

MAGIC SOFTWARE ENTERPRISES LTD.

Magic eDeveloper

Part of the Magic eBusiness Platform

Magic
eBusiness
PlatformTM **v9**

Magic コンポーネント開発

著作権表示と免責事項

The information in this document is subject to change without prior notice and does not represent a commitment on the part of MSE(Magic Software Enterprises Ltd.) and MSJ(Magic Software Japan K.K.).

MSE makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose.

The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms and conditions of the license agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement.

No part of this document and/or databases may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or information recording and retrieval systems, for any purpose other than the purchaser's personal use, without the prior express written permission of MSE.

All references made to third party trademarks are for informational purposes only regarding compatibility with the products of Magic Software Enterprises Ltd.

Unless otherwise noted, all names of companies, products, street addresses, and persons contained herein are part of a completely fictitious scenario or scenarios and are designed solely to document the use of Magic.

MagicR is a registered trademark of Magic Software Enterprises Ltd.

PC/TCPR Network Software is a registered trademark of FTP Software Inc.

MicrosoftR and FrontPageR are registered trademarks, and WindowsTM, WindowsNTTM and ActiveXTM are trademarks of Microsoft Corp.

MacromediaR DreamweaverR is a registered trademark of Macromedia, Inc.

VeriSignR is a registered trademark of VeriSign, Inc.

Clip art images copyright by Presentation Task Force, a registered trademark of New Vision Technologies Inc.

All other product names are trademarks or registered trademarks of their respective holders.

Copyright 2002 by Magic Software Enterprises Ltd.and Magic Software Japan K.K. All rights reserved.

2002年7月16日

目次

1 コンポーネント

コンポーネントとは	4
なぜコンポーネントを使用する方が良いでしょう？	4

2 Magic コンポーネント

Magic コンポーネントはより効率的です	6
どのようにして Magic はオブジェクトを公開するのですか？	7
Magic コンポーネントタイプ	7

3 コンポーネントの作成と読み込み

インターフェイスを作成する	8
ホストアプリケーションにコンポーネントを読み込む	9
コンポーネントを使用した開発	9

4 効率的なアプリケーション・コンポーネント

いつコンポーネントを使用するべきでしょう？	12
-----------------------------	----

5 実行環境でのコンポーネントの動作

いつコンポーネントが読み込まれるのでしょうか？	14
実行環境でのタスクツリー	15
変数はどこに格納されるのでしょうか？	16

6 イベントとハンドラ

イベントハンドラはどのようになっていますか？	18
------------------------------	----

7 ネストした Magic コンポーネント

ネストしたコンポーネントとは？	21
-----------------------	----

8 あとがき

Chapter

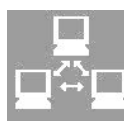
1

コンポーネント

コンポーネントとは？なぜ使用すべきなのでしょう？

コンポーネントを使用すると、短期間にアプリケーションの開発と実行を行うことができます。短期間でのアプリケーション開発のキーは、開発すべき仕様に合わせた既存アプリケーション・コンポーネントの再利用にあります。

コンポーネントとは



コンポーネントとは、全体構造の組立てに役立つ、ある程度の大きさの構造体、もしくはその一部と定義することができます。これをソフトウェア・コンポーネントに当てはめると、あらかじめ決められたインターフェースに基づいて意味のあるサービスを記述 / 実行することのできるアプリケーションと定義できます。

コンポーネントはカプセル化されており、インターフェースとして提供するサービスのみが公開されています。これにより、コンポーネントの実装部分は外部に対して隠蔽されることとなります。

コンポーネントは、例えば会計処理のようなビジネスロジック全体にすることも可能ですが、ID 番号の有効性チェックのような、より小さいものを作ることも可能です。

コンポーネントを使用する時には、それがどのように処理を行なうか（実装方法）ではなく、それは何をするコンポーネントか（使用目的）にフォーカスする必要があります。実装方法と使用目的の分離により、コンポーネントへの変更とそれを使用するアプリケーションを分離することが可能です。

なぜコンポーネントを使用する方が良いでしょう？



全てのアプリケーション開発者には同じゴール（最も効率的なアプリケーションを最も短い時間で開発すること）があります。また開発者は開発環境で既に利用できるサービスを用いることが、そのゴールに到達するのに必要であることも知っています。この部分がコンポーネントが最も有益な部分です。

コンポーネントベースの開発の特徴は、コンポーネントとして設計されたビジネスサービスが再利用可能であり、アップグレードと配布実行が簡単に行える事です。

コンポーネントはホストアプリケーションからは独立しているため、ホストアプリケーションとは無関係に修正およびアップグレードを行なう事が可能です。これにより、アップグレード、カスタマイズ、保守が容易になります。例えばプログラミングミスが発見された場合、不具合のあるコンポーネントだけを修正し、顧客へ送付すればよく、アプリケーション全体を変更する必要はありません。この方法でアプリケーションを常に最新の状態に保っておくことが可能です。

