Microsoft Corporation

Windows 7 タッチ アプリケーション 開発ガイダンス

Windows 7 での Windows タッチ アプリケーション開発の基本と推奨事項

本ドキュメントは Windows 7 Touch Application Development Guidance の日本語訳であり、掲載している Web サイトやプログラムを日本語で提供していない場合があります。

2011 年 4 月 8 日

要約

このドキュメントでは、Windows® 7 での Windows タッチ機能への投資について説明し、タッチ対応 PC で Windows を最大限に活用したいソフトウェア開発者向けのガイダンスを示します。

この情報は、Windows 7 オペレーティング システムに適用されます。
ここで説明されているリファレンスやリソースは、このドキュメントの最後に掲載されています。

このドキュメントの最新バージョンは、下記の Web サイトで管理されています。
http://code.msdn.microsoft.com/wintouchguide

目次

はじめに

Windows® 7 では、PC をより自然に、楽しく、直感的に使える Windows タッチが導入されています。マルチタッチ テクノロジをサポートする Windows 7 では、ハードウェア/ソフトウェア開発の可能性が大きく広がっています。PC 関連の業界ではマルチタッチの採用が急速に進んでおり、ノート PC、オールインワン デバイス、タッチ対応モニター、スレート PC などでさまざまなタッチ機能が実装されています。

Windows 7 のマルチタッチ アプリケーションに求められるパフォーマンス、セキュリティ、UI 設計、およびウィンドウ管理は他の Windows アプリケーションと同じですが、タッチ対応の Windows 7 PC で優れたユーザー エクスペリエンスを実現するには、ユーザーの入力と対話操作に重点を置くことが重要です。

Windows タッチの一部の機能は、Windows 7 のコアに組み込まれています。開発者が明示的にサポートを有効にしなくても、ユーザーはこの機能を使用することができます。ただし、Windows タッチを最大限に活用するには、開発者は Windows タッチ API を使用する必要があります。

このドキュメントは、タッチ対応 PC で Windows を最大限に活用する方法についてのソフトウェア開発者向けのガイダンスです。本ドキュメントで扱うトピックは以下のとおりです。

- **設計原則:** アプリケーションの設計に関する推奨事項
- **Windows 7 タッチの API:** Windows 7 の各種 API (タッチ、ジェスチャ、操作)
- **開発者プラットフォームの選択:** Windows タッチ対応の Windows 7 開発者プラットフォーム
- **ソフトウェア品質:** Windows 7 クライアント ソフトウェア ロゴ要件
- **Windows タッチに関する参考資料:** 開発者向けのその他のリソース

本ガイダンスと一緒に『スレート PC に関する Windows 7 エンジニアリング ガイダンス』を参照することを強くお勧めします。このドキュメントには、ソフトウェア アプリケーションとタッチ対応 PC ハードウェアに共通のトピックについて説明されています。

タッチ アプリケーションの設計に関する推奨事項

タッチ操作は、操作しやすくシンプルであると同時に、誤操作に対して寛容で即応性に優れていなければなりません。あらゆるユーザー エクスペリエンスと同様、タッチ エクスペリエンスは見た目の美しさも重要ですが、美しさと実用性を兼ね備えていなければなりません。このような設計上の課題は、以下に示すタッチ操作の最適化原則に従うことで対応できます。

信頼性と効率の向上

操作が簡単だと、ユーザーはミスやエラーの発生を恐れることなく、新しい方法をもっと試してみたいという気になります。ユーザー インターフェイス (UI) がシンプルだと、ユーザーは厄介なタスクや過度に複雑なデザインに煩わされずに済み、新たな使用方法の発見や学習に専念できます。タッチ操作のサポートは、既存のユーザー エクスペリエンスに追加される場合がほとんどで、その UI には、スクロール バーなど、タッチ操作に最適化されていない古い要素が含まれています。最悪の場合には、スクロール バーがタッチ操作に対応していないこともあります。スクロール バーをなくして、ユーザーがパンを使用したくなるようにすべきでしょう。

UI が寛容であれば、操作が正確でなくても、簡単に作業を進めることができます。従来のマウスとキーボードを使った操作方法では当たり前だったサイズの小さいアイテムも、タッチだと操作が難しくなります。小さいカーソルとは異なり、指ではアイテムをうまく捉えられません。タッチでの操作があまり困難だと、ユーザーは不満を感じてしまいます。

タッチ操作には即座に反応が求められるため、タッチ環境では即応性がきわめて重要になります。ユーザーが画面に触れて何らかの操作を実行したら、直ちに視覚的な反応を示さなければなりません。フィードバックがなければ、ユーザーは対象アイテムを正しく捉えることができなかったと思い、反応が得られるまでタッチ操作を繰り返すかもしれません。

人間の指はマウス ポインターよりもはるかに大きいため、コントロールの選択によってユーザーの操作性に大きな違いが生じます。たとえば、スピン ボタンやツリー コントロールはタッチ操作を想定して作成されたものではないため、大量のデータを扱うコントロールの使用に関しては十分に検討するようにしてください。

