 昼休みになり、リモートユーザーが仕事を中断して、サーバーへの接続を開いたまま席を離れます。離席中にマシンはスリープモードに切り替わります。ユーザーが席に戻ってマシンをスリープモードから復帰すると、4D リモートアプリケーションは自動的にサーバーへ再接続し、セッションを復元します。

**注:** 以前のバージョンでは、クライアントマシンがスリープ状態へ切り替わると、サーバーはそのセッションを終了します。スリープからの復帰後、リモートマシンでは接続エラーが発生します。

## スリープモードの自動管理

4D Server はクライアントマシンのスリープ開始の検出と管理を自動で行います:

- 接続中クライアントマシンがスリープ状態に入る際、4D リモートアプリケーションはその接続がまもなく切断されることを 4D Server に通知します。
- サーバー側では、当該ユーザーの Activity が **Sleeping** ステータスに変わります:



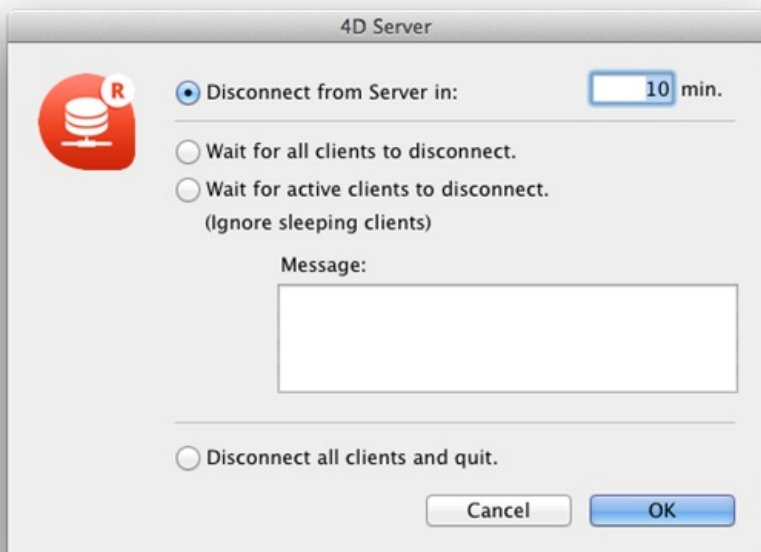
4D User	Machine name	Session name	IP Address	Login date	CPU Time	Activity
Designer	macmini-program	program	192.168.18.11	20/10/15 09:43	00:00:18	Sleeping
Designer	iMac-VTalbot-0833	Vanessa Talbot	localhost	20/10/15 08:40	00:02:16	0%

この新しいステータスはサーバーのリソースを解放します。

- スリープモードから復帰した 4D リモートアプリケーションは自動的に 4D Server に再接続します。

## 新しい 4D Server 終了オプション

スリープ状態のクライアントは接続中と評価されます。4D Server 終了時に表示されるダイアログには新しいオプションが追加されており、“すべてのユーザの接続解除を待つ”場合にスリープ中のクライアントを無視することができます:



現在選択可能なオプションは次の通りです:

- **“サーバは終了します!”**

ダイアログ内で指定した時間が経過すると、スリープ中のユーザーを含むすべてのユーザー接続が切断され、サーバが終了します。

- **“すべてのクライアントが切断するまで待つ”**

スリープ中のユーザーを含むすべてのユーザーが接続を解除次第、サーバは終了します。例えば、昼休み中にメンテナンス処理を行いたい場合には、ユーザーがスリープ中である可能性が高いため、このオプションは適切ではないでしょう。

- (新) **“アクティブなクライアントが切断するまで待つ (スリープのクライアントを無視)”**

スリープ中のユーザーを除くすべてのアクティブユーザーが接続を解除次第、サーバは終了します。このオプションを選択すると、スリープ中のクライアントは接続中とみなされません。例えば、昼休み中にメンテナンス処理を行いたい場合には、このオプションが便利です。

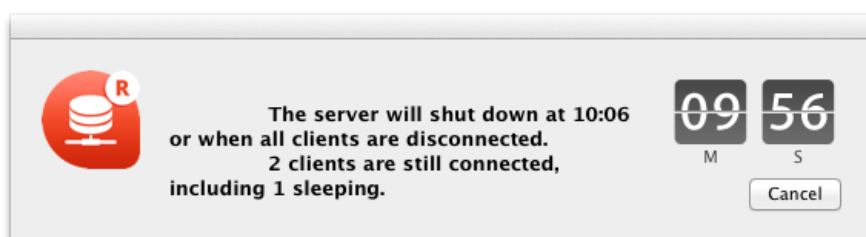
このオプションでサーバを終了したあとにスリープ中のクライアントが復帰すると、接続エラーが発生します。

- **“すべてのクライアントの接続を解除して終了”**

サーバはスリープ中のユーザーを含むすべてのユーザー接続を切断し、終了します。

## スリープ中クライアントの情報

終了待機中にサーバ側で表示されるウィンドウには、スリープ中クライアントの情報も表示されます:



## Windowsでのシングルサインオン(SSO)

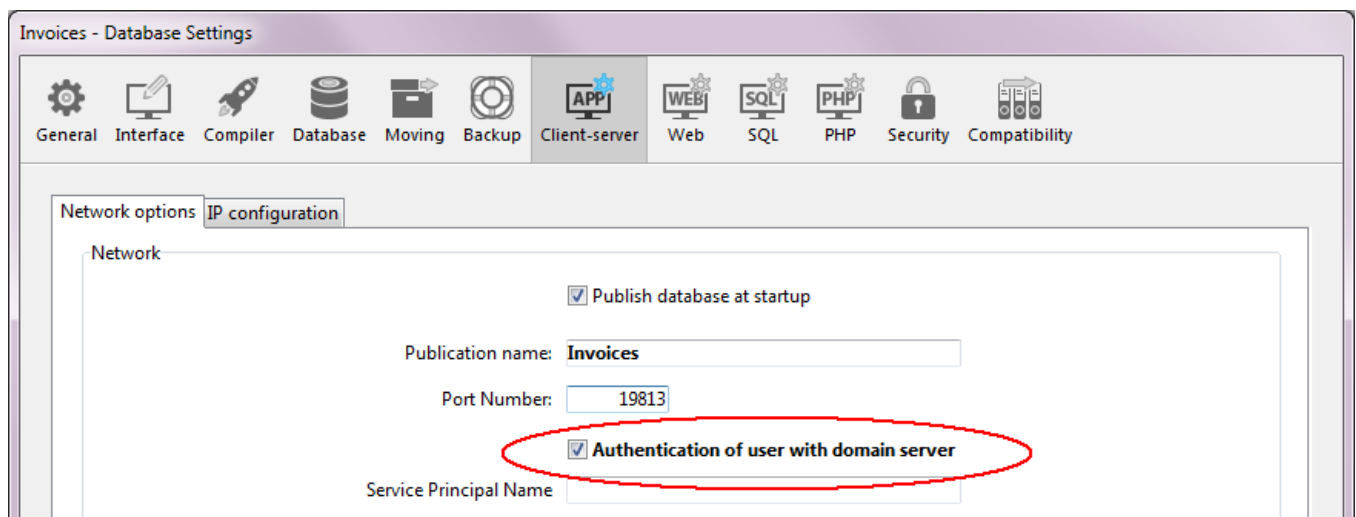
v15 R5以降、4D Serverでは、Windows上のクライアント/サーバーソリューションにSSO(*Single Sign On*)機能を実装することができるようになりました。

SSOを4Dソリューションに実装することにより、ユーザーはカンパニーのWindowsドメインにログインしていた場合に、パスワードを再入力する事なくWindows上の4Dアプリケーションにアクセスできるようになります(Active Directoryを使用)。仕組みとしては、4D ServerアプリケーションはActive Directoryに認証を委任し、標準のメソッドを使用して4Dユーザーをデータベースにログインさせるための、Windowsのセッションログインを取得します。

**注:** SSOは4D Serverでのみ利用可能です(4DのシングルユーザーアプリケーションはSSOをサポートしていません)。

### SSO機能の有効化

デフォルトでは、SSO機能は4D Serverでは有効化されていません。この機能を利用するためには、4D Serverのデータベース設定ダイアログボックスのCSページにある**ドメインサーバーによるユーザー認証**オプションをチェックする必要があります:

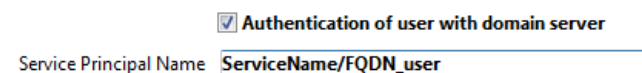


このオプションをチェックした場合、4Dは透過的にWindowsドメインサーバーのActive Directoryに接続し、利用可能な認証トークンを取得します。

このオプションはNTLMプロトコル経由の標準の認証を提供します。4DはNTLMとケルベロスプロトコルをサポートしています。使用されるプロトコルはカレントの設定に応じて4Dによって自動的に選択されます(**SSOのための必須要件**を参照して下さい)。ケルベロスプロトコルを使用したい場合、追加のSPNフィールドに入力する必要があります(後述参照)。

### ケルベロスの有効化

ケルベロス認証プロトコルとして使用したい場合、データベース設定ダイアログボックスのC/Sのネットワークオプションページ内の**サービスプリンシパル名(SPN)**オプションに入力をする必要があります:



このオプションはSPNをActive Directory設定内で設定されているものと同じに宣言します。サービスプリンシパル名とはサービスインスタンスの固有の識別子です。SPNは、ケルベロス認証によってサービスインスタンスとサービスログインアカウントを関連づけるのに使用されます。これによりクライアントがアカウント名を持っていなくても、サービスがアカウントを認証する事をリクエストできるようになります。詳細な情報については、[MSDNウェブサイトのSPNのページ](#)を参照して下さい。

SPN識別子は以下のパターンに従う必要があります:

- SPNがコンピューター属性である場合には、 "ServiceName/FQDN\_user"
- SPNがユーザー属性である場合には "ServiceName/FQDN\_computer"



このとき上記の略称の意味は以下の通りです:

- *ServiceName* はクライアントが認証しようとしているサービスの名前です。
- *Fully Qualified Domain Name (FQDN)* は、コンピューターとユーザーの両者の対し、Active Directoryの3階層のうちどの位置にいるのかを指定するドメイン名です。

4Dデータベースは、SPNは以下のようにして設定することができます:

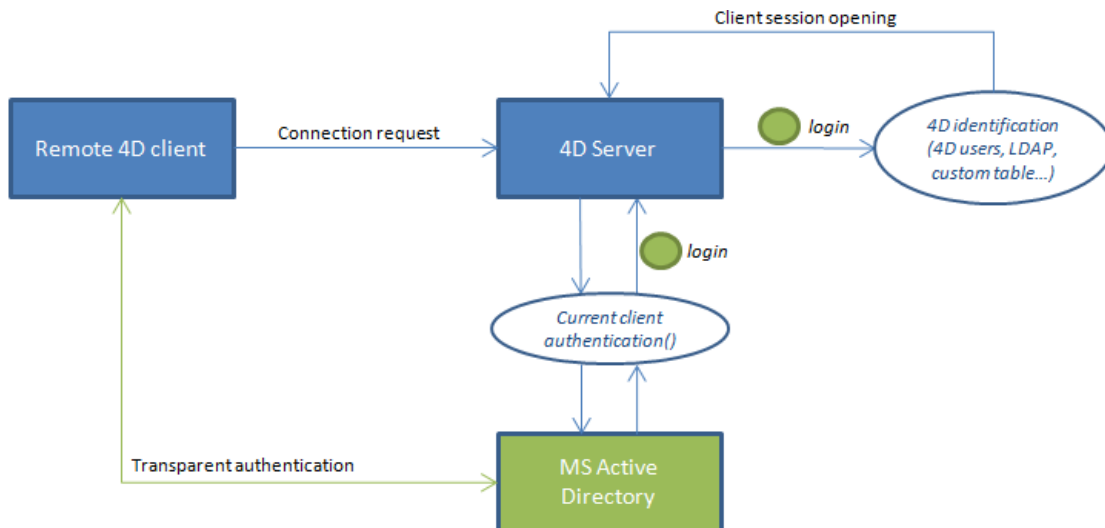
- 4D Serverでの使用に対しては、データベースストラクチャー設定内で設定できます。
- 配布のための使用については、ユーザー設定(データベースのPreferencesフォルダ内に保存されている *settings.4DSettings* ファイル)で設定できます。