コンポーネントを利用して配布すると、アプリケーションの機能追加が行いやすくなります。お客様がコンポーネント・ベースのパッケージとして、一つのモジュールだけ購入されたとします。お客様が将来、他のモジュールを購入した場合、配布すべきなのは新しいコンポーネントのみです。例えば、「簿記モジュール」と「税金モジュール」のついた会計パッケージを買ったお客様が、将来「株価取得モジュール」を購入し、組み込むことができます。

コンポーネントのライブラリが充実してくるにつれ、アプリケーション開発、あるいはカスタマイズ能力が向上してくるでしょう。例えば、「株価取得モジュール」は最初は会計アプリケーションのパーツに過ぎなかったものが、e コマースアプリケーションのパーツにも使用できます。そうすると、e コマースアプリケーションの開発に要する時間は劇的に減少します。

つまり、コンポーネントを使用した開発はマーケットに出るまでの期間を短縮するだけでなく、ダイナミックに変化する現在のビジネス環境の需要に対し、素早い対応を可能にします。

Magic コンポーネント

Magic コンポーネントとは？ 通常のコンポーネントとの違いは？

Magic コンポーネントは通常の Magic アプリケーションです。Magic 開発者として、Magic コンポーネントを作成するのに新しく何かを学ぶ必要はありません。必要なことは外部に対して公開する特性を設定するだけです。この章ではその方法について簡単に説明します。詳細については『*Magic リファレンス*』を参照してください。

Magic コンポーネントはより効率的です



Magic コンポーネントは Magic アプリケーションであるため、通常のコンポーネント以上の利便性を提供します。Magic コンポーネントはホストアプリケーションの機能の一部を提供するコンポーネントとしても使用できますし、単独のアプリケーションとして使用することも可能です。

Magic コンポーネントは他のコンポーネントのホストアプリケーションとなることも可能です。つまり、Magic コンポーネントはネスト使用が可能です。Magic コンポーネントは Magic コントロールファイル (MCF) または Magic フラットファイル (MFF) のどちらの形態でも提供することが可能です。

公開することができるもの

一般的にコンポーネントとは、主にメソッドやプログラムが公開されますが、Magic コンポーネントでは以下のオブジェクトを公開することができます。

- モデル
- テーブル
- プログラム
- ヘルプ
- 権利
- イベント
- 環境設定

どのようにして Magic はオブジェクトを公開するのですか？

Magic コンポーネントのオブジェクトを公開する方法は簡単です。公開したいオブジェクトに対して、公開名をつけるだけです。図 2-1 は外部に対して公開するプログラムの例を示しています。この場合、「Send to Excel」というプログラムが「SendToExcel」という公開名で外部に公開されます。

プログラムリポジトリ				
#	名前	フォルダ	公開プログラム名	最終更新日
1	Main Program			1920/02/02
2				
3	Prepare to send data to Excel			1971/02/01
4	Actual conversation with Excel			1971/02/01
5				
6	Send to Excel		SendToExcel	1920/02/02
7	Data display			1980/02/02
8				
9	Load Excel and worksheet	Object loading		1980/02/02

図 2-1 公開プログラム名

このように新しい知識なしに、Magic アプリケーションをコンポーネント化することができます。

注意： メインプログラムには公開名を付けることはできません。従って公開されることはありません。

メインプログラムのイベントのみ公開できます。

Magic コンポーネントタイプ

Magic コンポーネントは必ずしも他の Magic アプリケーションをホストとする必要はありません。Magic コンポーネントは EJB として J2EE アプリケーションをホストとすることも可能です。また Web サービスプロバイダとなることも可能ですが、本ドキュメントでは扱いません。

注意： コンポーネントが EJB や Web サービスとして作成されているときは、プログラムのみが公開可能です。

この機能は現在、日本語版ではサポートされていません。

コンポーネントの作成と読み込み

どのようにコンポーネント・インタフェースを作るのですか？

この章ではインタフェースの作成方法とホストアプリケーションにコンポーネントを読み込む方法を説明します。ツールメニューから選択できる Magic コンポーネント・ビルダを使用して、コンポーネントのインタフェースを構築します。

インターフェイスを作成する



コンポーネント・ビルダは Magic コンポーネント・インタフェース (MCI) ファイルを作成し、ホストアプリケーションへのインタフェースを提供します。MCI ファイルは単純なテキストファイルです。コンポーネント・ビルダを使用してオブジェクトを公開したり、非公開に変更したりすることが出来ます。

Magic コンポーネント・ビルダはコンポーネント情報をデータベースで保存し、インタフェースの作成や変更を容易にします。コンポーネント・ビルダは、公開名を持つ全てのオブジェクトをデータベースに追加できます。さらに、コンポーネント・ビルダでは様々な環境設定情報を公開することが可能です。例えば次の情報が公開できます。