直接操作

直接操作は、タッチによって画面上のオブジェクトを直接的に操作する場合の操作です。ユーザーは、指を使ってユーザー インターフェイスを制御し、コンテンツの移動 (パン)、サイズの調整 (縮小/拡大)、コンテンツの回転などを行います。画面上に表示されるコンテンツを、ちょうど物体を拾い上げて動かすように、タッチで簡単に操作したいというユーザーの期待に応えるため、視覚的なインターフェイスは正確で信頼性のある応答を即座にする必要があります。これは簡単なことように思われるかもしれませんが、直接操作を実現するには、スムーズなアニメーションや物理的な表現などの課題や、手や指が画面を覆う部分が大きくなるときに視覚的にどう対処するかといった課題をクリアしなければなりません。

ジェスチャ

ジェスチャは、直接操作とは別のタッチ操作です。通常、ジェスチャはキーボード ショートカットのように、よく行う操作にマップされています。ジェスチャはコンテンツに対する二次的な操作や機能の実行に適しています。Windows 7 のタッチは、常に一貫した操作性を実現するよう設計されているため、各種の Windows アプリケーションで同じジェスチャを使用する必要があります。頻繁に使用するジェスチャが想定外の動作をすると、ユーザーは混乱し、不満を感じてしまいます。

Windows 7 の設計原則

Windows の強みはプラットフォームとしてのオープン性であり、その成功の基盤は Windows 上で動作するソフトウェアの豊富さと多様性にあると、マイクロソフトは考えています。ただし一方で、PC の使用は、理由はさまざまですが、複雑で一貫性が欠けていると思われていることも、各種のフィードバック チャネルを通じて認識しています。Windows 7 の設計原則に関するページで、Windows 7 におけるユーザー モデルおよびユーザー エクスペリエンスの設計アプローチを説明しています。

このセクションでは、スレート PC に適用できる主な Windows 7 の設計原則について説明します。特にフルスクリーン アプリケーション、およびフルスクリーン アプリケーションと Windows 間の対話の設計に関する考慮事項について重点的に説明します。

基本を押さえて信頼を高める

良いアプリケーションとは、Windows が得意な起動、切り替え、構成設定といった処理や、Windows の汎用性の高さを活かして、革新的で、目的が明確な、緻密に設計されたユーザー エクスペリエンスを提供するアプリケーションを指します。Windows UI は、アプリケーションの実行とオペレーティング システムの管理に必要な幅広い種類のオブジェクトへのアクセスをユーザーに提供します。「Shell Developer's Guide (英語)」は、シェルの動作と、アプリケーションでシェルの API を使用する方法について概念的に説明した資料です。シェル リファレンスのセクションでは、各種シェル API を構成するプログラミング要素について説明しています。「Shell SDK Samples (英語)」は、シェルのサンプルのリンク集です。サンプルの多くは、MSDN コード ギャラリーからダウンロードできます。サンプルはすべて Windows SDK に同梱されています。

遠隔測定調査によれば、ユーザーは PC の使用時間の 19% を Windows シェルの使用に費やしています。これには、[スタート] メニュー、ファイルの参照、コントロール パネルが含まれます。シェル アプリケーションでシェル機能の一部を複製しても、ユーザー エクスペリエンスの強化にはつながりません。一般に、ユーザーは Windows で多くのタスクを実行しており、シェル アプリケーションと Windows を明確に区別するのは困難です。シェル アプリケーションで Windows 機能の一部を複製する方法では、新しい概念の導入によって使用方法が複雑になり、かえってユーザーのフラストレーションを高める結果になってしまいがちです。

質の高い Windows 用アプリケーションを構築することが、スレート PC の魅力を高める最善の方法です。

細かい点が重要: 良い面と悪い面

タッチ操作が主な作業で必須となる場合や頻繁に使用される場合は、細かい点がユーザー エクスペリエンスに大きく影響します。そうした細かい要素の品質を高めることで、ユーザーを満足させる操作性を実現できます。Windows 7 には、開発者がタッチ アプリケーションを差別化するのに役立つたくさんの新機能が用意されています。

Tablet PC 入力パネルには複数のモードがあります。また Tablet PC 入力パネル API によって、起動設定や配置設定のカスタマイズが可能です。以下に例を示します。