## SSOの実装

SSO機能が有効化されていると(上記参照)、4D Serverでのユーザーセッションを開くのにWindowsセッション証明書に基づいたユーザー認証を利用できるようになります。

SSO機能はあくまで認証されたログインのみを提供し、そのログインは自力で4Dの標準ログインメソッドに渡す必要があるという点に注意して下さい。4Dリモートアプリケーションがサーバーに接続しようとするとき、Active Directoryで定義されたユーザーログインを返す **Current client authentication** 4Dコマンドを使用する必要があります。それからこのログインを(ビルトインのユーザーとグループ、LDAPコマンド、あるいは他のカスタムの機構などを使用して)認証システムに渡すことで、お使いの4Dアプリケーションのリモートユーザーへの適切なセッションを開く事ができます。

この原理は以下のような図にまとめる事ができます:



**Current client authentication** コマンドは **On Server Open Connection Database Method** 内で呼び出される必要があります。これはリモートの4Dが4D Serverデータベースへの新しい接続を開くときに毎回呼び出されるものです。認証が失敗した場合、\$0に非ヌル値を渡し接続を拒否する事ができます。

## Current client authenticationコマンドの使用

新しい **Current client authentication** コマンドを呼び出すためには、以下のシンタックスを使用して下さい:

```
login:=Current client authentication (domain;protocol)
```

ここでの略称の意味は以下の通りです:

- *login* はActive Directoryにログインするためにクライアントで使用されるID(テキスト値)です。この値はデータベース内でユーザーを認識するために使用する必要があります。  
ユーザーが正常に認証されていない場合、空の文字列が返され、エラーは返されません。
- *domain* と *protocol* は任意のテキスト引数です。これらはコマンドによって入力され、これらの値によって接続を受け入れまたは拒否することができます:
  - *domain* はActive Directoryのドメイン名です。
  - *protocol* はユーザーを認証するのにWindowsが使用するプロトコル名です。

このコマンドについてのより詳細な情報については、**Current client authentication** コマンドの詳細を参照して下さい

## SSOのための必須要件

4D Serverはカレントのアーキテクチャーや設定によって、様々なSSO設定を管理します。認証に使用するプロトコル(NTLMまたは Kerberos)に加えて**Current client authentication**コマンドによって返される情報は、要件(以下参照)が満たされていた場合には、実際の設定によって変化します。認証に実際に使用されるプロトコルは**Current client authentication**コマンドの*protocol*引数に返されます。

以下のテーブルは**NTLM あるいはケルベロス認証を使用する際の必須要件**をまとめたものです:

	NTLM	ケルベロス
4D Server と 4D リモートが異なるマシン上にあること	yes	yes
4D Server ユーザーがドメイン上にあること	yes	yes
4D リモートが4D Serverユーザーと同じリモート上にあること	yes または no(*)	yes
4D ServerでSPN入力されていること	no	yes(**)
<b>要件が満たされている場合にCurrent client authenticationによって返される情報</b>	<i>user</i> =予想されるログイン、 <i>domain</i> =予想されるドメイン、 <i>protocol</i> ="NTLM"	<i>user</i> =予想されるログイン、 <i>domain</i> =予想されるドメイン、 <i>protocol</i> ="Kerberos"

(\*) 次の特定の設定のみサポートされます: 4Dリモートユーザーが4D Serverと同じADに属するマシン上のローカルアカウントであること。この場合、*domain*引数には4D Serverのマシン名が入力されます。サポートの可否は実際のユーザー設定に依存し、サポートされない場合は空文字列が返されます。

(\*\*) ケルベロスの必須要件が全て満たされているのに**Current client authentication**コマンドが*protocol*引数に"NTLM"を返す場合、以下の状況のどちらかである事を意味します:

- SPNシンタックスが無効です。つまり、[Microsoftによって提示された制約](#)に従っていない事を意味します。
- または、AD内に複製されたSPNが存在する事を意味します。この問題はAD管理者によって修正される必要があります。

**注:** シンタックスが有効であっても、SPN宣言自身が正しいことを意味する訳ではありません。具体的には、AD内にSPNが存在しない場合、**Current client authentication**コマンドは空の文字列を返します。

## 4D Developer Edition 64-bit版

---










4D v15 R5 リリースで 4Dが提供するののは、**OS X 用の 4D Developer Edition 64-bit および 4D Volume Desktop 64-bit のファイナル版**です。また、Windows 用の 4D Developer Edition 64-bit および 4D Volume Desktop 64-bit のプレビュー版も提供されます。

**注:** 4D はすでに Windows および OS X 用の 4D Server 64-bit版を提供しています。

これらの新しい製品によって 4D スタンドアロンアプリケーションと 4D リモートアプリケーションは 64-bit版 OSの利点を最大限活用することができます。64-bit テクノロジーの主な利点は、より多くの RAM メモリを割り当てられることです。

大幅な改定内容にもかかわらず、4D 64-bit版アプリケーションは今までの 4D データベースと高い互換性を持ちます。しかしながら、最新技術を利用するにあたり、いくつかの機能を更新し、いくつかの機能についてはサポートを終了する必要が生じました。変更点についての詳細は [64-bit版リリースの詳細](#) を参照ください。

その反面で 64-bit アーキテクチャーの実装は、マルチスレッドプロセスの処理、OS をベースにした印刷アーキテクチャー、最新のクイックレポートやラベルエディターなど、新しく強力な機能を導入する機会にもなりました。

-  システム設定
-  64-bit版リリースの詳細
-  プリエンプティブ4Dプロセス
-  OS Xでのアニメーションフォームオブジェクト
-  印刷アーキテクチャー(新デザイン)
-  クイックレポートエディター (新デザイン)
-  ラベルエディター (新デザイン)
-  プロパティリスト (新デザイン)
-  OS X上の4Dアプリケーションを32-bitから64-bitへとアップグレードする

## システム要項

64-bit版4D Developerを動かすには、以下のスペックが必要です:

	Windows	OS X
OS	Windows 7 またはそれ以上(64-bit版)	OS X 10.10 (Yosemite) またはそれ以上
RAM	8 GB	8 GB

お使いのバージョンの4DがどのOSと対応しているのかを探すためには、[4D Web サイトにある対応早見表](#)を参照して下さい。

## アーキテクチャー

64-bit アーキテクチャーを想定した 4Dアプリケーションは、この環境専用のバージョンとなります。言い換えるとそれらは 32-bit版OSでは実行できません。

インタープリタモードでは、64-bit版・32-bit版のどちらのアプリケーションでも同じ4Dデータベースを開くことができます(サーバー・ローカルを問いません)。どちらのアプリケーションを使おうと、開発の段階では一切違いはありません(ただし以下の制限を除く)。

コンパイルモードでは、データベースは適切なプロセッサ向けにコンパイルされている必要があります。つまり、64-bit版アプリケーションで開くためには64-bitの、32-bit版アプリケーションで開くためには32-bitのプロセッサ向けにコンパイルしなければなりません。32-bit用にコンパイルされた、インタープリタコードを含んでいないデータベースを 64-bitの 4Dアプリケーションで開くことはできず、その逆もまたしかりです。データベースはどちらか特定のアーキテクチャーだけにコンパイルすることもできますし、両方にコンパイルすることもできます。コンパイルについての詳細は、次の章を参照してください。

以下の一覧は様々な4D実行環境と、そのデータベースのコードとの互換性をまとめたものです:

	利用可能コード	32-bit 4D	64-bit 4D
<b>32-bit 4D Server</b>	インタープリタ	OK	OK(*)
	32-bit コンパイル済みのみ	OK	-
	32-bit と 64-bit コンパイル済み	OK	OK(*)
<b>64-bit 4D Server</b>	インタープリタ	OK	OK(*)
	64-bit コンパイル済みのみ	-	OK(*)
	64-bit と 32-bit コンパイル済み	OK	OK(*)
<b>Local database</b>	インタープリタ	OK	OK
	32-bit コンパイル済みのみ	OK	-
	64-bit コンパイル済みのみ	-	OK
	32-bit と 64-bit コンパイル済み	OK	OK

(\*) 64-bit版 4D は旧式ネットワークレイヤーを利用できないため、64-bit版 4D から 32-bit版 4D Server (Windows・OS X とともに) および Windows用 64-bit版 4D Serverに接続する場合には、サーバー側において**ServerNet** ネットワークレイヤーが有効化されていることを確認する必要があります。なお、Mac OS用 64-bit版 4D Server はそもそも旧式ネットワークレイヤーをサポートしていないため、この確認は不要です。より詳細な情報については、[新しい ServerNet ネットワークレイヤー\(互換性\)](#)の章を参照してください。

## コンポーネントとプラグイン

以下のプラグインおよびコンポーネントは、32-bit版および 64-bit版の 4D Server、4D Developer Edition、そして 4D Volume Desktop によってロード・実行することが可能です:

- 4D for OCI
- 4D Internet Commands
- 4D ODBC Pro
- 4D Pack

- 4D Progress
- 4D SVG
- 4D Widgets
- 4D Write Pro Interface

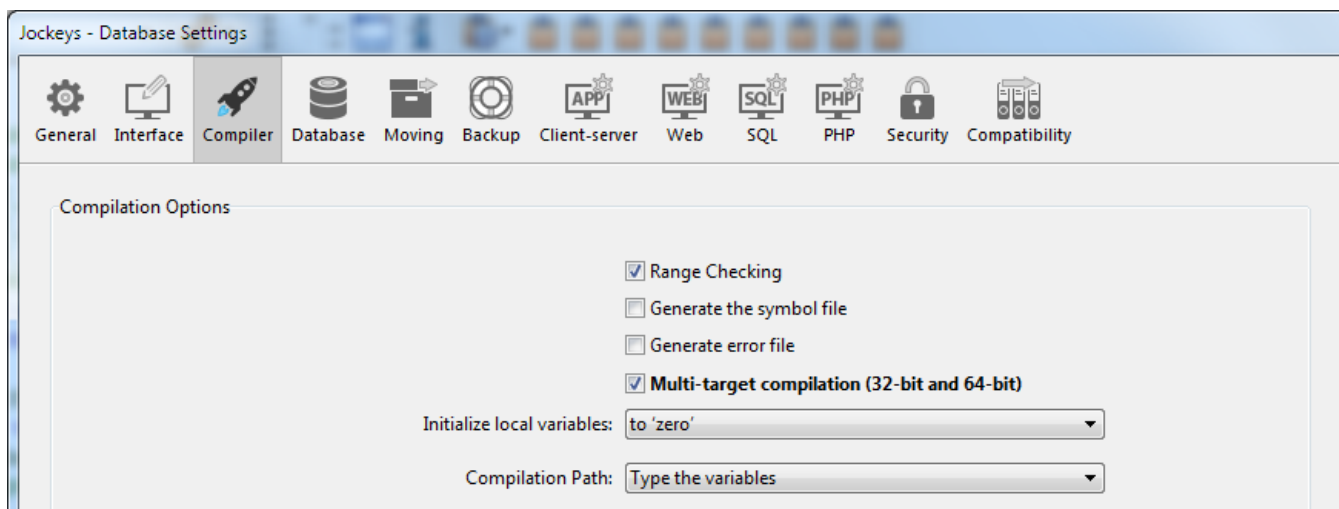
## 4D View と 4D Write

4D View と 4D Write は 32-bit プラグインのため、これらを使用できるのは 32-bit 版の 4D のみとなりますが、64-bit版でも次のことが可能です：

- 64-bit版で開発を行って 32-bit版用にコンパイル / 運用するのを可能にするため、64-bit版 (OS X と Windows) でも プレースホルダーが提供されています。
- 4D Server 64-bit Windows では、これらのプラグインはオフスクリーンエリアに限り実行することができます。

## マルチターゲットコンパイル

4D アプリケーションは 32-bit用と 64-bit用、両方にコンパイルすることもできます。このためには、データベース設定の "コンパイル" ページにある、**マルチターゲットコンパイル(32-bitと64-bit)** オプションを使用する必要があります：