- サーバ
- サービス
- データベース
- 論理名
- 環境設定

公開可能な環境設定情報は『*Magic リファレンス*』に掲載されています。

コンポーネント・ビルダは、コンポーネントに対するヘルプファイルとそのリンクキーを登録することで、コンポーネントを利用する開発者への情報を提供することも可能です。

いつインタフェースを変更すべきですか？

以下の場合、新しいインターフェイスを作成する必要があります。：

- 公開していたオブジェクトが削除されたとき
- 新しいオブジェクトを公開する必要が発生したとき

ホストアプリケーションにコンポーネントを読み込む



コンポーネントを読み込むには、まずホストアプリケーションの中からコンポーネントリポジトリを開きます。空のエントリからズームするか、メニューから [読込 / 再読込] メニューを選択すると、あらかじめ作成された MCI ファイルの一覧から、読込みたいコンポーネントを選択できるようになります。MCI ファイルが選択されると、Magic は対応するコンポーネントを現在のアプリケーションへ読込み、公開されているコンポーネントの特性がホストアプリケーションで使用可能になります。

コンポーネントリポジトリ			
#	名前	説明	フォルダ
1	Excel connectivity		

図 3-1A ExcelConnectivity という名前の読込まれたコンポーネント

インタフェースに変更が発生した場合、ホストアプリケーションの修正が必要になることもあります。常にそれが必要であるとは限りません。新たなオブジェクトが公開されたときは、そのオブジェクトをホストアプリで使用しない限りは、コンポーネントの再読込みを行なう必要はありません。

コンポーネントを使用した開発

コンポーネント・リポジトリでコンポーネントを読込んだときに表示される全てのリポジトリ・アイテムは、アプリケーション内で使用可能です。

例として、6 ページの「Send to Excel」プログラムを取り上げます。このプログラムをどのようにコールすれば良いでしょうか？ このプログラムは既にアプリケーションの一部となっているので、通常の方法（コールプログラム）でプログラムを呼ぶだけです。

下の図 3-2 に示されているように、通常の Magic プログラムのコールと同様にコールされています。パラメータは二つで戻り値があります。ここでの唯一の違いは名前です。プログラム名の前にコンポーネント名が表示されています。

#	処理コマンド	内容	
1	例外 V=変数	1 Return Code	代入: 0
2			
3	コール P=プログラム	Excel connectivity.Send to E	戻り: 2
4			
5	アクション	3 MMSTOP ()	

図 3-2 コンポーネントを使用する

全ての公開オブジェクトが、このプログラムと同じ手法で使用することができます。

環境設定と特性

6 ページで触れたように、様々な環境設定情報を公開することができます。サーバ、サービス、データベース、論理名の場合、各エントリそれぞれに対してホストアプリケーションの Magic.ini ファイルの対応するセクションに行が追加されます。

これらの特性は一般的にアプリケーションの設定やローカライズに使用されます。それらは現在の環境に合わせる必要があります。論理名を除いて、全ての特性はその値と共に公開されたままの状態です。開発者はそれらの値を注意深く調整する必要があります。

論理名の解釈は開発マシン上で行なわれるため、論理名はコンポーネントを作成したマシンと異なる値を持っている可能性があります。従って、あらかじめ実行名を持たない論理名を作成してください。これにより、開発者はホストコンピュータにおいて適切な値を付加することができます。

コンポーネントを読込んだ後はホストアプリケーションの一部となるので、これらの設定はアプリケーション内でアクセスすることが可能です。

重要： 公開されているコンポーネントの環境設定が現在のアプリケーションの設定とは異なる場合、たとえば日付モードは西暦開始年など、公開されている特性はコンポーネント内のプログラムにおいてのみ有効となります。これらの値は一般的に実行エンジンとの関連性があります。

コンポーネントのプログラムがコールされたとき、これらの値の異なる環境設定値はそのプログラム内でのみ有効になります。例えばホスト側のプログラムがコンポーネントのテーブルを使用するような場合でも、ホストアプリケーション側の環境設定値が使用されることとなります。

参考： Magic は公開名を使用してコンポーネント内のオブジェクトにアクセスします。したがって「Send to Excel」プログラムをコールしているにも関わらず、実際は「SendToExcel」をコールしていることとなります。開発者が異なるプログラムをコールしたいときは、公開名の「SendToExcel」を削除して他のプログラムにその公開名をつけてください。この場合、インタフェース（パラメータの数と書式）は同じになっている必要があります。

重要： 前述のように、Magic はコンポーネントのオブジェクトに対して公開名を使用してアクセスします。オブジェクトがインタフェースから削除されても、公開名を使用した呼び出しを受ける可能性が残ります。オブジェクトをコールされないようにするには、公開名を削除する必要があります。

