# コンポーネント開発
# Magic xpa 3.x


OUTPERFORM THE FUTURE™

本マニュアルに記載の内容は、将来予告なしに変更することがあります。これらの情報について MSE（Magic Software Enterprises Ltd.）および MSJ（Magic Software Japan K.K.）は、いかなる責任も負いません。

本マニュアルの内容につきましては、万全を期して作成していますが、万一誤りや不正確な記述があったとしても、MSE および MSJ はいかなる責任、債務も負いません。

MSE および MSJ は、この製品の商業価値や特定の用途に対する適合性の保証を含め、この製品に関する明示的、あるいは黙示的な保証は一切していません。

本マニュアルに記載のソフトウェアは、製品の使用許諾契約書に記載の条件に同意をされたライセンス所有者に対してのみ供給されるものです。同ライセンスの許可する条件のもとでのみ、使用または複製することが許されます。

当該ライセンスが特に許可している場合を除いては、いかなる媒体へも複製することはできません。ライセンス所有者自身の個人使用目的で行う場合を除き、MSE または MSJ の書面による事前の許可なしでは、いかなる条件下でも、本マニュアルのいかなる部分も、電子的、機械的、撮影、録音、その他のいかなる手段によっても、コピー、検索システムへの記憶、電送を行うことはできません。

サードパーティ各社商標の引用は、MSE および MSJ の製品に対するコンパチビリティに関しての情報提供のみを目的としてなされるものです。

本マニュアルにおいて、説明のためにサンプルとして引用されている会社名、製品名、住所、人物は、特に断り書きのないかぎり、すべて架空のものであり、実在のものについて言及するものではありません。

Magic は Magic Software Japan K.K. 登録商標です。
Magic xpa は Magic Software Enterprises Ltd. のイスラエルその他の国での商標または登録商標です。
Magic xpa Enterprise Studio、Magic xpa Enterprise Client、Magic xpa Enterprise Server および Magic xpa RIA Server は Magic Software Japan K.K. の商標です。
Pervasive.SQL® は Pervasive Software, Inc. の商標です。
IBM®, iSeries™, xSeries®, DB2® および WebSphere® は、IBM Corporation の商標または登録商標です。
Microsoft® および FrontPage® は、Microsoft Corporation の登録商標です。また、Windows™, WindowsNT™ および ActiveX ™ は Microsoft Corporation の商標です。
Oracle® は Oracle Corporation の登録商標です。
Linux® は Linus Torvalds の登録商標です。
GLOBEtrotter® と FLEXlm® は、Macrovision Corporation の登録商標です。
Interstage® は、富士通株式会社の登録商標です。
JBoss™ は、JBoss Inc. の商標です。
Systinet™ は、Hewlet-Packard Development Company の商標です。

一般に、会社名、製品名は各社の商標または登録商標です。
MSE および MSJ は、本製品の使用またはその使用によってもたらされる結果に関する保証や告知は一切していません。この製品のもたらす結果およびパフォーマンスに関する危険性は、すべてユーザが責任を負うものとします。

この製品を使用した結果、または使用不可能な結果生じた間接的、偶発的、副次的な損害（営利損失、業務中断、業務情報の損失などの損害も含む）に関し、事前に損害の可能性が勧告されていた場合であっても、MSE および MSJ、その管理者、役員、従業員、代理人は、いかなる場合にも一切責任を負いません。



20012 年 10 月 31 日

# 1 コンポーネント



# 2 Magic コンポーネント



# 3 コンポーネントの作成と読込み



# 4 効率的なアプリケーションコンポーネント



# 5 実行環境でのコンポーネントの動作



# 6 イベントとハンドラ



# 7 ネストされた Magic コンポーネント

# 第 1 章 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フェースに基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になります。

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のでしょうか？

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。

# 第 2 章 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 xpa 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『リファレンスヘルプ』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。

### 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
- データソース
- プログラム
- ヘルプ
- 権利
- イベント
- ユーザ定義関数
- 環境設定

## どのようにして Magic xpa 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 3-1 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、「MSMQ OpenQueue」というプログラムが「MSMQ Open Queue」という公開名で外部に公開されます。


図 2-1 公開されるプログラムの例

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。メインプログラムに定義されたイベントやユーザ定義関数のみ公開できます。

## 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や COM オブジェクトに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、COM オブジェクト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。