このオプションがチェックされていない場合(デフォルト)、コンパイルすると4D Developerのカレントのアーキテクチャー (32-bit または 64-bit)用のみコードが一つだけコンパイルされます。このオプションがチェックされている場合には、コンパイラーは .4DC および .4DBファイル内に 64-bit版と32-bit版のコードを両方含めます。これにより、ファイルは32-bit版・64-bit版両方の4Dアプリケーションで実行できるようになります。

## 組み込み4Dクライアントの移行

4D v14 R4以前のバージョンで、組み込みクライアントで動くアプリケーションを使用しており、アップデートされたクライアントアプリケーション (32-bit/64-bit) を 4D Serverの自動機構を利用してネットワーク越しに配付したい場合には、移行戦略が必要になります。移行手順は運用中の 4D Server のバージョンと、ServerNet レイヤーの有効化のタイミングを中心に考えます。そのうえで、次の原則を考慮する必要があります：

- v14.x のクライアントは ServerNet が有効化された v15サーバーに接続することはできません。先にアップグレードをする必要があります。
- v15以上の32-bit版クライアントは、ServerNet の設定に関わらず、v15サーバーにアクセスすることができます。
- v15 Rx の64-bit版クライアントは、ServerNet が有効化された v15サーバーにしかアクセスできません。そのため、4D Serverが旧式のネットワークを使用している間は、64-bit版クライアントはアップデートとして提供しないことが重要です。
- OS X用の 64-bit版 4D Server は ServerNet のみサポートしているため、クライアントのアップデートには利用できません。

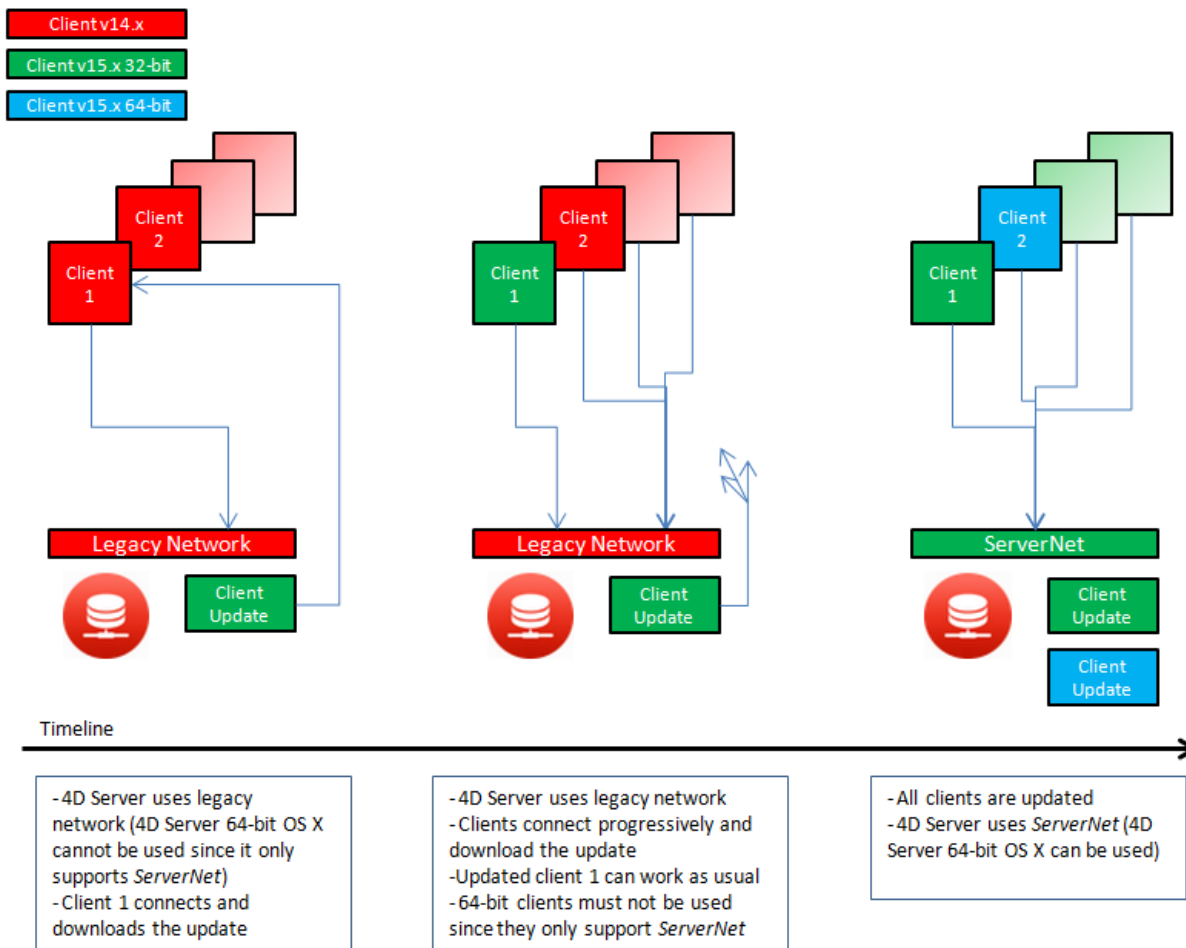
**注:** ServerNet レイヤーについての詳細は、[新しい ServerNet ネットワークレイヤー\(互換性\)](#) を参照ください。

移行にあたっては、以下の手順を踏む必要があります：

1. 4D v15以降のバージョンにアップデートされたクライアントアプリケーションをビルドします。

- "旧式ネットワークレイヤーを使用" ネットワーク設定を有効化して、4D Server v15 を実行します。  
この設定により、すべてのクライアントが接続することができます。  
**注:** OS X用の 64-bit版 4D Server v15 はこのオプションをサポートしていないため、この段階では 32-bit版を使用する必要があります。
- すべてのクライアントが接続して新しいバージョンをダウンロードするまでの期間、この状態で運用を継続します。  
これに要する期間は 1日、1週間、あるいはそれ以上の可能性があります。この移行期間中、以前のバージョンのクライアントも、アップデートされたクライアントも、旧式のネットワークサーバーに接続することができます。
- すべてのクライアントのアップデートが完了したら、旧式のネットワークレイヤーを無効化し、4D Server を ServerNet へと切り替えることができます。
- (任意) 64-bit版にアップデートされたクライアントアプリケーションをビルドします。

この戦略を図に表すと、以下のようになります:



## 64-bit版リリースの詳細

この章では Windows と OS X 用の 64-bit版 4D Developer Edition の実装についての詳細を記載しています。

### 更新された機能

64-bit アーキテクチャーをサポートするために多くの4D機能とダイアログが更新、あるいは書き換えられました。ほとんどの変更は見えないところで行われており、32-bit版と同じように動作します。しかし、一部のエディターについては32-bit版とは異なる形に更新され、印刷などの基本機能についても変更がありました。

機能	対象となる 4D のバージョン	補足
クイックレポートエディター	OS X & Win	完全に書き換えられました。 <a href="#">クイックレポートエディター (新デザイン)</a> 参照。
ラベルエディター	OS X & Win	完全に書き換えられました。 <a href="#">ラベルエディター (新デザイン)</a> 参照。
グラフ	OS X & Win	<b>GRAPH</b> コマンドにグラフ設定を指定する Object 型の引数を渡せます。
フォームエディタープロパティリスト	OS X & Win	新デザイン + 新機能。 <a href="#">プロパティリスト (新デザイン)</a> 参照。
印刷	OS X & Win	更新されました。 <a href="#">印刷アーキテクチャー(新デザイン)</a> 参照。
読み込み / 書き出しダイアログボックス	OS X & Win	32-bit版と同じに動作しますが、XML への書き出しについて XSL をサポートします (XSLT はサポートしません。後述参照) また、ODBC ソースを介することも可能になる予定です (現在は無効、後述参照)

### 無効化された機能

64-bit版 4D Developer Edition では一部の機能が無効化されており、現在検証中です:

機能/テクノロジー	対象となる4Dのバージョン	補足
ODBC ソースを介した読み込み / 書き出し	OS X & Win	無効化
クイックレポートのクロステーブル	OS X & Win	無効化
クイックレポートエディター: 境界線	OS X & Win	無効化
ラベルエディターの標準コード	OS X & Win	無効化
Webエリア内での統合 Web Kit	OS X & Win	無効化 (オプション使用時にはシステム Web エンジンに自動切替; OS X では 4D メソッド (\$4d) へのアクセスを維持)

### サポートされない機能

以下の機能またはテクノロジーは廃止され、64-bit版 4D Developer Editionではサポートされません:



機能/テクノロジー	対象となる4Dのバージョン	コメント
XSLT と Xalan	OS X & Win	<a href="#">_o_XSLT APPLY TRANSFORMATION</a> 、 <a href="#">_o_XSLT SET PARAMETER</a> 、そして <a href="#">_o_XSLT GET ERROR</a> は何もしません。代わりに <a href="#">PROCESS 4D TAGS</a> を使用するか、PHP <i>libxslt</i> モジュールを使用してください。
PICT フォーマット	OS X & Win	'サポートされていない画像フォーマットです' ピクチャ+画像の拡張子が代わりに表示されます。PICT フォーマットは 4D 全体において廃止予定となっています。 <a href="#">AP Is Picture Deprecated</a> も参照してください。
QuickTime	OS X & Win	QuickTime のサポートが廃止されました。 <a href="#">QuickTime support</a> データベースパラメーターは無視されます。
cicn アイコン	OS X & Win	<a href="#">GET ICON RESOURCE</a> はサポートされていません。エラーを返します。
データベース .RSR ファイル	OS X & Win	データベース .RSR ファイルは自動的に開かれませんが、 <a href="#">Open resource file</a> を使用する必要があります。
書き込み可能リソースファイル	OS X & Win	<a href="#">_o_Create resource file</a> はサポートされていません。リソースファイルは読み込みのみモードでしか開けません。
<a href="#">_o_Font number</a>	OS X & Win	このコマンドはサポートされていません。エラーを返します。
旧式ネットワークレイヤー	OS X & Win	<i>ServerNet</i> のみがサポートされています。
ASCII 互換モード	OS X & Win	Unicode モードのみがサポートされています。
4D Write と 4D View プラグイン	Win	旧式プラグインは 64-bit版の 4D とは互換性がありません ([#title id="8695" anchor="2899167"/] 参照)。代わりに 4D Write Pro と 4D View Pro を使用してください。
AP Print settings to BLOB / AP BLOB to print settings (4D Pack)	OS X & Win	内部機能に置き換えられつつあります。 <a href="#">印刷アーキテクチャー(新デザイン)</a> 参照。
OLE ツール	Win	サポートされていません



## 📄 プリエンプティブ4Dプロセス

Windows および OS X用4D Developer Edition 64-bit版では**プリエンパティブ4Dコード**を書き添って使用する、強力な新機能が付属しています。この新機能のおかげで、コンパイルされた4Dアプリケーションはマルチコアコンピューターの利点を全て活かす事ができ、それによって実行速度が向上し、またより多くのユーザーの接続をサポートする事ができます。

### プリエンパティブプロセスとは？

プリエンパティブモードで実行された場合、プロセスはCPUに割り当てられます。プロセス管理はシステムへと委任され、マルチコアのマシン上にてシステムはプロセスをそれぞれのCPUへと個別に割り当てます。

コオペラティブモード (4D v15 R5 までは、こちらのみ使用可能) で実行された場合には、たとえマルチコアのマシン上であっても、すべてのプロセスは親アプリケーションのスレッドにより管理され、同じCPUを共有します。

結果として、プリエンパティブモードでは、アプリケーションの全体的なパフォーマンスは向上します。マルチコアのマシン上では複数のプロセス (スレッド) が真実同時実行可能であるため、パフォーマンスの向上はさらに顕著になります。

その一方で、プリエンパティブモードではそれぞれのスレッドは他から独立しており、アプリケーションによって直接管理されている訳ではないため、プリエンパティブに準拠させたいメソッドには特定の制約が課されます。それに加え、プリエンパティブ実行は特定のコンテキストでのみ使用可能です。