効率的なアプリケーション・コンポーネント

コンポーネントを使用してアプリケーションを構築するには？

これまではコンポーネントとは何か、Magic コンポーネントとは何か、Magic コンポーネントの使用方法について述べてきました。この章ではコンポーネントを利用したアプリケーションの構築方法について説明します。

いつコンポーネントを使用すべきでしょう？



アプリケーションを迅速に開発するキーは既存コードの再利用性にあります。すなわち、一度だけコーディングし、それを何度も再利用することです。

コンポーネントに何を配置すべきか、コンポーネント型アプリケーションの設計には何に気を配るべきかといった事に関する、明確な規定や制限はありませんが、いくつかのガイドラインとテクニックを提供いたします。

アプリケーションに多くのモデルが使用されている場合、これらをコンポーネント化することが考えられます。モジュールが単独で存在し、単独で販売できるような場合も、そのモジュールをコンポーネント化できます。例えば、アプリケーションに「営業モジュール」、「経理モジュール」、「給与モジュール」等がある場合、各モジュールをコンポーネント化することができます。

コンポーネント化を推奨するオブジェクトは、以下のようなものです。

- 頻繁に使用されるモジュール
- 頻繁に使用されるヘルプと権利
- 頻繁に使用されるテーブルと、そのテーブルを使用したプログラム
- 頻繁に使用されるイベント
- アプリケーション全体で使用できるイベント・ハンドラ。19 ページで説明するようなグローバルハンドラ

以下のような場合にコンポーネントを使用することをお奨めします。

- システム内の複数箇所で行なわれている動作を探し、複数のアプリケーションで使用できるユーティリティ・コンポーネントとして作成します。

例えば、コンポーネント内に数字のパリティチェックを行なうプログラムを作成

する等。

- カスタマイズ.....全てのお客様に対してアプリケーションは同じものを使用するが、モデルや、インターフェイスなどが顧客ごとに僅かに異なる場合。例えば、メインのアプリケーションは同じで顧客ごとに異なった帳票が必要な場合、帳票機能をコンポーネント内に記述し、カスタマイズ用パッケージとすることが可能です。
- 特定のプログラムや機能が頻繁に変更される必要があるとき。例えば、会計アプリケーションをベースにしておき、税金計算モジュールのみを定期的に変更するといったことが可能です。
- アウトソーシング.....アプリケーション開発の一部をアウトソーシングする場合。

モデルを使用した簡単な例で説明します。シリアル番号を使用する単純なケースを考えます。この番号を「9 桁マイナス符号なし」の数値型の書式に設定するとします。この番号がアプリケーション内の複数のテーブルで使用され、プログラム中で変数として何回も使用されているとき、通常は毎回定義を行わず、モデルを使用します。

一つのアプリケーション内でなく、複数のアプリケーションで、同じシリアル番号を同じ書式で使用するような場合、このモデルをコンポーネント化することを推奨します。モデルを一カ所定義することで、全てのアプリケーションで使用できるようになります。

制限： オブジェクトが他のオブジェクトに対して内部参照を持っているとき、これらを分割してコンポーネントに入れることは有効ではありません。

例えば、あるテーブルのデータを使用するコンボボックスのモデルを作成し、そのテーブルでこのコンボボックスのモデルを使用する場合、一方のオブジェクト(テーブル)を公開せずに他方のオブジェクト(モデル)だけを公開することは無効です。

相互内部参照を持つオブジェクトテーブルが存在するとき、全てのモデルを一つのコンポーネントに入れ、全てのテーブルは別のコンポーネントに入れる、といったコンポーネント化を行なってはいけません。

実行環境でのコンポーネントの動作

実行環境ではコンポーネントとその中の変数はどのように動作するのでしょうか？

実行環境でコンポーネントはどのように動作するのでしょうか？どのタイミングで Magic はコンポーネントを読み込むのでしょうか？コンポーネントで定義された変数はどのような扱いになるのでしょうか？この章では実行環境についてのこれらの問題について説明します。

いつコンポーネントが読み込まれるのでしょうか？



コンポーネントリポジトリの特性で、実行中のどのタイミングでコンポーネントが読み込まれるかを指定することができます。コンポーネント特性の [即時有効] をチェックすると、ホストアプリケーションと同時にコンポーネントが読み込まれます。チェックをしないときは、最初にコンポーネントオブジェクトが呼ばれたときに読み込まれます。従って、読み込まれるタイミングはアプリケーションの構造に依存します。

コンポーネントが読み込まれるときには、メインプログラムも同時に読み込まれます。変数の初期化やメモリーテーブルの初期化など、メインプログラムにオペレーションが記述されている場合は、これらの処理は常に実行されます。メインプログラムは一度実行されるだけなので、メインプログラムの処理があるとき、即時有効にするかどうかで、動作が大きく異なる場合があります。