プログラムを公開する場合は、［外部］のチェックボックスをチェックする必要があります。

# 第 3 章 コンポーネントの作成と読込み

*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プロジェクトにコンポーネントを読み込む方法を説明します。オプションメニューから選択できる Magic コンポーネントインタフェースビルダを使用して、コンポーネントのインタフェースファイルを構築します。

## インターフェイスを作成する

コンポーネントインタフェースビルダは eDeveloper コンポーネントインタフェース（ECI）ファイルを作成し、ホストプロジェクトへのインタフェースを提供します。ECI ファイルは単純なテキストファイルです。コンポーネントインタフェース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できます。

コンポーネントインタフェースビルダはコンポーネント情報をデータベースに保存し、インタフェースの作成や変更を容易にします。コンポーネントインタフェースビルダは、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、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『リファレンスヘルプ』に掲載されています。

コンポーネントインタフェースビルダは、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場合
- 新しいオブジェクトを公開する必要が発生した場合
- ［プログラム］リポジトリの［外部］カラムのチェックボックスが変更された場合

## ホストプロジェクト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、まずホストプロジェクトの中から［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 ECI 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。ECI ファイルが選択されると、Magic xpa は対応するコンポーネントを現在のプロジェクトへ読込み、公開されているコンポーネントの特性がホストプロジェクトで利用可能になります。

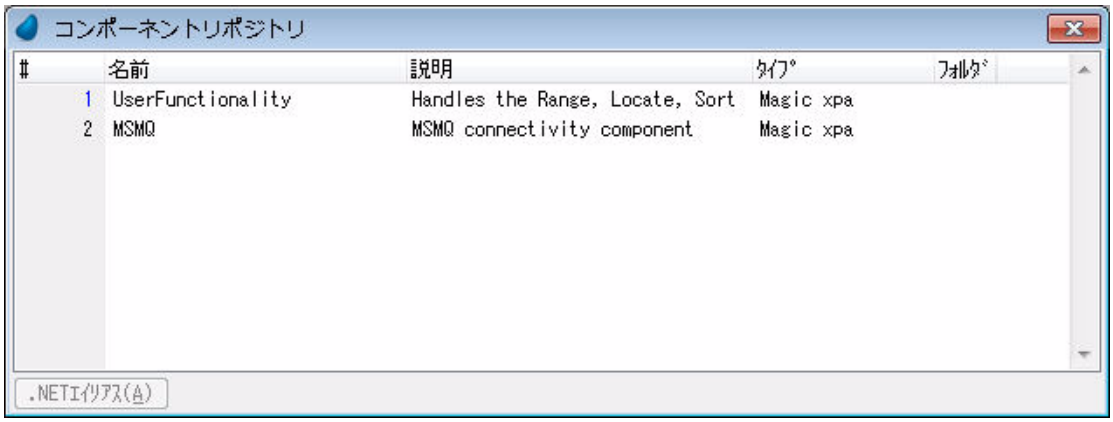


図 3-1 MSMQ とい名前で読込まれたコンポーネント

インタフェースに変更が発生した場合、ホストプロジェクトの修正が必要になることもありますが、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プロジェクト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項目は、プロジェクト内で使用可能です。例として、「MSMQ Open Queue」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プロジェクトの一部となっているので、通常の方法（コール処理プログラム）でプログラムを呼ぶだけです。