### プリエンパティブモードの利用可能状況

プリエンパティブモードの使用は、**4D 64-bit版**においてのみ可能です。

以下の実行コンテキストが現在サポートされています：

	プリエンパティブ実行
4D Server	○
4D リモート	-
4D シングルユーザー	○
コンパイル済みモード	○
インタープリターモード	-

実行コンテキストがプリエンパティブモードをサポートし、かつメソッドが "スレッドセーフ" である場合、**New process** あるいは **CALL WORKER** コマンドあるいは "メソッドを実行" メニュー項目を使用してローンチされた新しい4Dプロセスは、プリエンパティブスレッド内にて実行されます。

それ以外の場合で、サポートされていない実行コンテキスト (例えばリモート 4Dマシンなど) から **New process** あるいは **CALL WORKER** コマンドを呼び出した場合、プロセスは常にコオペラティブに実行されます。


**注:** 例えば **Execute on server** などのランゲージを使用したストアドプロシージャーをサーバー上で開始する事で、4Dリモートからでもプロセスをプリエンパティブに実行する事が可能です。

### スレッドセーフとスレッドアンセーフ

4Dコードは、いくつかの特定の条件に合致していた場合に限りプリエンパティブスレッド内で実行することができます。実行コードのそれぞれの部分 (コマンド、メソッド、変数など) がプリエンパティブ実行に準拠している必要があります。プリエンパティブスレッドで実行可能な要素は**スレッドセーフ**と呼ばれ、プリエンパティブスレッドで実行できない要素は**スレッドアンセーフ**と呼ばれます。

**注:** スレッドは親プロセスメソッドをスタートとして独自に管理されているので、呼び出しチェーン全体のどこにおいてもスレッドアンセーフなコードが含まれてはいけません。そのようなコードが含まれていた場合、プリエンパティブに実行することはできません。この点については、**プロセスがプリエンパティブに実行される条件とは？** の章で詳細な説明があります。

それぞれの "スレッドセーフティ" プロパティは、要素自身によります：

- 4Dコマンド: スレッドセーフティは内部プロパティです。ランゲージリファレンス内では、スレッドセーフなコマンドは  のアイコンで識別されています。4Dコマンドの大部分はプリエンティブモードで実行可能です。
- プロジェクトメソッド: スレッドセーフであるための条件は [スレッドセーフなメソッドの書き方](#) の段落にまとめられています。

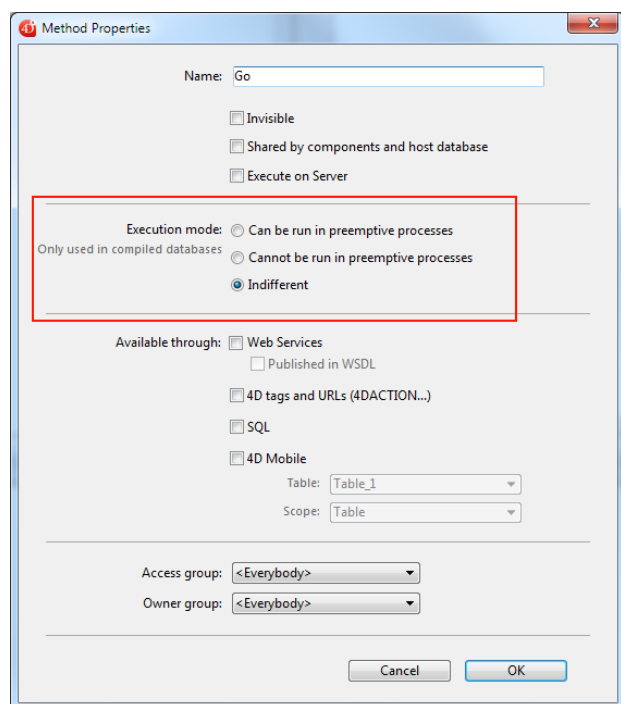
原則として、プリエンティブスレッド内で実行されるコードは外部との相互作用する部分、例えばプラグインコードやインタープロセス変数などを呼び出すことはできません。しかしながら 4Dデータサーバーはプリエンティブ実行をサポートしていることから、データアクセスは可能です。

## プリエンティブ実行宣言

デフォルトでは、4Dはすべてのプロジェクトメソッドをコオペラティブモードで実行します。プリエンティブモードを利用したい場合は、まず最初に、可能な限りプリエンティブモードで開始したいメソッドをすべて明示的に宣言することから始まります。これはつまり、プリエンティブプロセスで実行可能なメソッドであるということです。コンパイラーは、それらのメソッドが実際にスレッドセーフであるかどうかをチェックします (詳細な情報については [スレッドセーフなメソッドの書き方](#) を参照してください)。また、必要であれば、一部のメソッドに対してプリエンティブモードを禁止することもできます。

プリエンティブで使用可能 ("capable") であると宣言することは、当該メソッドにプリエンティブ実行の資格を与えますが、実行時にそのメソッドが実際にプリエンティブモードで実行されることを保障するものではないことに留意が必要です。プロセスをプリエンティブモードで開始することは、プロセス内の呼び出しチェーン内のすべてのメソッドの、関連プロパティを4Dが評価して初めて可能になります (より詳細な情報については、[プロセスがプリエンティブに実行される条件とは?](#) の段落を参照してください)。

メソッドがプリエンティブモードに則していることを宣言するためには、メソッドプロパティダイアログボックスの**実行モード宣言オプション**を使用する必要があります:



以下のオプションが提供されています:

- **プリエンティブプロセスで実行可能:** このオプションをチェックすると、メソッドがプリエンティブプロセスでの実行が可能であると宣言し、可能な場合にはプリエンティブモードで実行するべきと宣言します。メソッドの "プリエンティブ" プロパティは "capable" に設定されます。

このオプションがチェックされていた場合、4Dコンパイラーはメソッドが実際にプリエンティブモードで実行可能かどうかを検証し、そうでない場合 (例えば、プリエンティブモードで実行不可能なコマンドやメソッドを直接的あるいは間接的に呼び出している場合など) にはエラーを返します。その場合にはメソッドを編集してスレッドセーフにするか、あるいは別のオプションを選択します。

メソッドのプリエンティブ性が証明されると、内部で "thread safe" というタグ付けがされ、すべての要件が満たされればプリエンティブモードで実行されます。このプロパティはプリエンティブモードの資格を定義しますが、メソッドが実際にプリエンティブモードで実行されることを保証するものではありません。実行モードは特定のコンテ

キストを必要とするからです (**プロセスがプリエンティブに実行される条件とは?** を参照して下さい)。

- **プリエンティブプロセスでは実行不可:** このオプションをチェックすると、メソッドがプリエンティブモードで実行されてはならないと宣言し、以前の 4D のバージョンと同じように常にコオペラティブモードで実行されます。"プリエンティブ" プロパティは "incapable" に設定されます。

このオプションがチェックされている場合、4D コンパイラーはメソッドがプリエンティブに実行可能かどうかを検証しません。メソッドは内部で自動的に "thread unsafe" とタグ付けされます (例えば理論的にはスレッドセーフであっても)。ランタイムに呼び出された場合、このメソッドは同じスレッド内の他のメソッドを "汚染" し、例えば他のメソッドがスレッドセーフであったとしても、スレッド実行はコオペラティブモードを強制されます。

- **無関係 (デフォルト):** このオプションをチェックすると、メソッドのプリエンティブプロパティを管理したくないということを宣言します。メソッドの "プリエンティブ" プロパティは "indifferent" に設定されます。このオプションがチェックされているとき、4D コンパイラーはメソッドのプリエンティブな実効性を評価し、内部的に "thread safe" あるいは "thread unsafe" のタグ付けをします。プリエンティブ実行に関するエラーは報告されません。メソッドがスレッドセーフと評価されていれば、実行時にプリエンティブコンテキストから呼び出された場合にはプリエンティブスレッド実行を禁止しません。逆に、メソッドがスレッドアンセーフであると評価された場合には、呼び出されたときにどのようなプリエンティブスレッド実行も許可しません。このオプションを使用した場合、内部でのスレッドセーフの評価に関わらず、メソッドが 4D から最初の親メソッドとして直接呼び出された場合 (例えばコマンドから呼び出された場合など)、メソッドは必ずコオペラティブモードで実行されます。内部で "スレッドセーフ" とタグ付けされていた場合には、そのタグは呼び出しチェーン内の他のメソッドから呼び出された場合に限り考慮されます。

**注:** "コンポーネントとホスト間で共有" と宣言されたコンポーネントメソッドも、ホストデータベースによってプリエンティブプロセスで実行される場合には "capable" と宣言される必要があります。

例えば **METHOD GET CODE** を使用して、メソッドのコードを書き出す場合、"プリエンティブ" プロパティは "capable"、あるいは "incapable" の値で "%attributes" コメント内に書き出されます ("indifferent" オプションを選択している場合には、このプロパティは提供されません)。 **METHOD GET ATTRIBUTES** および **METHOD SET ATTRIBUTES** コマンドも、"プリエンティブ" 属性の値 ("capable", "incapable", "indifferent") を取得、あるいは指定します。

以下の表はプリエンティブモードの宣言オプションをまとめたものです:

オプション	プリエンティブプロパティ値 (インタープリター)	コンパイラーの挙動	内部タグ (コンパイル済み)	呼び出しチェーンがスレッドセーフだった際の実行モード
プリエンティブプロセス内で実行可能	capable	資格をチェックし、不可能だった場合にはエラーを返します	スレッドセーフ	プリエンティブ
プリエンティブプロセス内で実行不可能	incapable	評価しません	スレッドアンセーフ	コオペラティブ
無関係	indifferent	評価しますが、エラーを返しません	スレッドセーフあるいはスレッドアンセーフ	スレッドセーフの場合: プリエンティブ; スレッドアンセーフの場合: コオペラティブ; 直接呼びされた場合: コオペラティブ

## プロセスがプリエンティブに実行される条件とは?

**リマインダー:** プリエンティブ実行はコンパイル済みモードでのみ利用可能です。

コンパイル済みモードでは、 **New process** あるいは **CALL WORKER** メソッドで作成されたプロセスを開始するとき、4D はプロセスメソッド (別名親メソッド) のプリエンティブプロパティを読み、そのプロパティに応じてプロセスをプリエンティブモードあるいはコオペラティブモードで実行します:

- プロセスメソッドが "thread safe" であった場合 (コンパイル時に評価)、プロセスはプリエンティブスレッド内で実行されます。
- プロセスメソッドが "thread unsafe" であった場合、プロセスはコオペラティブスレッド内で実行されます。

- プロセスメソッドのプリエンブティブプロパティが "indifferent" であった場合、(メソッドが実際にはプリエンブティブに実行可能だったとしても) 互換性のためにプロセスはコオペラティブスレッド内で実行されます。この互換性機能はメソッドがプロセスメソッドとして使用された場合のみ適用されるという点に注意してください。また "indifferent" と宣言されたもののコンパイラによって内部で "thread safe" とタグ付けされたメソッドに関しては、他のメソッドからプリエンブティブに呼び出すことが可能です (以下参照)。

実際のスレッドセーフプロパティは呼び出しチェーンによります。"capable" と宣言されたプロパティを持つメソッドが、スレッドアンセーフなメソッドをサブレベル (どちらでも) で呼び出した場合、コンパイルエラーが返されます。呼び出しチェーン全体の中で一つでもメソッドがスレッドアンセーフであれば、それは他のすべてのメソッドをいわば "汚染" し、プリエンブティブ実行はコンパイラによって拒否されます。プリエンブティブスレッドは、呼び出しチェーン全体がスレッドセーフであり、プロセスメソッドが "プリエンブティブプロセスで実行可能" と宣言されていた場合にのみ作成可能です。その一方で、同じスレッドセーフメソッドを、呼び出しチェーン内ではプリエンブティブスレッド内で実行し、他の呼び出しチェーン内ではコオペラティブスレッド内で実行することが可能です。

例えば、次のプロジェクトメソッドの場合:

```
// MyDialog プロジェクトメソッド
// インターフェースコールを含み、内部的にスレッドアンセーフです
$win:=Open window("tools";Palette form window)
DIALOG("tools")
```

```
// MyComp プロジェクトメソッド
// 単純な演算を含み、内部的にスレッドセーフです
C_LONGINT($1)
$0:=$1*2
```

```
// CallDial プロジェクトメソッド
C_TEXT($vName)
MyDialog
```

```
// CallComp プロジェクトメソッド
C_LONGINT($vAge)
MyCom($vAge)
```