- 即時有効に設定されているとき.....ホストアプリケーションがコンポーネントのプログラムをコールした場合、コンポーネントのメインプログラムが再び実行されることは有りません。
- 即時有効に設定されていないとき.....ホストアプリケーションがコンポーネントのプログラムをコールした場合、コンポーネントのメインプログラムが実行されます。これはコンポーネントのロードと初期化が同時に行なわれるためです。以後、コンポーネントのプログラムがコールされても、メインプログラムが再び実行されることはありません。

コンポーネントがどのタイミングで読み込まれるかを吟味することは重要です。もしたくさんコンポーネントを使用し、それら全てがアプリケーションの起動と同時に読み込まれると、初期化プロセスに時間がかかる場合があります。この場合、全てのコンポーネントのリソースは即時に利用可能となりますが、メモリリソースを大量に消費します。あまりアクセスしないコンポーネントや、特定のユーザのみにアクセスされるコンポーネント等は、即時有効にすることは効果的ではありません。しかし、コンポーネントがモデル、テーブル、イベント、プログラム、またはその他の広く頻繁に使用されるアイテムを含む場合は、アプリケーション起動時に読み込むようにする必要があります。

実行環境でのタスクツリー

タスクツリーとは？ どのように現在のタスクがグローバルタスクと関連するのでしょうか？



プログラムが他のプログラムやタスクをコールするとき、実行環境にタスクツリーが構成されます。このツリーはメインプログラムから始まり、現在実行中のプログラムまで続きます。実行中のタスクは用意されている関数等を使用して、タスクツリー中の上位のタスクのいかなる値も参照することが可能です。関数に特定の上位タスクを伝えるには、タスクツリー中の位置に応じて順につけられるタスク番号と呼ばれる連続番号を使用します。タスクツリーの一番下、つまり最後のコールされたタスク番号は0（ゼロ）です。親タスクのタスク番号は1になります。

コンポーネントを含まない Magic アプリケーションでは、プログラム A がプログラム B をコールするプログラムは図 5-1 のようになっています。

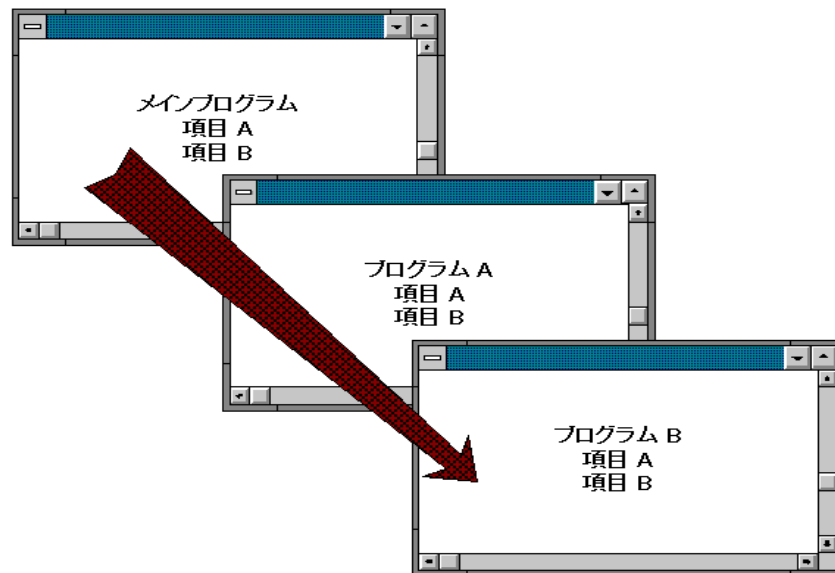


図 5-1 通常の実行環境のタスクツリー

この場合、実行タスクツリーは次のようになっています：

メインプログラム プログラム A プログラム B

プログラム B からタスクツリーを見たとき、プログラム B のタスク番号は「 0 」、プログラム A のタスク番号は「 1 」、メインプログラムは「 2 」です。図 5-2 のように、コンポーネント中のプログラムがコールされた場合はどうなるのでしょうか。