下の図 2-2 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。

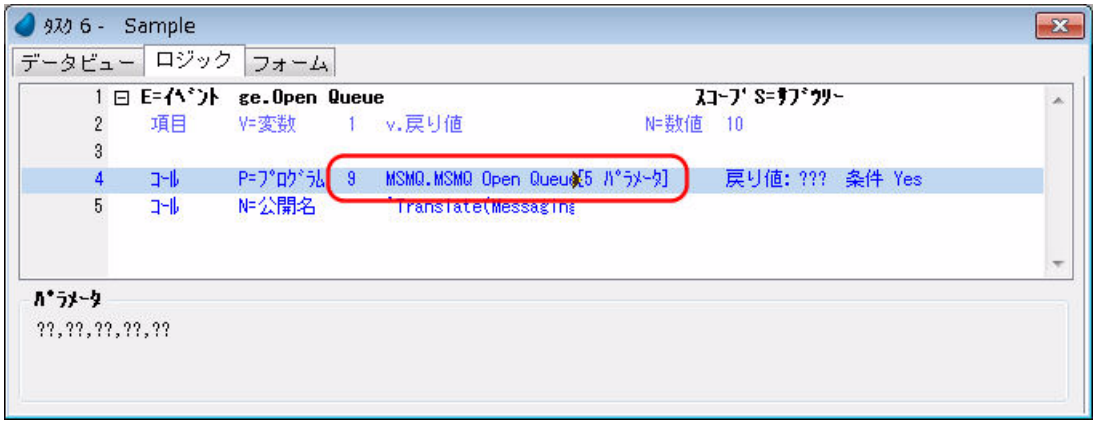


図 3-2 コンポーネントの使用例

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

2 ページで触れるように、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プロジェクトの 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プロジェクト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読込まれます。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者側の PC 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 PC 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 PC 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プロジェクトの一部となるので、これらの設定はプロジェクト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プロジェクト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、ホストプロジェクト側の環境設定値が使用されることになります。

**参考：**

Magic xpa 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「Open Queue」プログラムをコールしているにも関らず、実際は「MSMQOpenQueue」をコールしていることになります。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「Open Queue」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 xpa 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。

J2EE や Web サービス、COM インタフェースに対しては、［外部］カラムのチェックボックスをクリアするだけで十分です。これによって、プログラムは公開されなくなります。

# 第 4 章 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プロジェクト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るべきでしょう？

プロジェクト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、コンポーネント型プロジェクト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。

プロジェクト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、プロジェクトに「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デル
- 頻繁に使用されるヘルプと権利
- 頻繁に使用されるデータソースと、そのデータソースを使用したプログラム
- 頻繁に使用されるイベント
- プロジェクト全体で使用できるイベントハンドラ。13 ページで説明するようなグローバルハンドラ
- システム内の複数箇所で行なわれている動作を探し、複数のプロジェクト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プロジェクト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、メインのプロジェクト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……プロジェクト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。この番号がプロジェクト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。一つのプロジェクト内でなく、複数のプロジェクトで、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、全てのプロジェクト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、あるデータソースのデータを使用するコンボボックスのモデルを作成し、そのデータソース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（データソース）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、全てのデータソース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。

# 第 5 章 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 xpa はコンポーネントを読込むのでしょうか？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］特性をチェックすると、ホストプロジェクトと同時にコンポーネントが読込まれます。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込まれます。従って、読込まれるタイミングはプロジェクトの構造に依存します。

コンポーネントが読込まれるときには、メインプログラムも同時に読込まれます。変数の初期化やメモリテーブルの初期化など、メインプログラムに処理コマンド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場合**……ホストプロジェクトがコンポーネントのプログラムを呼び出した場合、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ありません。
- **即時有効に設定されていない場合**……ホストプロジェクトがコンポーネントのプログラムを呼び出した場合、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呼び出されても、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込まれるかを吟味することは重要です。もしたくさんの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、データソース、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込むようにする必要が考えられます。