プリエンブティブモードでのメソッド実行は、"プリエンブティブ" プロパティや呼び出しチェーンに依存します。

以下の表は、これらの様々な状況をまとめたものです:

<input type="checkbox"/> Can be run in preemptive processes	<input checked="" type="checkbox"/> Thread safe for the compiler
<input checked="" type="checkbox"/> Cannot be run in preemptive processes	<input checked="" type="checkbox"/> Thread unsafe for the compiler
<input type="checkbox"/> Indifferent	

宣言と 呼び出し チェーン	コン パイ ル	ス レ ッ ド セ ーフ の 結 果	実行 モー ド	補足
	OK		プリ エン プ ティ ブ	CallComp は親メソッドで、プリエンプティブな使用が "capable"(可能) と宣言されています。MyComp は内部的にスレッドセーフなので、CallComp も内部的にスレッドセーフとなり、プロセスはプリエンプティブになります
	エ ラー 発生		実行 不可 能	CallDial は親メソッドでプリエンプティブ "capable" (可能)、MyDialog は "indifferent" と宣言されています。しかし、MyDialog が内部的にはスレッドアンセーフのため、呼び出しチェーンを "汚染" してしまいます。CallDial の宣言と実際の実効性が矛盾するためコンパイルは失敗します。解決方法は、MyDialog を変更してスレッドセーフにして実行をプリエンプティブにするか、CallDial のプロパティを変更してコオペラティブに実行するようにします。
	OK		コオ ペラ ティ ブ	CallDial はプリエンプティブな使用が "incapable"(不可) と宣言されているのでコンパイル時には内部的にスレッドアンセーフとなり、MyDialog の状況に関わらず実行はかならずコオペラティブになります。
	OK		コオ ペラ ティ ブ	CallComp が親メソッドでプロパティが "indifferent" のため、呼び出しチェーンがすべてスレッドセーフでも、プロセスはコオペラティブになります。
	OK		コオ ペラ ティ ブ	CallDial が親メソッドでプロパティが "indifferent" のため、プロセスはコオペラティブになり、コンパイルは成功します。

## 実際の実行モードを調べるには？

4Dではプロセスに対してコオペラティブ実行かプリエンプティブ実行かを識別する新機能を提供しています：

- **PROCESS PROPERTIES** コマンドを使用するとプロセスがプリエンプティブモードあるいはコオペラティブモードで実行されているかを調べる事ができます。
- ランタイムエクスプローラーと4D Server管理ウィンドウは、どちらもプリエンプティブプロセス(と新しいワーカープロセス)に対して新しい特定のアイコンを表示するようになりました：

プロセス型	アイコン
プリエンプティブストアドプロシージャ	
プリエンプティブワーカープロセス	
コオペラティブワーカープロセス	

**注:** ランタイムエクスプローラーインターフェースに加え、他の既存のプロセスアイコンは4Dおよび4D Server v15 R5 においてアップデートされました。

## スレッドセーフなメソッドの書き方

メソッドがスレッドセーフであるためには、いくつかの条件があります。現在のリリースでは、以下のルールに従う必要があります：

- "プリエンプティブプロセスで実行可能"もしくは"無関係"プロパティを持っている
- スレッドセーフでない4Dコマンドを呼び出していない
- スレッドセーフでない他のプロジェクトメソッドを呼び出していない
- プラグインを呼び出していない



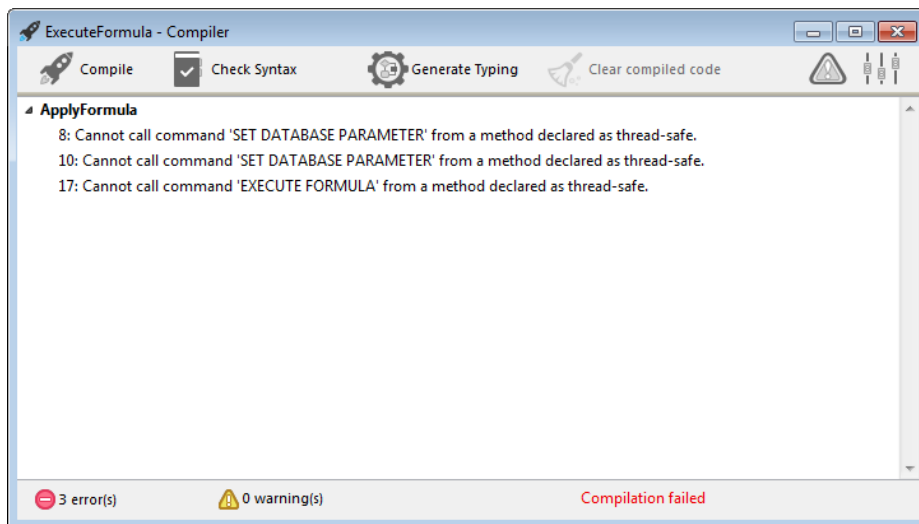
- **Begin SQL/End SQL** コードブロックを使用していない
- インタープロセス変数を使用していない(\*)
- インターフェースオブジェクトを呼び出していない(\*\*) (例外あり、以下参照)

**注:** "コンポーネントとホストデータベース間で共有"メソッドの場合、"プリエンティブプロセスで実行可能"プロパティが選択されている必要があります。

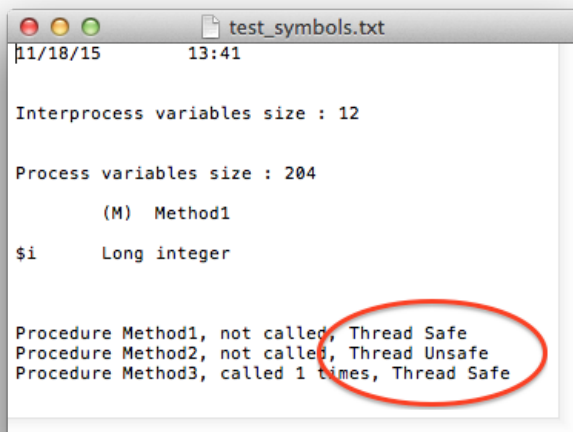
(\*) ワーカープロセスという新種のプロセスによって、プリエンティブプロセスを含むあらゆるプロセス間でデータの交換ができるようになります。より詳細な情報に着いては[プロセス間のメッセージ通信](#)を参照して下さい。

(\*\*) 新しい**CALL FORM** コマンドは、プリエンティブプロセスからインターフェースオブジェクトを呼び出すためのエレガントな(???)ソリューションを提供します。

"プリエンティブプロセスで実行可能"プロパティを持つメソッドは、コンパイル時に4Dによってチェックされます。メソッドがスレッドセーフになるのを妨げる要因をコンパイラーが見つけた場合にはコンパイルエラーが生成されます。:



シンボルファイル(有効化されていた場合には、それぞれのメソッドについてのスレッドセーフの状態が含まれます:



## ユーザーインターフェース


厳密に言えば"外部"アクセスにあたるため、フォームやデバッガなどのユーザーインターフェースオブジェクトへの呼び出しは、プリエンティブスレッドでは許可されません。

プリエンティブスレッドからアクセス可能なユーザーインターフェースは以下のものに限られます:

- 標準のエラーダイアログ。ダイアログはユーザーモードプロセス(4Dシングルユーザー)、あるいはサーバーユーザーインターフェースプロセス(4D Server)内で表示されます。ただし**トレース**ボタンは無効化されます。
- 標準の進捗バー。
- ALERT、REQUESTそしてCONFIRMダイアログ。ダイアログはユーザーモードプロセス(4Dシングルユーザー)、あるいはサーバーユーザーインターフェースプロセス(4D Server)内で表示されます。ただし4D ServerをWindows上でユーザー操作を許可しないサービスとしてローンチした場合には、ダイアログは表示

されないという点に注意して下さい。

## スレッドセーフな4Dコマンド

4Dコマンドのうち、大多数のものがスレッドセーフです。ドキュメントの中では、コマンドプロパティエリア内の  のアイコンがコマンドがスレッドセーフであることを表しています。ランゲージリファレンスマニュアル内にてスレッドセーフであるコマンドの一覧を取得することができます。

また**Command name**を使用するとそれぞれのコマンドについてスレッドセーフかどうかのプロパティを取得することができます(以下参照)。

## トリガー

メソッドがトリガーを呼び出す事のできるコマンドを使用している場合、4Dコンパイラはメソッドがスレッドセーフであるかどうかをチェックするために、トリガーがスレッドセーフかどうかを評価します:

```
SAVE RECORD ([Table_1]) //Table_1をトリガーし、存在すれば、スレッドセーフでなければならない
```

以下は、コンパイル時にトリガーがスレッドセーフであるかどうかをチェックされるコマンドの一覧です:

- SAVE RECORD
- SAVE RELATED ONE
- DELETE RECORD
- DELETE SELECTION
- ARRAY TO SELECTION
- JSON TO SELECTION
- APPLY TO SELECTION
- IMPORT DATA
- IMPORT DIF
- IMPORT ODBC
- IMPORT SYLK
- IMPORT TEXT

テーブルが動的に渡された場合、コンパイラはどのトリガーを評価すべきなのかが分からない場合があります。以下はそのような状況の一例です:

```
DEFAULT TABLE ([Table_1])
SAVE RECORD
SAVE RECORD ($ptrOnTable->)
SAVE RECORD (Table (myMethodThatReturnsATableNumber ()) ->)
```

この場合、すべてのトリガーが評価されます。

少なくとも一つのトリガー内でスレッドセーフでないコマンドが検出された場合、グループ全体がチェックに失敗し、メソッドはスレッドアンセーフと宣言されます。

## エラーハンドルメソッド

**ON ERR CALL**コマンドによって実装されたエラーキャッチメソッドは、プリエンティブプロセスから呼び出される可能性が高いのであれば、スレッドセーフでなければなりません。このような状況を管理するため、コンパイラはコンパイル時に**ON ERR CALL**コマンドに渡されたエラーキャッチプロジェクトメソッドのスレッドセーフプロパティをチェックするようになり、メソッドがプリエンティブ実行に適応していない場合には適切なエラーを返します。

このチェックはメソッド名が定数として渡された場合にのみ可能であり、以下に示す様な、計算された値の場合にはチェックされないという点に注意して下さい:

```
ON ERR CALL ("myErrMethod1") //コンパイラによってチェックされる
ON ERR CALL ("myErrMethod"+String ($vNum)) //コンパイラによるチェックはされない
```

これに加え、4D v15 R5以降、エラーキャッチもロジックとメソッドがランタイムで呼び出す事ができない場合(スレッドセーフに関する問題がある、あるいは"メソッドが見つかりません"などの理由の場合)、新しいエラー-10532 "'methodName'というエラーハンドルメソッドを呼び出す事ができません"エラーが生成されます。

## 4Dランゲージ内での変更点

以下の4Dコマンドはプリエンティブモードを管理するために変更がなされました:

- **New process**

**New process** コマンドは、適切な条件に合致していればプリエンティブスレッドをローンチします。

- **METHOD GET CODE / METHOD GET ATTRIBUTES / METHOD SET ATTRIBUTES**

これらのコマンドは"preemptive"属性を取得あるいは設定します。

- **PROCESS PROPERTIES**

**PROCESS PROPERTIES** コマンドは以下のように更新されました:

- プリエンティブスレッドで経過した累計時間を計算し、その状態を性格に検知
- プロセスの実行モード(プリエンティブあるいはコオペラティブ)を返す

- **Command name**

このコマンドは二つの新しい任意の引数を受け取るようになりました:

**Command name**(command ; info; theme) -> name

- *info* 引数は倍長整数変数でコマンド属性をビットフィールドとして受け取ります。現在はbit 0のみが有効であり、コマンドがスレッドセーフであった場合には1に設定されます。
- *theme* 引数はコマンドの4Dランゲージ内でのテーマ名を返すテキスト変数です。

**Command name**も参照して下さい。

- **DocRef 参照番号** (開かれたドキュメントの参照番号で次のコマンドに使用、または戻り値として返されます: **Open document**、**Create document**、**Append document**、**CLOSE DOCUMENT**、**RECEIVE PACKET**、**SEND PACKET**)。