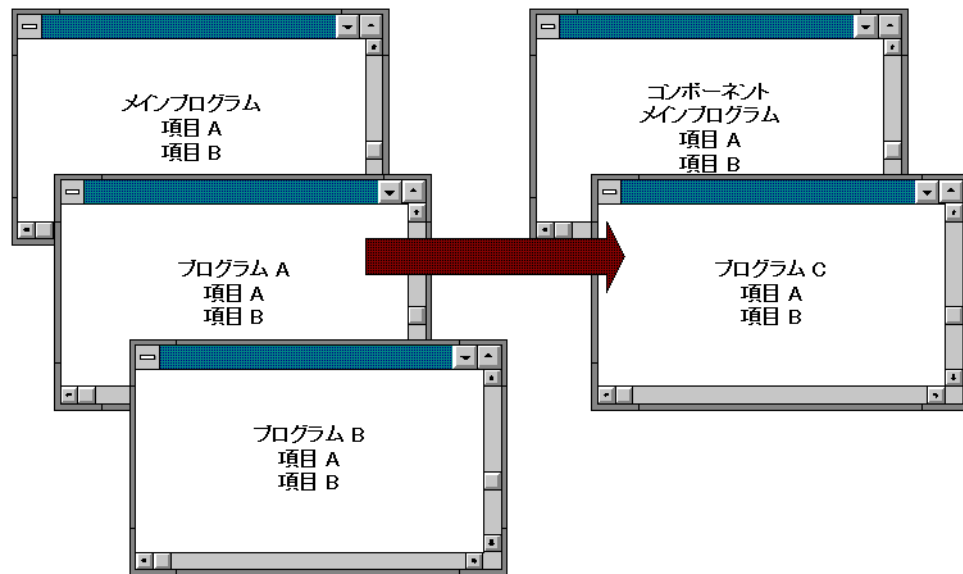


図 5-2 プログラムが、コンポーネント中のプログラムをコールする例

この場合、コンポーネントのメインプログラムによって少し複雑になり、実行タスクツリーは次のようになります。

メインプログラム プログラム A コンポーネント・メインプログラム プログラム C

プログラム C が実行中にタスクツリーを参照する場合、プログラム C のタスク番号が「 0 」、プログラム A のタスク番号が「 1 」、メインプログラムのタスク番号が「 2 」となります。コンポーネントのメインプログラムはタスク番号の対象になりません。

タスク番号を使用する Magic 関数には次のものがあります。

CHEIGHT, CLEFT, CLEFTMDI, COUNTER, CTOP, CTOPMDI, CURROW, CURRPOSITION, CWIDTH, DBCACHE, EOF, EOP, LASTPARK, LEVEL, LINE, PAGE, ROLLBACK, STAT, VARINP, VIEWMOD, WINBOX, WINHWN

変数はどこに格納されるのでしょうか？

下位のタスクから上位タスク中の変数にアクセスする場合があります。これはタスクツリーとよく似た方法で実現することができます。メインプログラムの最初の変数は A です。それに続く変数は現在実行中のプログラムまで順に番号（シンボル名）が付けられます。従って、15 ページの 図 5-1 で示すような通常の Magic アプリケーションでは、変数ツリーは次のようになります。

プログラム名	項目名	シンボル名 (リテラル)	シンボル名 (番号)
メインプログラム	A	'A'VAR	1
	B	'B'VAR	2
プログラム A	A	'C'VAR	3
	B	'D'VAR	4
プログラム B	A	'E'VAR	5
	B	'F'VAR	6

しかし、16 ページの 図 5-2 のようにコンポーネントがコールされている場合はどのようなようになるのでしょうか？この場合、コンポーネントのメインプログラムの変数が、最後のプログラムより前にタスクツリーに追加されます。従って、変数ツリーは次のようになります。

プログラム名	項目名	シンボル名 (リテラル)	シンボル名 (番号)
メインプログラム	A	'A'VAR	1
	B	'B'VAR	2
プログラム A	A	'C'VAR	3
	B	'D'VAR	4
メインプログラム (コンポーネント)	C	'E'VAR	5
	D	'F'VAR	6
プログラム C	C	'G'VAR	7
	D	'H'VAR	8

ここで見てきたように、タスク番号の場合と異なり、メインプログラムの変数はシンボル名を指定してアクセスすることができます。

変数ツリーを扱う Magic 関数には次のものがあります。

VARATTR, VARCURR, VARCURRN, VARINDEX, VARINP, VARMOD, VARPIC, VARPREV, VARSET

注意： ここでのシンボル名は、関数で使用する場合の指定でのみ有効で、ホストプログラムの項目テーブルにはコンポーネントプログラムの変数は表示されません。

Chapter 6

イベントとハンドラ

コンポーネントベースのアプリケーションでのイベントハンドリングはどのように動作するのでしょうか？

イベントドリブンとは、イベントを使用して処理することです。この章ではイベントとハンドラが、コンポーネントを使用したアプリケーションでどのように動作するのかを説明します。

イベントハンドラはどのようになっていますか？



前述したように、メインプログラムに登録されたイベントだけが、ホストアプリケーションに対して公開することが可能です。公開されたイベントはプログラムのおときと同じようにホストアプリケーションで使用することができます。コンポーネントのイベントは通常のイベントと全く同じ方法で処理されます。