## 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。

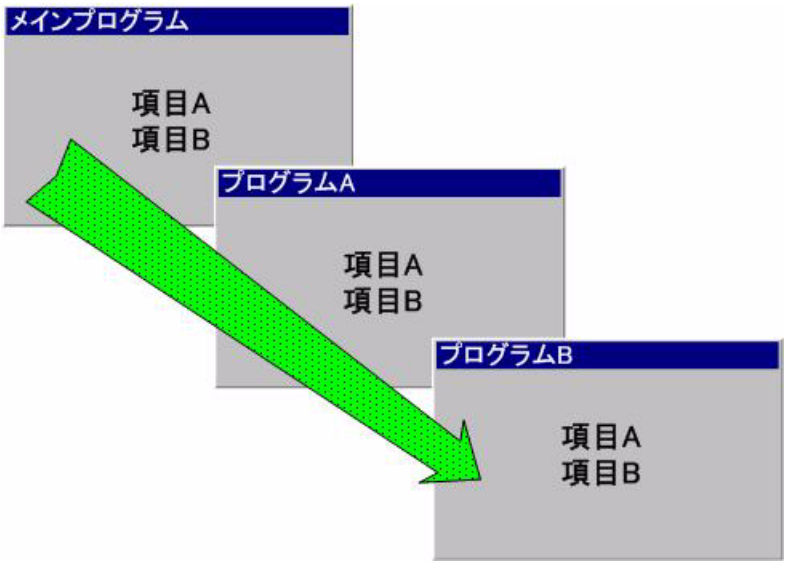


図 5-1 通常の実行環境のタスクツリー

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。

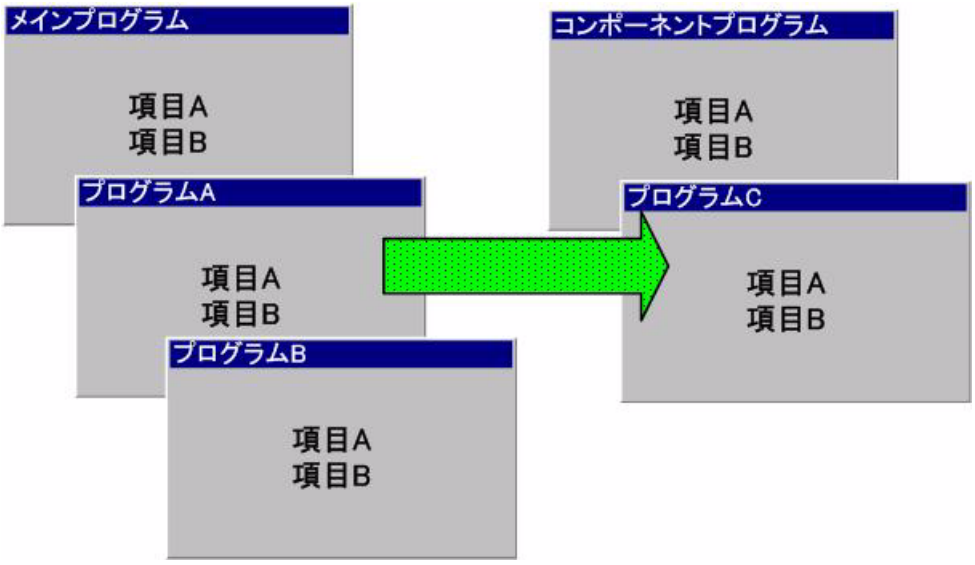


図 5-2 プログラムが、コンポーネント中のプログラムをコールする例

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。

## 項目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項目にアクセスする場合が数多くあります。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項目は A です。それに続く項目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、14 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、項目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、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項目が、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、項目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項目はシンボル名を指定してアクセスすることができます。項目ツリーを扱う Magic 関数には次のものがあります。

VarAattr, VarCurr, VarCurrn, VarDbName, VarIndex, VarInp, VarMod, VarName, VarPic, VarPrev, VarSset

**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。

## 動的にコンポーネントをロードするにはどのようにするのでしょうか？

動的または条件付きでコンポーネントをロードしたい場合があります。ある条件に応じて一定のコンポーネントロードし、異なる条件の場合は異なるコンポーネントをロードすることが要求されるかもしれません。また、条件に従って、動的に同じコンポーネントの異なるプログラムを呼ぶ必要が出てくることもあります。

このような場合、コール公開名処理コマンドを使用することで可能になります。これは、プログラムの公開名を指定して通常のプログラムを呼び出す方法です。プロジェクト内に公開名が見つからない場合は、プログラムが存在しないものして処理されます。

### どのようにして動的にプログラムを呼びだすのでしょうか？

コール公開名処理コマンドを使用して、アプリケーションとそこに定義された公開プログラムを呼び出します。

呼び出されたアプリケーションは、通常のコンポーネントのように動作します。この場合、この章に説明された通常の方法で読み込まれます。アプリケーションが読み込まれると、公開プログラムが呼び出されます。

呼び出されるキャビネットファイルとプログラムの指定は、式によって行われます。

図 5-3 は、どのように動的なプログラムを定義するかを指名した例です。

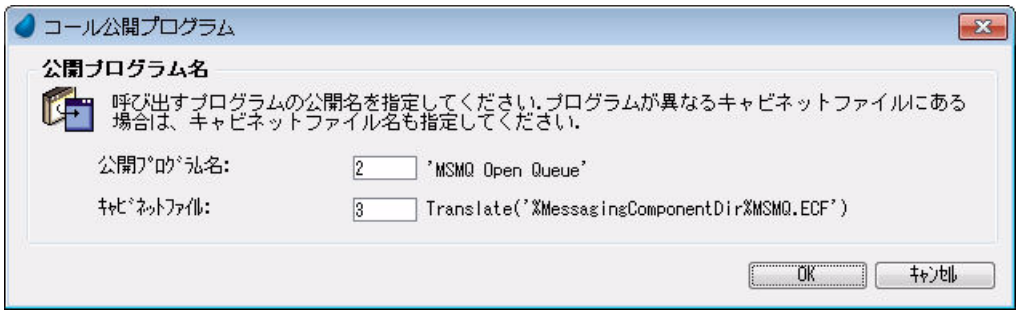


図 5-3 動的なプログラムを呼び出す例

**注意：**

公開名が存在していない場合、存在していないプログラムを呼び出した場合と同じような動作になります。

**注意：**

キャビネットファイルの式が、空白と評価された場合、現在のプロジェクト内の公開プログラムとして扱われます。

# 第 6 章 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（ロジックユニット）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。