- プリエンティブプロセスからコールされた場合に生成される DocRef 参照は同プリエンティブプロセスでのみ使用可能です。
- コオペラティブプロセスからコールされた場合に生成される DocRef 参照は別のコオペラティブプロセスでも使用可能です (これまでの動作と変わりません)。

### ポインターの互換性

プロセスは、両プロセスがともにコオペラティブであった場合に限り、ポインターを参照して他のプロセス変数の値へアクセスすることができます。それ以外の場合、4Dはエラーを生成します。プリエンティブプロセスにおいては、4Dコードがインタープロセス変数の値をポインター経由で参照しようとした場合、エラーが生成されます。

以下のメソッドでそのような例を考えます:

Method1:

```
myVar:=42
$pid:=New process ("Method2";0;"process name";->myVar)
```

Method2:

```
$value:=$1->
```

Method1、あるいはMethod2を実行するプロセスのどちらか一つがプリエンティブであった場合、"\$value:=\$1->"という式は実行エラーを生成します。

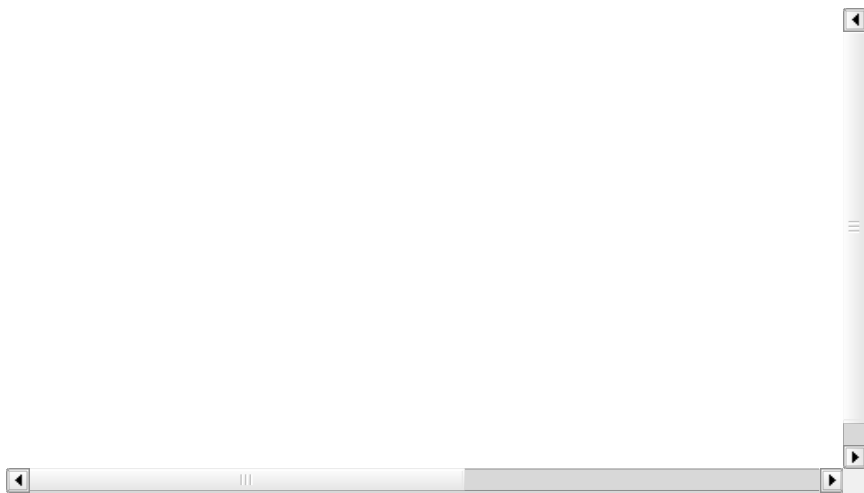


## OS Xでのアニメーションフォームオブジェクト

OS X で実行される 64-bit版の 4Dアプリケーションは OSネイティブのアニメーションの恩恵により、4Dフォームのユーザーエクスペリエンスを向上させることができます。具体的には以下のような点で向上しています：

- フィールド、入力可能変数、ラジオボタン、チェックボックスはフォーカスを受けるとアニメーションを表示します。
- ラジオボタンとチェックボックスはマウストラッキング時にアニメーションをトリガーします。
- スクロールバーに特殊なアニメーション効果があります (ただし Yosemite版のみ)
- リストフォームで TABを押すと、カラムがスムーズに移動します。

以下の動画はこれらのアニメーションの概要をまとめたものです：



**注:** Apple はアプリケーション内でどのようにアニメーションを使用したらいいかという[ガイドライン](#)を発行しています。

## 印刷アーキテクチャー(新デザイン)

4D v15 R5 64-bit版 (4D Developer Editionおよび4D Volume Desktop) において、印刷アーキテクチャーが書き換えられ、これにより最新のOSに基づいた印刷ライブラリとダイアログボックスの恩恵を受けられるようになりました。

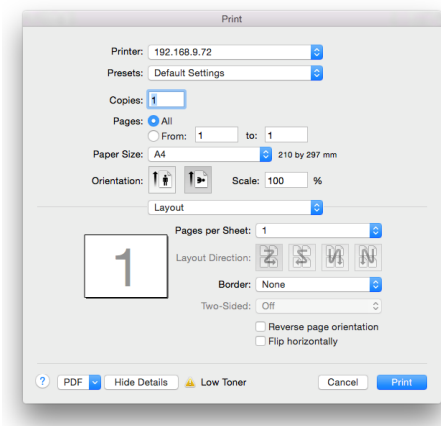
この内部的なアップデートは大部分において4Dユーザーからは透過的ですが、以下の変更については注意する必要があります：

- "印刷ジョブ" ダイアログボックス (WindowsとOS X) はアップグレードされ、両プラットフォームにおいて標準のシステムダイアログボックスとなりました。
- "ページ設定" ダイアログボックスは今後自動的に表示されなくなりました。それに伴い印刷ダイアログボックスを管理するコマンドは更新されています。
- **SET PRINT OPTION / GET PRINT OPTION** での既存のオプションは変更されています。
- **SET CURRENT PRINTER** では PDF印刷が可能になる新しいオプションが利用可能になりました。

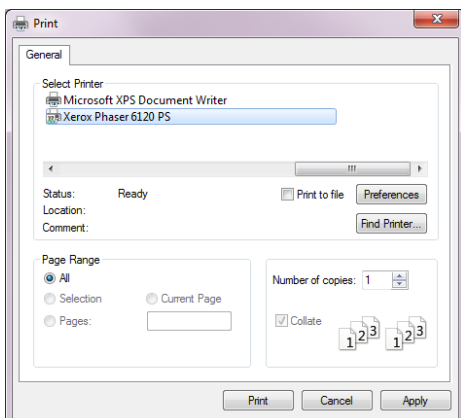
### 印刷ジョブダイアログボックスのアップデート

4Dの64-bit版では、印刷ジョブダイアログボックスはOS XとWindows両方で新設計され、標準の印刷オプションを提供するようになりました。

- **OS X:** ダイアログボックスは用紙サイズ、印刷の向き、拡大/縮小オプションを含めるようになりました。



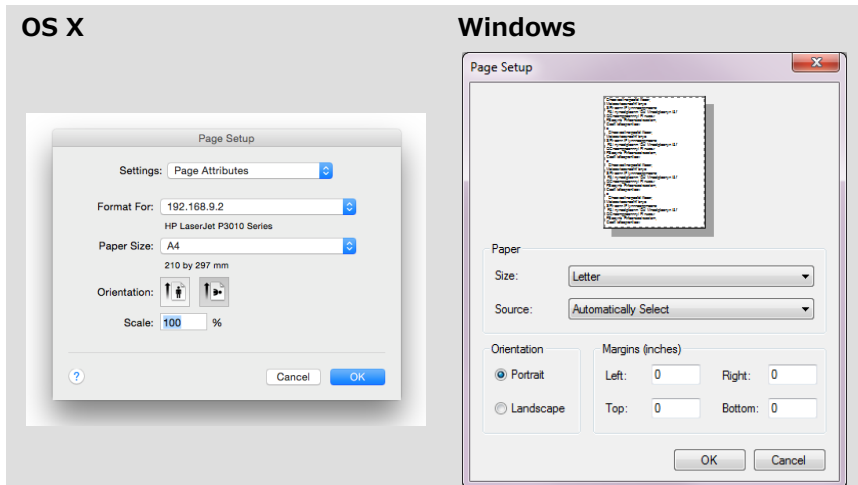
- **Windows:** ダイアログボックスはOSによって提供され、**設定**ボタンを押す事によってページ設定オプションへとアクセスすることができます。**プリンターを検索**ボタンを押す事によってプリンター検索機能へとアクセスする事もできます。



これらの新しいダイアログボックスは、デザインモードのメニュー項目からローンチしたものやプログラミングによるものと、どのような印刷ジョブにおいても使用可能です。

### ページ設定ダイアログボックスはデフォルトでは表示されない

印刷オプションの大部分は"印刷ジョブ"ダイアログボックスから選択可能になったことから、"印刷設定"ダイアログボックスは印刷コマンドが呼ばれた際にはデフォルトでは表示されなくなりました。この変更によって影響を受けるコマンドはこの章の中に説明があります。



**注:** Windows上では、ページ設定ダイアログボックスはアップデートされ、OSによって提供されるようになりました。

## PRINT SETTINGS

*PRINT SETTINGS* { ( dialType ) }

更新されたアーキテクチャーに合わせて、**PRINT SETTINGS**コマンドはデフォルトでは"印刷ジョブ"のみを表示するようになりました。"ページ設定"ダイアログボックスは *dialType* 引数に *Page\_setup\_dialog* を渡した場合にのみ表示されるようになりました。以下の表は4D v15 R5 64-bit版で表示されるダイアログと他(以前)のリリースでのダイアログの比較をまとめたものです:

<i>dialType</i> 値(定数)	4D v15 R5 64bit版	以前のバージョン
0 または省略時	印刷ジョブ	ページ設定+印刷ジョブ
1 (Page setup dialog)	ページ設定	ページ設定
2 (Print dialog)	印刷ジョブ(= 0 または省略時)	印刷ジョブ

**注:** "印刷オプション"テーマ内に、*dialType*の値を管理するための定数が追加されました: *Page setup dialog* (値 1) と *Print dialog* (値 2)です。

## PRINT SELECTION, PRINT RECORD, PRINT LABEL

*PRINT SELECTION* ( { aTable } ; ; { \* | > } )

*PRINT RECORD* ( { aTable } ; ; { \* | > } )

*PRINT LABEL* ( { aTable } ; ; { document ; ; \* | > } )

これらのコマンドは以前は\*または>引数省略時には、印刷ダイアログボックスを両方(ページ設定+印刷ジョブ)表示していました。

4D v15 R5 64bit版では、これらのコマンドは"印刷ジョブ"ダイアログボックスのみを表示するようになります。

## 変更された印刷オプション

以下の印刷オプション (**GET PRINT OPTION** あるいは **SET PRINT OPTION** コマンドで使用) は4D v15 R5 64bit版において変更されました:

オプション(定数)	OS	4D v15 R5 64bit版での状況
2 (Orientation option)	Windows と OS X	アップデート済み:印刷ジョブ内から呼び出し可能。これは同じドキュメント内においてページごと縦向きと横向きを切り替える事ができるということです。
8 (Color option)	Windows のみ	削除済み(非推奨): 使用はできません
13 (Mac spool file format option)	OS X のみ	削除済み: <b>SET CURRENT PRINTER</b> オプションで代用(以下参照のこと)

## SET CURRENT PRINTER

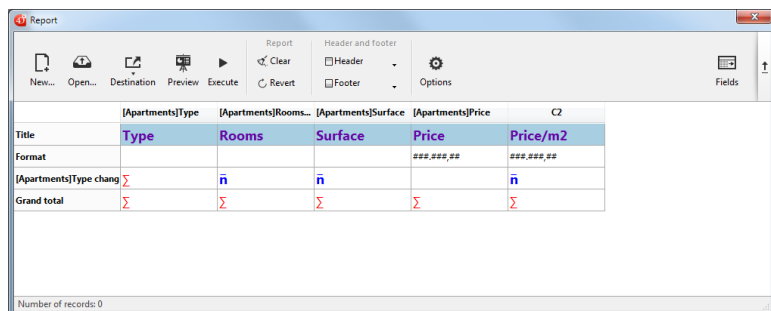
### SET CURRENT PRINTER ( printerName )

4D v15 R5 64bit版では、このコマンドはOS XとWindows 10の両プラットフォームにおいて引数に対して新しい定数を受け取り、それによって一般的なPDFプリンターの使用を指定します。この定数は**Print Options**"印刷オプション"テーマ内にあります:

定数	型	値	詳細
一般的なPDFプリンター	テキスト	_4d_pdf_printer	<ul style="list-style-type: none"><li>- OS X上では、カレントプリンターをデフォルトのドライバーに設定します。このドライバーは非表示となっており、<b>PRINTERS LIST</b>コマンドを使用しても一覧の中に返されません。PDFドキュメントのパスは<b>SET PRINT OPTION</b>を使用して設定されている必要があるという点に注意して下さい。そうでない場合にはエラー3107が返されます。</li><li>- Windows上では、カレントプリンターをWindows PDFドライバー("Microsoft Print to PDF" という名前)に設定します。このオプションはPDFオプションがインストールされているWindows 10でのみ使用可能です。他のWindowsやバージョン、あるいはPDFドライバーが使用不可の場合には、コマンドは何もせず、OK変数は0に設定されます。</li></ul>