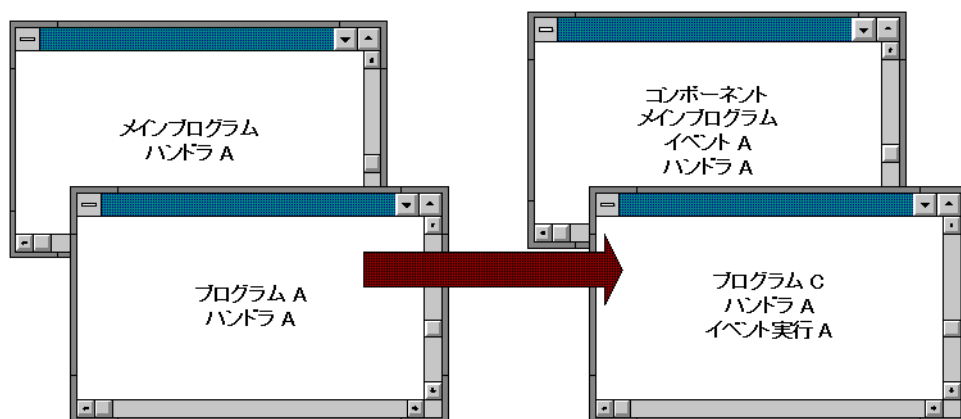


図 6-1 コンポーネントプログラムがイベントを起動した場合

図 6-1 のイベントが起動された場合の図を見てください。ここではコンポーネントのプログラムがイベントを起動し、タスクツリーに沿ってイベントが処理されます。つまり、このイベントに対するハンドラの検索順序は、最初にプログラム C 内を検索され、コンポーネントのメインプログラム、プログラム A、最後にホストアプリケーションのメインプログラムという順に検索されます。この例のように、各プログラム内に対応するハンドラがある場合、それらのハンドラはタスクツリーの順序に沿って実行されます。ハンドラの伝播の特性が「No」の時は、そのハンドラが実行される最後のハンドラとなります。

もし、プログラム A がイベントを実行した場合、ハンドラの検索はホストアプリケーション内でのみ行なわれます。つまり、プログラム A とホストアプリケーションのメインプログラム内でのみ検索が行なわれます

イベントの処理方法の詳細については、『イベントドリブンアーキテクチャ』のホワイトペーパーを参照してください。

グローバルハンドラ

コンポーネントのメインプログラムにハンドラが記述されている場合、ホスト側のプログラムでこのハンドラに対応したイベントを発生させてもイベントは処理されません。なぜならハンドラは実行中のタスクツリーの中には存在しないからです。しかし、そのハンドラが下図で示すようにグローバルのスコープを持っていると、イベントが処理されるようになります。

№	レベル	イベント	詳細	スコープ	伝播	有効	処理
1	T=奴カ	P=前処理					0
2	T=奴カ	S=後処理					0
3	R=レコト	M=メイン					10
4	H=ハンドラ	Create application instance	コントロール:	G=グローバル	No	Yes	3
5	H=ハンドラ	Destroy application instance	コントロール:	S=サブツリー	No	Yes	1
6	H=ハンドラ	Error handler	コントロール:	S=サブツリー	No	Yes	12

図 6-2 グローバルハンドラ

それでは、グローバルハンドラはいつ処理されるのでしょうか？先ほどの例をもとに説明します。今回はコンポーネントのメインプログラムに定義されたグローバルハンドラを使用するとします。

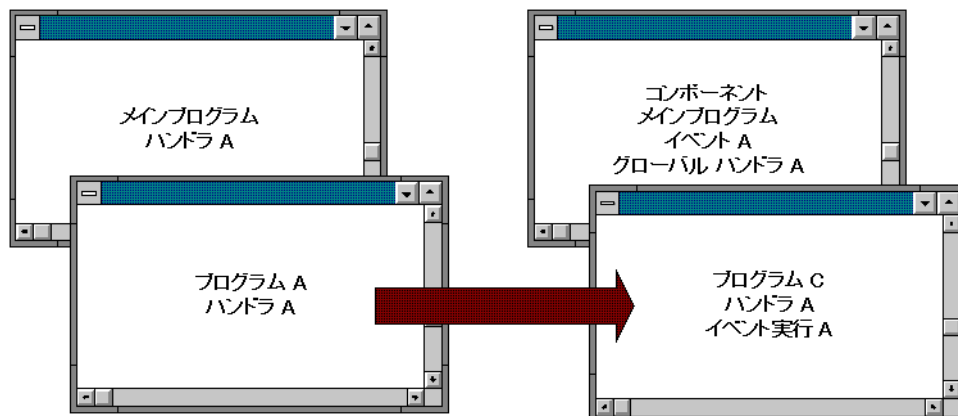


図 6-3 グローバルハンドラ

先ほどと同じようにプログラム C でイベントを発行した場合、ハンドラの検索と伝播の順序が少し異なります。順序は以下のようになります

1. プログラム C
2. プログラム A
3. ホストのメインプログラム
4. コンポーネントのグローバルハンドラ

ここから分かるように、グローバルハンドラは実行タスクツリーの最後に処理されます。グローバルハンドラを使用する主なメリットは、アプリケーション全体で、イベントに対してデフォルト動作を提供できることです。