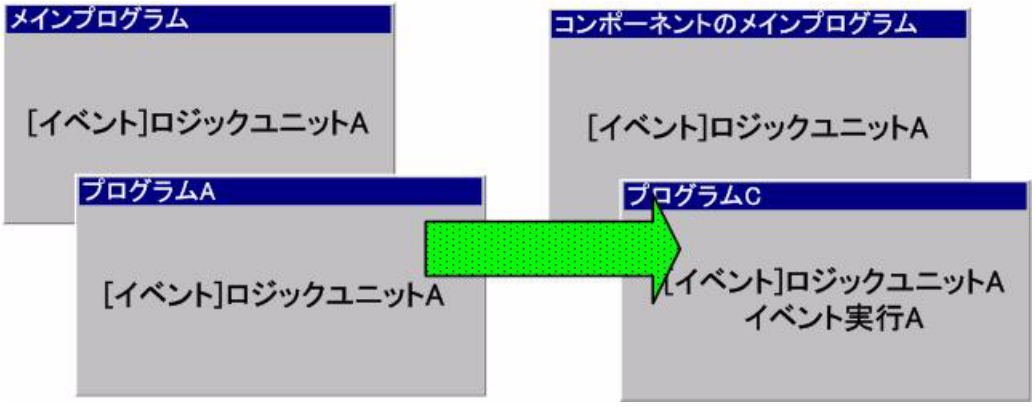


図 6-1 コンポーネントプログラムがイベントを発行した場合

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発行し、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ロジックユニットの検索順序は、最初にプログラム C 内が検索され、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ロジックユニットがある場合、それらのロジックユニットはタスクツリーの順序に沿って実行されます。ロジックユニットの［伝播］特性が「No」の時は、そのロジックユニットが実行される最後のロジックユニットとなります。

もし、プログラム A がイベントを発行した場合、ロジックユニット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ロジックユニットが記述されている場合、ホスト側のプログラムでこのロジックユニット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。なぜならロジックユニット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、そのロジックユニットが図 6-2 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。