1. 入力パネルは、入力パネル アイコン (テキストボックスの隣にある) から起動されると、インプレース モードで実行されます。タッチを使用してインプレース モードで起動すると、入力パネルは、テキスト ボックスに重ならないように画面の下部または上部に表示されます。入力パネルは自由に移動することができ、フォーカスが他のアイテムに移ると自動的に終了します。
   - Tablet PC 入力パネル API を使用して、インプレース モード時の入力パネルをカスタマイズすることが可能で、開発者はタッチ キーボードの起動と配置場所を制御できます。開発者はこの API を使って操作性を最適化できます。たとえば、タッチ操作によってテキスト ボックスにフォーカスを移動したときに、入力パネル アイコンは表示せず、タッチ キーボードを自動的に起動するといったことが可能になります。これは、スレート PC ではとても役に立ちます。また、開発者はタッチ キーボードの表示位置を指定することもできます。
2. タスク バーまたは入力パネル タブから入力パネルを起動すると、フローティング モードで表示されます。このモードでは、入力パネルはユーザーが閉じるまで開いた状態になります。ユーザーは入力パネルを完全に制御できます。入力パネルの既定のドッキング設定はフローティング モードで、タップしたテキスト ボックスの場所に応じてタッチ キーボードが画面上部または下部に表示されます。入力パネルの垂直方向の表示位置は、入力パネル タブの位置によって決まります。ユーザーは入力パネル タブを上下にドラッグして、任意の場所に移動することができます。入力パネル タブを画面の下部に移動すると、入力パネルは画面の下部に表示されます。また、最も重要な点として、ユーザーはアプリケーションを切り替えたり、ディスプレイの向きを変えたりしている最中でも、タッチ キーボードの表示位置を簡単に移動できます。
3. サードパーティ キーボードを使用する場合、入力パネルはアプリケーションから無効にすることができますが、Windows ログオンなどシステム レベルでは無効にできない点に留意してください。
4. Windows 7 では、入力パネルは完全にローカライズされています。オンスクリーン キーボードをアプリケーションに追加する場合はその点を考慮に入れてください。
5. フルスクリーンの DirectX® グラフィックスを使用するソフトウェアは入力パネルを起動できません。適宜、アプリケーション内で独自のオンスクリーン キーボードを用意する必要があります。
6. Silverlight® を使用するソフトウェアでは、入力パネルは自動的に起動されません。アプリケーションでこの後に説明するいずれかのプログラムによる方法を使用するか、テキストを入力するドッキングされた入力パネルをユーザーが開く必要があります。
7. Windows Presentation Framework 4.0 (WPF) を使用するソフトウェアでは、入力パネルは自動的に起動されません。アプリケーションでこの後に説明するいずれかのプログラムによる方法を使用するか、テキストを入力するドッキングされた入力パネルをユーザーが開く必要があります。
8. アプリケーションのソフト キーボードの有効化および拡張に関する詳細については、以下の API 関連資料を参照してください。
   - Text Input Panel Interfaces (英語)
   - **ITextInputPanel** Interface (英語)
9. 上記のインターフェイスにアクセスしない開発プラットフォームを使用するアプリケーションでは、Win32 の一連のキャレット追跡 API を使用して、不特定のテキスト ボックス クラスから必要なテキスト入力を Windows に通知することで、テキスト入力パネルを表示させることができます。

外観と操作性を兼備

タッチ操作では、期待を裏切らない魅力的なビジュアル エクスペリエンスを提供できるかどうかがきわめて重要です。いくら見た目が美しくても、タッチに反応しなければ役に立ちません。ユーザーがタッチ操作しやすい UI と、タッチ入力用に最適化されたコンテキスト タスクを提供することが理想的です。Windows 7 の Microsoft タッチ パック (詳細は後述) には、この原則に従ったタッチ アプリケーションおよびゲームのサンプルが含まれており、たとえば Surface Globe アプリケーションではいろいろなタッチ操作を試すことができます。UI はユーザーが次に何をするかを予測しており、ユーザーの直接操作やジェスチャによってすばやく簡単にナビゲーションを実行できます。

また、Windows 7 はセンサーをネイティブにサポートしており、Windows アプリケーション内で環境情報を利用できます。これには位置センサー (GPS デバイスなど)、環境光センサー、および温度計などが含まれます。位置センサーは位置情報サービスの新たな可能性を拓きます。センサーの使用に関する詳細については、「Sensor API」を参照してください。

Windows タッチ API は、パン、ズーム、回転など、豊富な数のジェスチャをサポートしています。ジェスチャは、即時に視覚的なフィードバックを提供し、コンテンツとの自然で直感的な対話を実現します。

Windows 7 のタッチ操作に関するガイドラインでは、タッチ操作向けに最適化されたアプリケーションの開発方法や既存のアプリケーションを最大限に活用する方法についての詳しい情報を提供しています。タッチ操作向けに最適化されたアプリケーションとは、以下のようなタッチ操作用に設計されたアプリケーションを指します。

- 最も頻繁に実行するコマンドがドロップダウン メニューではなく UI またはコンテンツ上に直に表示されており、タスクがタッチ操作向けに最適化されているアプリケーション
- タッチ操作が運動量や摩擦といった現実の物理的特性を備えているアプリケーション
- タスクが誤操作に対して寛容で、タッチやドラッグの操作が不正確でも簡単に修正できるアプリケーション
- テキストを大量に入力したり、正確にアイテムを選択したりせずに済むアプリケーション

画面のサイズが小さい場合の DPI 設定

画面のサイズが小さいと利用可能な画面領域も制限されるため、レンダリング方法や、画面サイズをさまざまに変えた場合にアプリケーションがどのように表示されるかに配慮する必要があります。また、異なる画面の向きにアプリケーションが適切に対応するようにします。また、アプリケーションが高 DPI 設定をサポートしていないというのもよくある問題の 1 つです。アプリケーションがもともとタッチ操作をサポートしない設計である場合や、Windows で DPI の設定変更が可能なことを設計担当者が認識していなかった場合にはこのような問題が生じる可能性があります。画面の DPI を大きくすれば、有効なフォント サイズが大きくなりテキストが見やすくなりますが、ディスプレイに表示される有効なピクセル数は少なくなります。『小型 PC 用の DPI 構成』に、Windows 7 では、システム DPI は下記のように 96 DPI を 100% とした割合で表されると記述されています。