重要： コンポーネント内に定義されていないイベントに対して、グローバルハンドラを定義することも可能です。内部イベントやシステムイベントに対してのグローバルハンドラを定義することも可能です。

ネストした Magic コンポーネント

ネストしたコンポーネントはどのように扱われるのでしょうか？

本書で説明しているコンポーネントの手法を使用すると、アプリケーション中にネストしたコンポーネント、つまりコンポーネント中に別のコンポーネントがある構造を作るとは、簡単に想像できると思います。この章ではネストしたコンポーネントの扱い方について説明します。

ネストしたコンポーネントとは？



コンポーネントを使用しているアプリケーションがあるとします。このアプリケーションをコンポーネントとして、さらに別のホストアプリケーションで使用することが可能です。もしくは、頻繁に使用されるモデル、テーブル、プログラムなどのオブジェクトを含んだコンポーネントを使用して何かのモジュールを作成し、そのモジュールを使用したパッケージを作成することも可能です。

どちらのケースでも、ネストしたコンポーネントのシステムを使用することになります。図 7-1 ではさらに複雑にネストしたコンポーネント・システムの例を表しています。

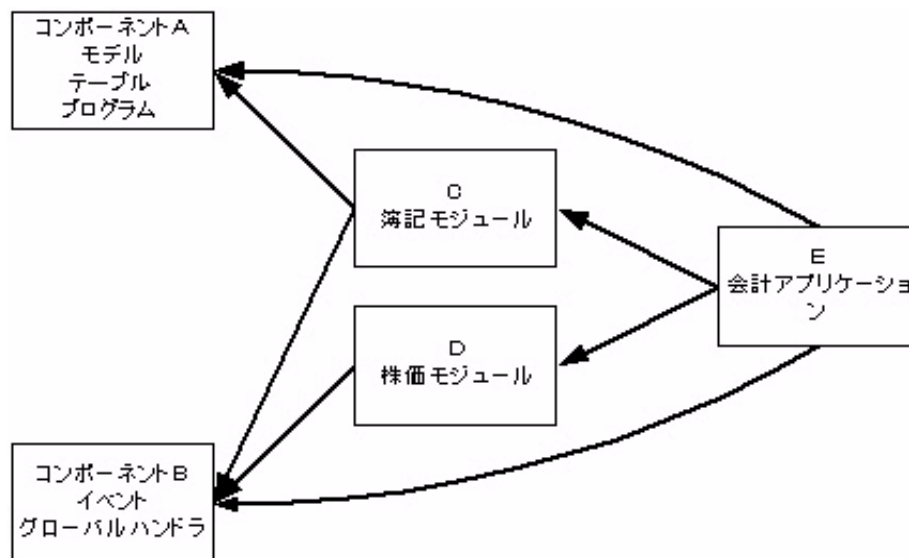


図 7-1 ネストしたコンポーネントシステムの例

この例では、コンポーネント A に頻繁に使用されるモデル、テーブル、プログラムが含まれており、コンポーネント B には頻繁に使用されるイベントとグローバルハンドラが含まれています。この二つのコンポーネントは簿記モジュール C と株価モジュール D で使用されています。この二つのモジュールは、コンポーネントとして会計アプリケーション E で使用されています。また会計アプリケーション E ではコンポーネント A、B 両方も同時に使用されています。

このような場合、コンポーネントは 3 回使用されており、理屈では 3 回読み込みが発生すべきですが、Magic エンジンはコンポーネントがいつメモリに読み込まれたか記憶しているため、インスタンス毎に再読み込みが行なわれることはありません。

参考： Magic はコンポーネントをフルネームで区別しています。あるコンポーネントがコンテナやアプリケーションに「c:¥Magic¥comp.mcf」として読み込まれ、別のコンテナに「¥myserver¥Magic¥comp.mcf」として読み込まれた場合、それらが仮に同じものであっても Magic はそれらを二つの異なるコンポーネント名として認識します。

制限： ネストしているコンポーネントのオブジェクトを公開することはできません。つまりホストアプリケーションは自分が使用しているコンポーネントに起因するオブジェクトを公開することはできません。

再帰的な（循環参照している）コンポーネントは実行エラーとなります。これは、コンポーネント A を使用しているコンポーネント B が、コンポーネント A と同一のホストアプリケーション A で使用されるような場合に発生します。

Chapter

8

あとがき

この新しい Magic のプログラミングスタイルを使う前に

このドキュメントがコンポーネントベースの Magic プログラミングを始めるのに役立つことができましたでしょうか。この技術資料では Magic コンポーネントの定義、制限、コンポーネントベース開発のメリット、コンポーネントの作成方法、オブジェクトの取り扱い、及び実行モードでのコンポーネントの動作について説明致しました。

コンポーネントベースの Magic プログラミングを使用すると、より洗練され、より強固なアプリケーションを短期間に開発することができます。つまり、Magic コンポーネントは RROI (Rapid Return on Investment) を提供いたします。