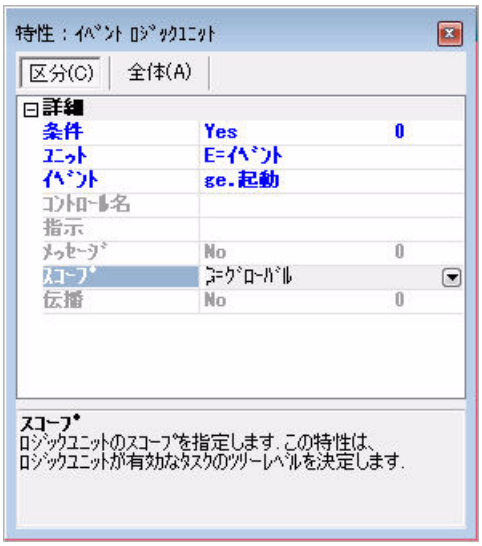


図 6-2 グローバルハンドラ

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。

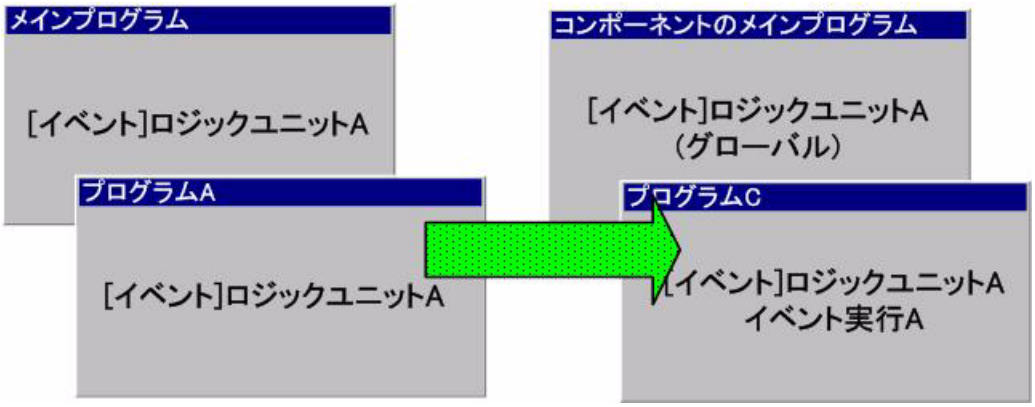


図 6-3 グローバルハンドラ

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。

## コンポーネントは、ホストプロジェクトに定義されたイベントをどのように実行するのでしょうか？

コンポーネントが、ホストプロジェクトに定義されるイベントを使用しなければならない場合があります。このような場合、公開イベントを使用することで可能になります。これらのイベントは、公開名を指定することで実行されます。公開名がタスクツリー内に見つからない場合は、イベントは実行されません。

### 公開イベントを定義するには？

公開イベントはメインプログラム内でのみ定義できます。イベントが定義されているホストプロジェクトでコンポーネントとして公開されていない場合、公開名を定義しただけではイベントは利用できません。イベントは、公開設定を指定する必要があります。

図 6-4 は、イベントを公開設定する例を示しています。


図 6-4 イベントを公開設定する

### 公開イベント使用するには？

公開イベントを使用する場合、[イベントタイプ] は「公開イベント」を選択します。イベント欄でズームすると、[式] エディタが開きます。ここで、イベントの公開名として評価される式を定義します。

図 6-5 は、公開イベントを指定する例を示しています。


図 6-5 公開イベントを使用する

**重要：**

イベントが定義されたプロジェクト内では、公開イベントは通常のイベントとして処理されます。

**覚えておいてください：**

[外部] チェックボックスがチェックされていない場合や、公開名が定義されていない場合、イベントはタスクツリー内には見つかりません。

# 第 7 章 ネストされた Magic コンポーネント

*ネストされ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ことは、簡単に想像できると思います。この章ではネストされたコンポーネントの扱い方について説明します。

## ネストされ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、データソース、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。

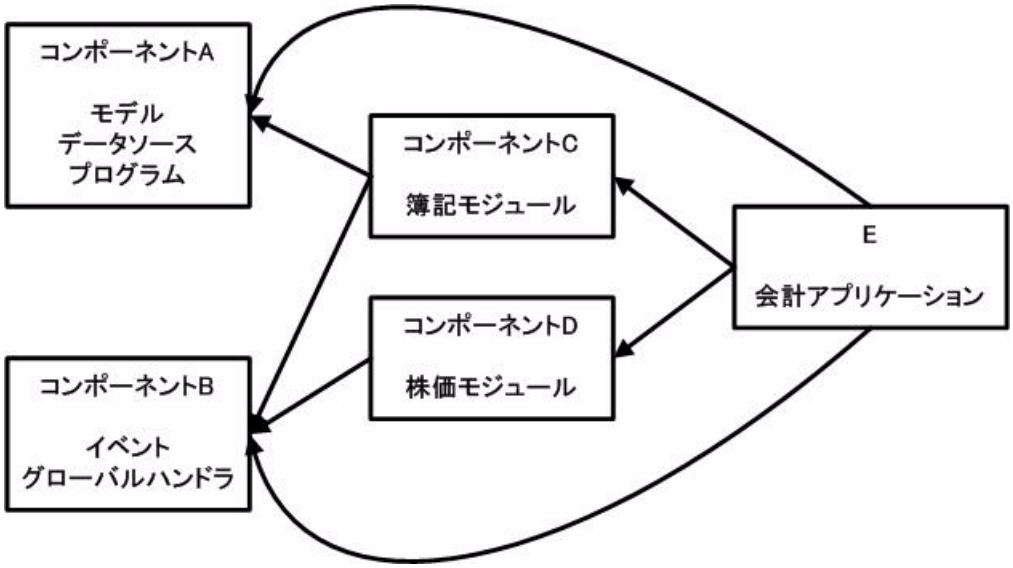


図 7-1 ネストされたコンポーネントシステムの例

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、データソース、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込まれたか記憶しているため、インスタンス毎に再読込みが行なわれることはありません。

**参考：**

Magic xpa 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:¥xpa¥comp.ecf」として読込まれ、別のコンテナに「¥¥myserver¥xpa¥comp.ecf」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 xpa 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：**

ネストされ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：**

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。