- 96 DPI = 100%
- 120 DPI = 125%
- 144 DPI = 150%

ディスプレイの有効解像度では、物理的な解像度とシステム DPI の設定値の両方が考慮されます。DPI 設定値が上がると、有効解像度は低くなります。これを計算する数式は次のとおりです。

有効解像度 = 物理的な解像度 / DPI

Windows 7 は、ディスプレイの解像度を 1024 x 768 として最適化されており、最小解像度は 800 x 600 です。小型で高密度の画面を使用する場合の課題として、そのような画面で最適な有効解像度に構成すると、文字のサイズが非常に小さくなってしまうことがあります。そのような場合は、有効解像度が Windows の最小要件を下回らない範囲で調整することをお勧めします。

組み込みの Windows コントロールと API を使えば、少ない労力でガイドラインに準拠したアプリケーションを作成できます。Windows のコントロールは拡大縮小やアクセシビリティに対応しており、タッチ操作用の設計が施されています。たとえば Windows リボン フレームワークは、従来の階層化されたメニュー、ツール バー、タスク ウィンドウに代わる最新の UI を提供する、リッチなコマンド表示システムです。アクセス性にとても優れた UI を提供するために、リボン フレームワークは Microsoft Active Accessibility を実装しています。関連する Microsoft Active Accessibility のプロパティには、有効で役に立つ情報がフレームワークによって自動的に設定されるため、あらゆるユーザーに対応できるエクスペリエンスを少ない負担で開発できます。また、リボン フレームワークは Windows の機能の 1 つであるため、Windows でサポートされるすべての言語にローカライズされています。ただし、独自のアプリケーション リソースのローカライズは開発者が担当します。

速度も重要: パフォーマンス向上のポイント

アプリケーションのパフォーマンスが低いと、システム全体について悪印象を与えてしまいかねません。アプリケーションを実行する特定のハードウェア構成およびソフトウェア構成でアプリケーションのテストを徹底的に行うことが重要です。システムの基本要件に加えて、アプリケーションのパフォーマンスに関する以下の推奨事項に従ってください。

- 起動時のクリティカル パス上にあるアプリケーションについてはマネージ コードの使用を避ける。
- シャットダウン通知 (WM_QUERYENDSESSION メッセージおよび WM_ENDSESSION メッセージ) に対してすべてのアプリケーションが直ちに応答するようにする。
- シャットダウン通知を受けたら、CPU、ディスク、およびネットワークのアクティビティを最小化し、サービスおよびアプリケーションのシャットダウン処理における遅延を低減する。
- 中断通知 (WM_POWERBROADCAST メッセージ) の処理の遅延を回避する。
- 復帰イベントにすばやく応答して、復帰直後の CPU、ディスク、およびネットワークの使用を最低限に抑える。
- 起動直後のアプリケーションによるリソースの消費を抑える。
- アプリケーション間でのコード共有というメリットが得られるように、テクノロジ プラットフォームの選定に一貫性を持たせる。たとえば、.NET コードは、複数のアプリケーションで使用した場合、1 回しか読み込まれない。
- 起動時に毎回 RunOnce キーからアプリケーションを起動しない。

エクスペリエンスのライフサイクル全体を評価する

Windows は、一貫性のある 1 つの設計で、構成設定、プログラムの起動、プログラムの切り替え、ファイルの参照と管理、デバイスの管理、ログオン、アカウントやタスクの管理など、ユーザーが PC 上で行うさまざまなタスクに対応します。ユーザーは標準でカスタマイズ、個人設定、多言語などに対応していることを期待しています。これらの重要な作業をユーザーが実行できなくなってしまわないように、アプリケーションを直ちにフルスクリーン状態にしないことが重要です。

Windows のこうした機能領域の設計では、全体の操作性を深く検討した上で設計を行っています。たとえば、Windows やドライバーの重要な更新プログラムの提供が開始されると、アクション センターの通知領域でその旨がユーザーに通知されます。その他の重要情報 (電源状態、ネットワーク状態、現在の音量など) もこれと同じ方法でユーザーに通知されます。これらの通知および通知を表示する UI では、よく行う重要なタスク

が数回以内のクリックで実行できるようになっています。Windows デスクトップの UI は、このようなリッチで連携したシナリオをサポートするように最適化されています。

ユーザーにフルスクリーン アプリケーションを表示すると、こうしたユーザー シナリオの多くが中断されてしまいます。ユーザーが他のタスクを実行するには Windows に戻る必要があるということを意識しておくことが重要です。フルスクリーン アプリケーションに [閉じる] ボタンなどの標準的なコントロールが用意されていないと、ユーザーを混乱させ、不満の元となってしまいます。たとえば、Windows 7 には Windows Media Center が搭載されており、タッチ操作で各種メディアを利用できます。Windows Media Center は、標準的なアプリケーションのように起動したり切り替えたりすることができるだけでなく、一時的にフルスクリーン モードで動作するように構成することも可能です。Windows Media Center は、フル スクリーン モードで操作できますが、ワイヤレス ネットワークへの接続や更新プログラムのインストールなど、OS に関するタスクを実行する場合は、簡単に Windows デスクトップに戻ることができます。フル スクリーン モードでは実行できないタスクがあるため、Windows デスクトップにユーザーがいつでも戻れるようにすることが大切です。