## クイックレポートエディター (新デザイン)

4Dの64-bit版には新しく作成された4D クイックレポートエディターが含まれます。インターフェースを見て、少し時間をかけてこの新しいエディターの現在の機能を体験してみましょう。現代的なアーキテクチャーに基づいた新エディターは、4Dのすべての後続バージョンにおいて旧エディターを置換します。直感的なリボンインターフェースを使って、洗練された新しいレポートをデザイン・実行することができる一方で、4Dの旧バージョンにおいて作成されたものを扱うこともできます：



新しいクイックレポートエディターは単純化され整頓されたインターフェースを特徴としています。つまりメニューバーがないということです。全ての昨日は新しいリボンベースのインターフェース、あるいはコンテキストメニューを通して使用可能です。ここには"ウィザードモード"はありません。標準のインターフェースを使用し、自然なステップでレポートをデザインし編集します。4Dコード生成ページは提供されていません。

これらに加え、クイックレポートのデザインを容易にするための新しい人間工学的な機能が提供されています。

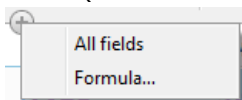
## 互換性

- **既存のクイックレポートファイルのサポート:** 新しいクイックレポートエディターは以前のバージョンで作成されたクイックレポートファイルとも互換性があります。ファイルの内部的なフォーマットを変更することなく、ファイルを開いて、編集して、保存する事ができます。
- **以前のエディターの機能のサポート:** 以前のエディターの機能はフルにサポートされます。ランゲージコマンドも含まれますが、以下の例外は除きます。
- **一時的に無効化された機能:** 以下の機能はカレントのバージョンのクイックレポートエディターでは無効化されています:
  - クロステーブルレポート
  - セル境界線

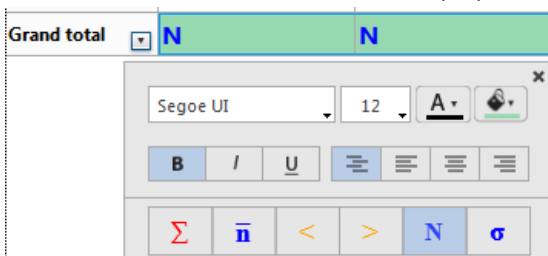
64-bitアプリケーションの互換性の詳細については、[64-bit版リリースの詳細](#)の章を参照して下さい。

## 主な機能

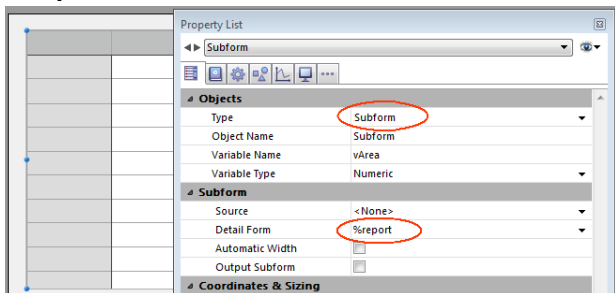
- カラム (あるいはフォーミュラカラム)、またはすべてのフィールドの追加をコンテキストメニューから行えます:



- 総計行を定義することができます。
- コンテキストウィンドウを通して、セル/行/カラムのグラフィック属性を設定することができます:



- **%report** サブフォームを使って、フォームに埋め込みエリアを追加することができます：



注：旧クイックレポートエディターのプラグインエリアもサポートしており、自動でサブフォームに変換されます。

注：

- **クリア** ボタン誤操作による削除を防ぐために確認ダイアログボックスをトリガーします。
- レポートを保存せずにエディターを閉じた場合でも、データベースが閉じられるまではレポートはカレントメモリーに保存されています。つまりエディターを再度開いた場合には表示されるという事です。

## 新しい機能

ツールバーには印刷プレビューおよび、フィールドという二つの新しいツールが含まれています。

### 印刷プレビュー



OS のデフォルトシステムビューワーを使い、レポートのプリントプレビューを表示します。

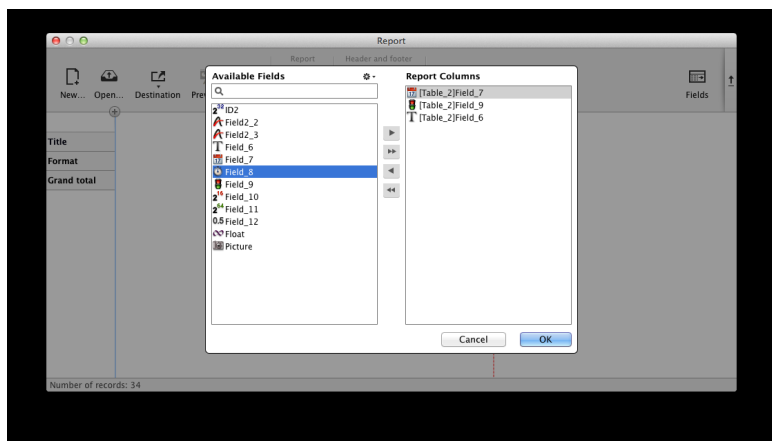
### フィールド



次の二つのリストを左右に表示するウィンドウを開きます：


- **使用可能フィールド**：テーブルのすべてのフィールドを表示します。
- **出力カラム**：レポートに挿入されているフィールドを表示します。

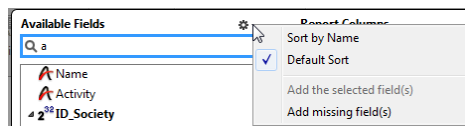
**出力カラム**へのフィールドの追加・削除は、二つのリストに挟まれている矢印ボタンを使用します。フィールドの順番を変更するには、リスト内でフィールドをドラッグ移動します。また、ダブルクリックでフィールドを追加することも可能です。



**OK** をクリックすると、すべての変更が自動的にレポートに反映されます。例えば、**出力カラム**へのフィールドの追加・削除は、カラムの追加・削除という形でレポートに反映されます。フィールドの順番を変更すると、同様にカラムの順番に反映されます。

注：検索エリアは**使用可能フィールド**のリストに対して "含む" 検索を行い、入力した文字が名称に含まれるフィールドを検出します。

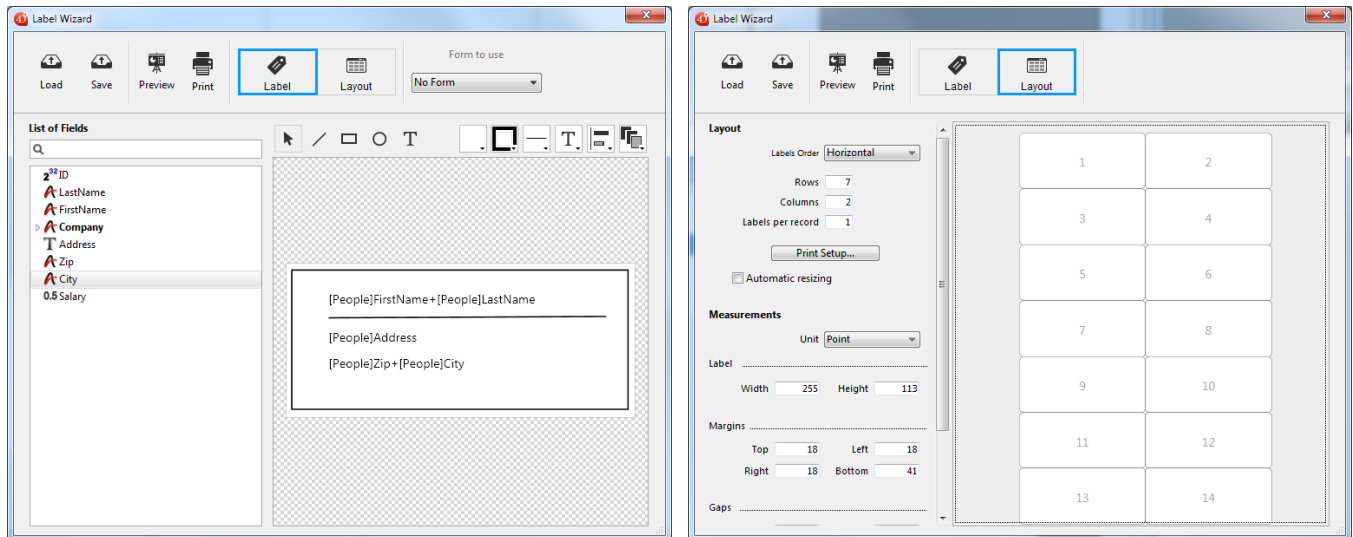
アクションメニュー  を使って、**使用可能フィールド**を並べ替えたり、複数のフィールドを一括で**出力カラム**に追加したりできます。



- **名前で並べ替え:** 名称を基準にフィールドを並べます
- **デフォルトの並べ替え:** 作成順にフィールドを並べます
- **選択したフィールドを追加:** 選択順にフィールドを**出力カラム**に追加します
- **未使用フィールドを追加:** **出力カラム**に追加されていないフィールドをすべて追加します

## ラベルエディター (新デザイン)

64-bit版の4D Developer Editionには、将来的にすべての4Dバージョンで現行のラベルエディターに取って代わる、新しい4Dラベルエディターが含まれています。この新しいエディターでは新しいリボンインターフェースとコンテキストメニューが特徴です:



4Dラベルエディターは旧ウィザードの機能の大部分をサポートする一方、新しい機能も提供します。

## 互換性

### ファイルフォーマット

新しいウィザードによって保存される4Dラベルのファイル拡張子は".4lbp"です。このフォーマットは内部的にはXMLで書かれているので、オープンであるという点に注意してください。

以前のウィザードで作成された古い4Dラベルファイル(".4lb" 拡張子)はサポートされます。新しいウィザードはフォーマットを変えずに、以前のラベルを読み込み、編集、保存することができます。

**注:** 以前のラベルファイルでは、ピクチャーは PICT フォーマットで保存されていました。このフォーマットは4Dではもうサポートされないため(後述参照)、新しいエディターではこれらのピクチャーは赤いバツ印の付いたエリアで置き換えられます。

### サポートされない機能

以下の機能は、新しいラベルウィザードではサポートされません:

機能	補足
フォーマットメニュー	代わりに <b>String</b> コマンドを使用してください。例: <pre>String([Emp]Birth;internal date long)</pre>
アウトラインスタイル	4Dでサポートされなくなりました。
パターン	4Dでサポートされなくなりました。
Quickdraw PICT	4Dで廃止予定の機能です / 4D Developer Edition 64-bitではサポートされていません。
デフォルトロックボタン	今後は使用できません

### 無効化されている機能

以下の機能は新しいラベルウィザードでは現在、無効化されています:

機能	補足
標準のコードポップアップメニュー	



### 使用可能なフォームとメソッドの定義

新しいラベルウィザードでは、ダイアログボックスで選択可能なフォームとメソッドを制限する機能が新しく追加されました。次のメニューが対象となります:

- "ラベル" ページの **使用するフォーム** ポップアップメニュー、または
- "レイアウト" ページの **適用するメソッド** ポップアップメニュー

選択可能なフォームとメソッドを指定するには、プロジェクトフォルダーに JSON ファイルを追加します。

**注:** 4D v15 R2 プレビューでは "使用するフォーム" ポップアップメニューの項目は特定のプロパティに基づいてフィルターされていました。このフィルターは取り除かれ、デフォルトですべてのテーブルフォームおよびプロジェクトメソッドが選択可能です。データベースがテーブルフォームを持たない場合には、このメニューは表示されません。

ラベルデザインに選択可能なフォームまたはメソッドを定義する手順は下のとおりです:

1. **label.json** という名前の JSON ファイルを作成し、データベースフォルダー内の **Resources** フォルダーにおきます。
2. このファイル内に、ラベルウィザードのメニュー内にて選択可能にしたいフォームまたはメソッド名を追加します。

**label.json** ファイルの中身は、以下のようになっています:

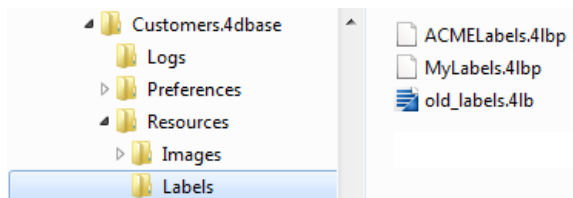
```
[ { "tableId":2, "forms":[], "methods":["myMethod1", "myMethod2"]}, { "tableId":1, "forms":["Sample Label 1", "Sample Label 2"], "methods":[] } ]
```