Windows 7 タッチの API の概要

タッチやジェスチャのサポートは、新しいアプリケーションにも既存のアプリケーションにも簡単に追加することができます。Windows 7 では、ほとんどの種類の PC デジタイザーに対応する各種の API によって、シングルタッチ入力とマルチタッチ入力がサポートされます。

既存のアプリケーション用に、Windows 7 の既定のジェスチャ ハンドラーによって、いくつかのジェスチャが以前のバージョンの Windows で使用されていた Windows メッセージにマップされます。次の表は、ジェスチャをレガシ メッセージにマップする方法についてまとめたものです。

ジェスチャ	説明	生成されるメッセージ
パン	パン ジェスチャは、スクロール ホイールの使用にマップされます。	WM_VSCROLL WM_HSCROLL
プレス アンド ホールド	プレス アンド ホールド ジェスチャは、マウスの右クリックにマップされます。プレス アンド タップ ジェスチャも同様です。	WM_RBUTTONDOWN WM_RBUTTONUP
ズーム	ズーム ジェスチャは、Ctrl キーを押しながらマウス ホイールを回転させてスクロールする操作と同様のメッセージをトリガーします。	*lParam* で MK_CONTROL が設定された WM_MOUSEWHEEL
タップ	タップ イベントは、マウスの左クリックにマップされます。WM_LBUTTONUP が WM_LBUTTONDOWN の直後に生成されます。	WM_LBUTTONDOWN WM_LBUTTONUP
選択/ドラッグ	ドラッグは、マウスの移動にマップされます。選択可能な UI 要素で最初のタップが発生すると、選択が発生します。	WM_LBUTTONDOWN WM_MOUSEMOVE WM_LBUTTONUP

タッチによる対話操作を行うアプリケーションを設計するときは、レガシ サポートを利用するのではなく、いずれかの Windows タッチ API を使用するようにしてください。Windows 7 には、アプリケーションでのタッチによる対話操作を可能にする次の API セットが用意されています。

- Windows タッチ メッセージ
- Windows タッチ ジェスチャ
- Windows 操作インターフェイス
- Windows 慣性プロセッサ

タッチのサポートを実装する際に使用する API セットは、主にアプリケーションにおけるユーザー シナリオによって決まります。たとえば、閲覧が目的のアプリケーションでは、ページ操作用の Windows タッチ ジェスチャ (WM_GESTURE) を使用すると最も効果的です。一方、対話型のゲームでは、Windows タッチ メッセージ (WM_TOUCH) を使用したタッチの追跡が必要になる場合もあります。また、操作インターフェイスと慣性プロセッサはよく似たインターフェイス実装で、どちらもより細かいレベルで複数の接触点をサポートできます。これらは、よりタッチ操作向けに最適化されたシナリオで必要になり、オブジェクトを処理するための直接操作機能と組み合わされた、ジェスチャのスーパーセットを提供します。

Windows 7 の各種の API を使用すると、柔軟にタッチ機能を実装することができます。必ずしも相互に排他的である必要はありませんが、十分な検討を行い、タッチ アプリケーションで求められるシナリオに基づく適切な API を選択するようにしてください。

Windows タッチ メッセージ

Windows タッチ メッセージは、1 つ以上の接触点 (指やペンなど) がタッチセンサー式デジタイザーの表面に触れたことを示します。WM_TOUCH メッセージは、ユーザー入力が生成されるとフォーカスがあるウィンドウに通知が送られる点で、他の入力メッセージに非常に似ています。他のマウス メッセージやキーボード メッセージと 1 つ大きく異なる点は、WM_TOUCH では、接触、解放、移動などの状態が 1 つのメッセージにまとめられることです。

実際のメッセージ (WM_TOUCH) の *lParam* には、**GetTouchInputInfo** の呼び出しで使用できるハンドルが格納されます。これを使用して、このメッセージに関連付けられた接触点に関する詳細情報を取得します。

Windows タッチ メッセージを操作するときは、一般に次の手順を実行します。

- 入力デジタイザーの機能をテストする
- Windows タッチ メッセージを受け取るように登録する
- メッセージを処理する

ハードウェアの機能を確認する

タッチ メッセージを有効にして受け取る前に、PC がタッチに対応しているかどうかを確認する必要があります。

入力デジタイザーの機能を照会するには、**GetSystemMetrics** 関数を使用して、SM_DIGITIZER の *nIndex* パラメーター値を渡します。**GetSystemMetrics** は、デバイスの準備ができているかどうか、デバイスがペンまたはタッチをサポートしているかどうか、入力デバイスが統合デバイスと外付けデバイスのどちらであるか、およびデバイスが複数入力 (Windows タッチ) をサポートしているかどうかを示すビット フィールドを返します。

次の表に、入力デジタイザーのタッチ機能のテスト用に windows.h で定義されている定数を示します。

名前	値	説明
TABLET_CONFIG_NONE	0x00000000	入力デジタイザーにタッチ機能がありません。
NID_INTEGRATED_TOUCH	0x00000001	統合型のタッチ デジタイザーが入力に使用されています。
NID_EXTERNAL_TOUCH	0x00000002	外付けのタッチ デジタイザーが入力に使用されています。
NID_INTEGRATED_PEN	0x00000004	統合型のペン デジタイザーが入力に使用されています。
NID_EXTERNAL_PEN	0x00000008	外付けのペン デジタイザーが入力に使用されています。
NID_MULTI_INPUT	0x00000040	複数入力がサポートされた入力デジタイザーが入力に使用されています。
NID_READY	0x00000080	入力デジタイザーで入力の準備ができています。この値が設定されていない場合、タブレット サービスが停止されているか、デジタイザーがサポートされていないか、デジタイザー ドライバーがインストールされていない可能性があります。

タッチ メッセージを受け取るように登録する

Windows タッチ入力メッセージを受け取るように登録するには、アプリケーションで **RegisterTouchWindow** 関数を呼び出す必要があります。この関数で、アプリケーションの **HWND** とフラグ パラメーターを指定します。このパラメーターを使用すると、タッチ入力の報告方法をカスタマイズする 2 つの重要なフラグを指定できます。次の表にそれらのフラグを示します。

フラグ	説明
TWF_FINETOUCH	*hWnd* パラメーターで指定したウィンドウで、一体化されていないタッチ入力を優先することを指定します。このフラグを指定しなかった場合、アプリケーション ウィンドウは、すべての接触点を 1 つのメッセージで受け取ります。
TWF_WANTPALM	このフラグを設定すると、手のひら接触の拒否が無効になり、WM_TOUCH メッセージを取得する際の遅延を減らすことができます。これは、ユーザーがアプリケーションに触れたときにできるだけ速く応答する必要がある場合に役立ちます。 既定では、手のひら接触の検出が有効であり、一部の WM_TOUCH メッセージがアプリケーションに送信されなくなります。これは、手のひら接触による WM_TOUCH メッセージを受け取らないようにする場合に役立ちます。

WM_TOUCH メッセージを処理する

アプリケーションの設計に応じて、さまざまな方法で Windows メッセージを処理できます。たとえば、**WndProc** 関数で処理したり、MFC やマネージ アプリケーションのハンドラーで処理したりできます。アプリケーションでのメッセージの処理方法に関係なく、WM_TOUCH の *lParam* は **GetTouchInputInfo** 関数に渡されます。**GetTouchInputInfo** の出力パラメーターは **TOUCHINPUT** 構造体です。**TOUCHINPUT** には、接触点に関する詳細情報が格納されます。これには、X 座標と Y 座標、入力デバイスのソース、ID、フラグ (接触、解放、移動を示す)、タイムスタンプ、およびいくつかのオプション フィールドが含まれます。

WM_TOUCH メッセージと Windows のその他の入力メッセージの大きな違いの 1 つは、**GetTouchInputInfo** の *dwFlags* パラメーターです。WM_TOUCH メッセージでは、接触、解放、移動などの関連するアクションのメッセージを個別に受け取るのではなく、*dwFlags* パラメーターでこの動作を示します。この違いがわかるように、次の表に **TOUCHINPUT** 構造体の **dwFlags** メンバーのフラグを示します。

フラグ	値	説明
TOUCHEVENTF_MOVE	0x0001	移動が発生しました。TOUCHEVENTF_DOWN と組み合わせることはできません。
TOUCHEVENTF_DOWN	0x0002	対応する接触点が新しい接触によって確立されました。TOUCHEVENTF_MOVE または TOUCHEVENTF_UP と組み合わせることはできません。
TOUCHEVENTF_UP	0x0004	接触点が解放されました。
TOUCHEVENTF_INRANGE	0x0008	接触点が範囲内にあります。このフラグは、互換性のあるハードウェアでタッチ ホバーをサポートするために使用されます。ホバーのサポートが不要なアプリケーションでは、このフラグは無視してかまいません。
TOUCHEVENTF_PRIMARY	0x0010	この **TOUCHINPUT** 構造体が第 1 接触点に対応することを示します。
TOUCHEVENTF_NOCOALESCE	0x0020	この入力は、**GetTouchInputInfo** を使用して受け取られるときに一体化されませんでした。
TOUCHEVENTF_PALM	0x0080	このタッチ イベントは、ユーザーの手のひらによるものです。

アプリケーションでは、**GetTouchInputInfo** から返される **TOUCHINPUT** 構造体に含まれる接触点を反復処理して、視覚的な要素をレンダリング、格納、操作できます。

WM_TOUCH メッセージは、ウィンドウの HTTRANSPARENT 領域を無視します。ウィンドウが WM_NCHITTEST メッセージへの応答として HTTRANSPARENT を返す場合、マウス メッセージは親に送られ、WM_TOUCH メッセージはウィンドウに直接送られます。