**labels.json** ファイルを設定しない場合には、メニューの選択項目は制限されません。

### ラベルファイルのプリロード

新しいラベルウィザードではラベルファイルをアプリケーション内に保存することができるため、ユーザーは直接 **読み込み** ボタンを使ってラベルデザインを選択し開くことができます。

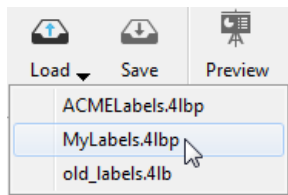
このためには、データベースの **Resources** フォルダー内に **Labels** フォルダーを作成し、そこにラベルファイルをコピーするだけです:



ラベルウィザードが起動するとこのフォルダーが検知され、中に有効なラベルファイルがあった場合には、**読み込み** ボタンにポップアップアイコンが追加されます:

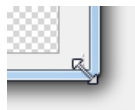


これで、この一つのポップアップメニューからラベルデザインを選択することが可能になります。新しい ".4lbp" ファイルと旧式の ".4lb" ファイルは両方サポートされます:



### ズーム

ウィンドウの右下のリサイズカーソルをドラッグすることでラベルデザイン ("ラベル" ページ) をズームすることが可能です:



### ドラッグ&ドロップ

"ラベル" ページではドラッグ&ドロップがサポートされています:

- デスクトップからピクチャーファイルをラベルデザインエリアにドロップすることができます。

- デスクトップからラベルファイル (".4lbp" ファイルのみ) をラベルデザインエリアにドロップすることができます。

## 改善された操作性

一般的な操作性が改善されました:

- 静的なテキストはラベルデザインエリアに直接入力されます。
- グラフィックオブジェクトはコンテキストメニューを通して管理されます。
- 印刷プレビュー: 最初のページはラベルサイズを区切るすかし入りで印刷されます。

## プロパティリスト (新デザイン)

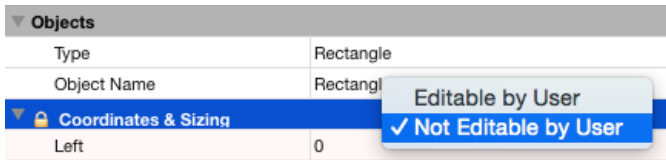
フォームエディターのプロパティリストは4D Developer Editionの64-bit版にて新設計となりました。  
これに加え、いくつかの機能も修正されています:

- ユーザーフォームエディター内にてプロパティのロック/アンロックを設定・閲覧する新機能
- カラーの管理と、カラーパレットが改善されました。

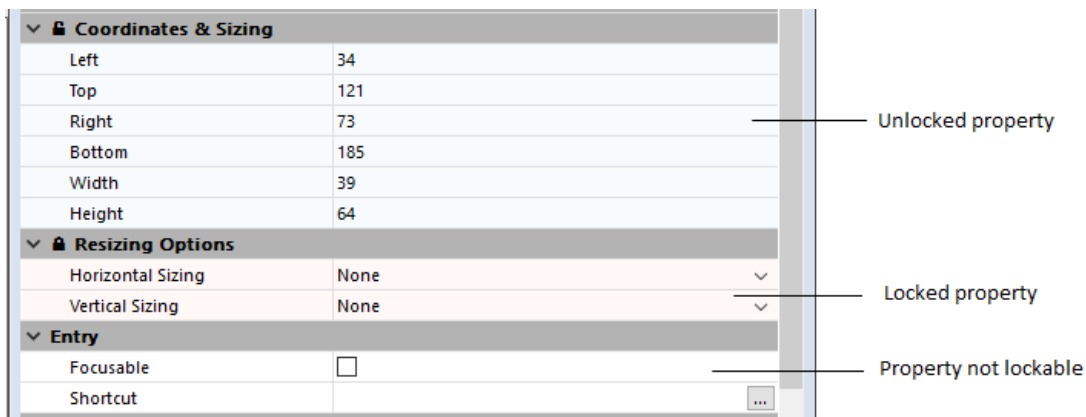
### ユーザーフォームプロパティ

フォームが"ユーザー更新可"に設定されているとき、プロパティリストにおいて二つの新しい機能が使用できるようになりました:

- デベロッパは新しいコンテキストメニューを使用して編集可能なプロパティを選択できるようになりました:



- 南京錠のアイコンに加えて、プロパティの背景色もロック・アンロックの状態を知らせるようになりました:
  - 薄い赤:ロック
  - 薄い青:アンロック



### カラープロパティ

プロパティリストの全てのカラープロパティにおいて、新しいデフォルトのカラーパレットが使用されるようになりました:

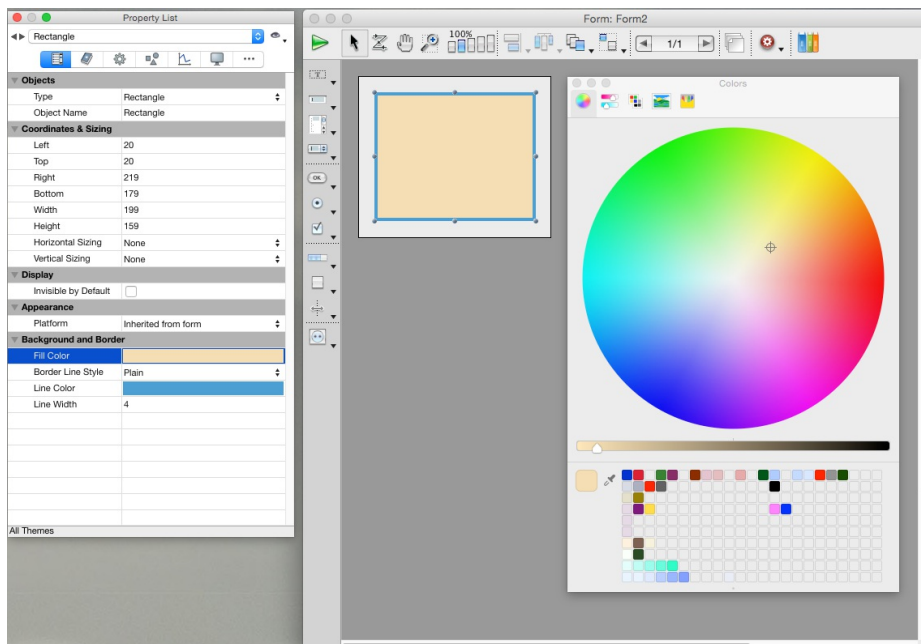
- 背景色
- 線カラー
- フォントカラー
- 交互背景色



## システム準拠のフローティングシステムカラーパレット(OS X)

OS Xにおいて、ユーザーがカラーリストメニューのパーソナライズ...を選択したとき、フローティングのシステムカラーパレットが表示され、これはユーザーが閉じるボックスをクリックするまで開かれたままです。このカラーピッカーウィンドウはOS X-準拠であり、どのようなカラープロパティを選択してもこれを使用してカラーを変える事が可能です。

以下の例では、プロパティリストで"塗りカラー"プロパティが選択されていて、システムパレットで選択されたカラーは"塗りカラー"プロパティに適応されます:



## OS X上の4Dアプリケーションを32-bitから64-bitへとアップグレードする

OS X上で既存の4Dアプリケーションを32-bit版から64-bit版へとアップグレードするには、多少の準備が必要となります。もしお使いのアプリケーションがWindowsあるいはOS X用の4D Server 64-bit版で実行できるなら、大部分の準備は既に済んでいると言ってよいでしょう。64-bit版のデスクトップアプリケーションの場合は、いくつかの追加の段階を踏む必要があるかもしれません。この章ではアップグレードの前と後、両方で必要なポイントを全て検証するのに手助けするステップ・バイ・ステップ形式のチェックリストを用意しています。

64-bit版製品への以降については、いくつかの機能がアップデートされていたり、無効化、あるいは廃止予定と宣言されています。全ての詳細はこちらのページにまとめてあります: [64-bit版リリースの詳細](#)

**注:** 通常のアップグレードプロセスと同様、大規模なステップの前にMSCを使用して検証を行い、データとストラクチャー両方に問題がない事を確認するのが良いでしょう。

### プラグインをチェックする

最初にするのは、プラグインがあれば、それらを64-bit版へとアップグレードすることです:

#### ● 4D プラグイン:

4D Write と4D Viewを除き、全てのプラグインは既に64-bit版で動作しています。

- お使いのアプリケーションが4D Writeを使用している場合、そのコードを4D Write Proへと移行することを検討する必要があります。このとき、既存の32-bit版のコードはそのまま保持しておき、並行して新しい64-bitベースのモジュールの4D Write Proを作り始めるのが良いでしょう。
- お使いのアプリケーションが4D Viewを使用する場合、4D View Pro機能あるいは別の選択肢を使用する必要があります。

#### ● サードパーティのプラグイン:

そのプラグインのプロバイダーに連絡をして、64-bit版を取得してください。

### 32-bit版においてアップグレードに必要な準備

1. お使いのアプリケーションを最新の32-bitリリース(例えば4D v15 R5 32-bitなど)へとアップグレードして下さい。
2. Unicodeモードが有効化されている事を確認して下さい。
3. あらゆるPICT/cicn/QuickTimeピクチャーを変換して下さい。  
データ内にある廃止予定の形式のピクチャーを検出するためには、**AP Is Picture Deprecated**コマンドを使用する事ができます。また、データベースのストラクチャー内にあるサポートされていないピクチャーも全て置き換える必要があります。
4. XSLTベースの機能(**\_o\_XSLT APPLY TRANSFORMATION**、**\_o\_XSLT SET PARAMETER**あるいは**\_o\_XSLT GET ERROR**コマンド)を、例えば**PROCESS 4D TAGS**コマンドで置き換えて下さい。
5. **\_o\_Font number**の呼び出しをフォント名での呼び出しに置き換えて下さい。
6. リソースファイルを作成あるいは変更するコードを全て削除して下さい。

この時点で、お使いのデータベースは64-bit版の4Dで開く準備ができました。

### データベースを64-bit版で開いてチェックする

1. お使いのアプリケーションを4D Developer Edition 64-bit版で開いて下さい。
2. Webエリアにおいて統合されたWebKitを使用している場合、これらは自動的にシステムエンジンに切り替わっているため、チェックしてみてください(4Dメソッド(\$4d)へのアクセスは引き続き有効です)。
3. コード内で**SET PRINT OPTION**コマンドのMac spool file format optionを使用している場合、それらを**SET CURRENT PRINTER**とGeneric PDF driver定数の組み合わせの呼び出しで置き換える必要があります。
4. ラベルエディターの呼び出しと使用をチェックして下さい([ラベルエディター \(新デザイン\)](#)を参照して下さい)。
5. クイックレポートの呼び出しと使用をチェックして下さい([クイックレポートエディター \(新デザイン\)](#)を参照して下さい)。

い)。


これでお使いのアプリケーションは完全に64-bit版で使用できるようになり、4D 64-bit版のみの新機能も使用できるようになりました。

## 64-bit版の機能の利点

---

1. データベースキャッシュが増加します！64-bitアーキテクチャーはデータベースキャッシュの限度を引き上げることができます。より大きいキャッシュを使用するだけでデータベースのパフォーマンスを向上させましょう。
2. プリエンプティブプロセス、アニメーション付きフォームオブジェクト、新しい印刷機能など、強力で新しい64-bit機能を使用してみましょう。
3. アプリケーションを4D Runtime Volumeライセンス64-bit版でビルドしましょう。

## 最適化

 SQL OUTER結合の実行時間を短縮

## SQL OUTER結合の実行時間を短縮

---

4D v15 R5 より、ビルトインの SQL OUTER結合が最適化されました。同一の *SELECT* ステートメント内で結合とテーブルを複数組み合わせるクエリが対象です。

特に、WHERE句で "field=constant" 型のフィルターを指定している場合は、今までの実装と比較して最大で10倍速く実行されます：

```
SELECT * FROM T1
LEFT JOIN T2
ON T1.ID=T2.ID
LEFT JOIN T3
ON T1.ID=T3.ID
WHERE T1.ID=123
```