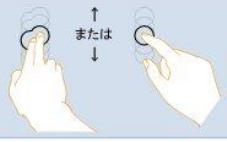
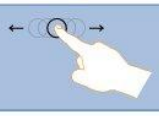
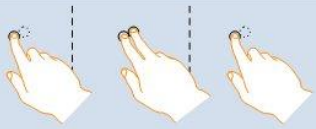
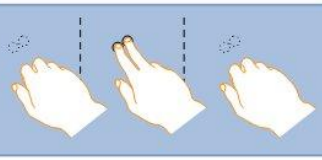
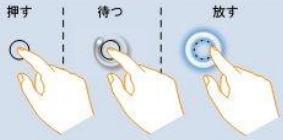
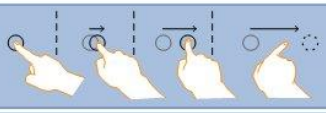
そのほかに注目すべき機能として、TOUCH_COORD_TO_PIXEL というマクロが用意されています。このマクロは、アプリケーションで入力をディスプレイに正しくレンダリングできるように、タッチ座標 (現在はセンチピクセル) をピクセルに変換するために使用されます。タッチ座標はピクセルよりも細かいので、アプリケーション開発者は、グラフィック デザインなどに特化したアプリケーションに対してサブピクセル単位の細かさを活用できます。

最後に、タッチ入力が不要になった場合やアプリケーションを終了するときは、アプリケーションで **CloseTouchInputHandle** および **UnregisterTouchWindow** を呼び出す必要があります。

Windows タッチ ジェスチャ

Windows ジェスチャのメッセージを使用すると、ユーザーが作成したマルチタッチ アクションを受け取り、単一の操作タスクを実行できます。Windows 7 のジェスチャは、未加工のタッチ メッセージよりも高度なサポートをアプリケーションに提供します。細かな回転、強化されたズームのサポート、または 1 本指によるパンに対するサポートだけが必要な場合、そのようなアプリケーションの Windows タッチの開発にはジェスチャが最も適しています。このより高度なメッセージング機能により、ユーザーは、一般的なジェスチャに対してすべての Windows アプリケーションで一貫した応答が得られます。

Windows では、「Windows タッチ ジェスチャの概要」の表に示されているように、さまざまなジェスチャがサポートされています。この表を次に示します。

ジェスチャ	WINDOWS の使用	ジェスチャ アクション	アクション (○ = 指で触れる | ◌ = 指を放す)	単一の接触	複数の接触
タップ/ダブルタップ	クリック/ダブルクリック			✦	✦
慣性を伴うパン	スクロール	1 本指または 2 本指で上下にドラッグする			✦
選択/ドラッグ (1 本指で左右の方向)	マウスでドラッグ/選択	1 本指で左右にドラッグする		✦	✦
プレス アンド タップ	右クリック	1 本の指で対象を押したままもう 1 本の指でタップする			✦
ズーム	ズーム (既定では Ctrl キーを押しながらスクロール ホイールを操作)	2 本の指の間隔を広げる/狭める			✦
回転	システムの既定の操作なし、アプリケーションで対応 (WM_GESTURE API を使用)	2 本の指を反対の方向に動かす - または - 1 本の指を中心にしてもう 1 本の指を回転させる			✦
2 本指のタップ	なし - アプリケーションでジェスチャ API を使用して独自に公開	2 本指で同時にタップする (2 本の指の中間点がタップ位置になる)			✦
プレス アンド ホールド	右クリック	押したまま待ち、青い円のアニメーションが表示されたら放す		✦	✦
フリック	既定: 上に移動、下に移動、戻る、進む	目的の方向にすばやくドラッグする		✦	✦

WM_GESTURE メッセージを処理する

Windows 7 のジェスチャは、WM_GESTURE という 1 つのメッセージで処理されます。既定では、すべてのアプリケーションがジェスチャを受け取ります。**RegisterTouchWindow** 関数を使用して Windows タッチ入力メッセージを受け取るように登録しない限り、ジェスチャの通知 (WM_GESTURE メッセージ) が作成されて、そのアプリケーション ウィンドウに送られます。Windows タッチおよびジェスチャのメッセージは最長方式で処理されます。つまり、アプリケーション ウィンドウでタッチが行われるかジェスチャが始まった後、ジェスチャが完了するか最初のタッチが完了するまで、すべてのメッセージがそのアプリケーションに送られます。

WM_GESTURE メッセージの *lParam* には、**GetGestureInfo** の呼び出しで使用できるハンドルが格納されます。これを使用して、ジェスチャ コマンドとジェスチャ固有の引数値を特定した詳細情報を取得します。**GetGestureInfo** の出力パラメーターには、**GESTUREINFO** 構造体が格納されます。

GESTUREINFO 構造体には、ジェスチャに関する詳細情報が格納されます。これには、ジェスチャ ID、対象のウィンドウの **HWND**、ジェスチャに関連付けられた座標が格納された **POINTS** 構造体、およびその他の引数が含まれます (「**GESTUREINFO 構造体**」を参照)。この構造体の *dwFlags* パラメーターには、次の 3 つのいずれかが格納されます。

名前	値	説明
GF_BEGIN	0x00000001	ジェスチャが開始されています。
GF_INERTIA	0x00000002	ジェスチャで慣性がトリガーされました。
GF_END	0x00000004	ジェスチャが完了しました。

アプリケーションでは、GF_BEGIN フラグと GF_END フラグを使用して、ジェスチャの開始と終了の情報 (位置、回転、ズームなど) を追跡できます。ジェスチャが開始された後、アプリケーションは、ユーザーがそのアクションを完了するまで後続のジェスチャ メッセージを受け取ります。GF_END フラグが設定されたメッセージを受け取るまで、各ジェスチャ メッセージ (GID_ZOOM、GID_PAN、GID_ROTATE、GID_TWOFINGERTAP、および GID_PRESSANDTAP) を前のメッセージからの差分を示す位置情報と共に受け取ります。GF_INERTIA フラグは、GID_PAN ジェスチャが慣性に対応していることを示します。

WM_GESTURE を処理する場合、アプリケーションはジェスチャが処理されたことを示す 0 を返します。ジェスチャの処理が完了した後、アプリケーションは **CloseGestureInfoHandle** を呼び出して、WM_GESTURE の *lParam* から取得したジェスチャ情報のハンドルを解放する必要があります。

このメッセージを処理しない場合、アプリケーションは **DefWindowProc** を呼び出す必要があります。呼び出さないと、タッチ入力ハンドルが閉じられず、関連付けられたプロセス メモリが解放されないため、アプリケーションでメモリ リークが発生します。**DefWindowProc** を使用して WM_GESTURE メッセージを転送することで、レガシ アプリケーションにも対応できます。レガシ サポートのために、Windows では、WM_GESTURE メッセージがバブルアップされるとそれらが解釈され、ジェスチャに対応する適切なメッセージの SEND または POST が実行されます。レガシ サポートの詳細については、MSDN の「Windows タッチ ジェスチャの概要」を参照してください。

パンの境界域フィードバック

Windows タッチを対象とするアプリケーションを作成する場合、パン機能の基本的なサポートが自動的に提供されますが、WM_GESTURE メッセージを使用すれば、1 本指によるパンに対する拡張サポートを提供することもできます。これにより、たとえば、GF_INERTIA フラグに対応し、パンの境界に達したときに視覚的なユーザー フィードバック ('バウンス' など) を追加することができます。このカスタム パン サポートの詳細については、MSDN ライブラリの「1 本指によるパンのエクスペリエンスの向上」を参照してください。このトピックでは、アプリケーションでフリックを無効にする方法、カスタム パンをサポートする方法、およびパンの境界域フィードバックで 'バウンス' を有効にする方法が説明されています。

Windows には、パンのフィードバックをサポートする関数がいくつか用意されています。

関数	説明
BeginPanningFeedback	この関数は、境界域フィードバックを使用する前にユーザーによって呼び出されます。
EndPanningFeedback	この関数は、境界域フィードバックの完了時に呼び出されます。オプションでアニメーションをトリガーします。
UpdatePanningFeedback	この関数は、境界域フィードバック中に呼び出されます。オプションでアニメーションをトリガーします。

ウィンドウにカスタム パン フィードバックを追加する前に、*hWnd* で MICROSOFT_TABLETPENSERVICE_PROPERTY を設定して、フリックを無効にする (TABLET_DISABLE_FLICKS フラグを使用) 必要があります。この呼び出しの詳細については、「スクロール バーを使用したパンのレガシ サポート」を参照してください。

Windows 操作インターフェイス

Windows 操作インターフェイスを使用すると、WM_TOUCH や WM_GESTURE よりも細かくタッチ入力を処理できます。アプリケーションで複数のジェスチャによるオブジェクト操作 (平行移動、回転、拡大縮小の組み合わせ) を実行できるようにする場合は、操作 API を使用することをお勧めします。

操作は、実質的には、ジェスチャ全体を示す値が関連付けられたジェスチャです。このサポートを利用するには、アプリケーションで **IManipulationProcessor** インターフェイスを実装する必要があります。この API を処理するアプリケーションの大まかな手順は次のとおりです。

1. Windows タッチ メッセージを受け取るように登録する
2. **_IManipulationEventSink** インターフェイスを実装する
3. COM オブジェクトを作成し、**IManipulationProcessor** インターフェイスと **IInertiaProcessor** インターフェイスを設定する
4. WM_TOUCH メッセージを処理する
5. **TOUCHINPUT** 構造体を適切なプロセッサに渡す
6. **ManipulationCompleted** イベント ハンドラー内で慣性を設定する
7. COM オブジェクトをクリーンアップする

IManipulationProcessor インターフェイスは、WM_TOUCH メッセージを解釈して、一連の接触点に関する平行移動、回転、および拡大縮小の情報を含むイベントを生成します。**IInertiaProcessor** インターフェイスを **IManipulationProcessor** インターフェイスと組み合わせて使用すると、アニメーションを有効にし、オブジェクトの移動中もオブジェクトがユーザーの画面上に表示されるようにすることができます。

以降のセクションでは、**IInertiaProcessor** インターフェイスを **IManipulationProcessor** インターフェイスと一緒に使用する方法の概要について説明します。

タッチ メッセージを受け取るように登録する

WM_TOUCH を使用する場合と同様に、操作 API を使用するには、**RegisterTouchWindow** 関数を使用して Windows タッチ入力メッセージを受け取るように登録する必要があります。

_IManipulationEventSink インターフェイスを実装する

_IManipulationEvents イベント シンクには、**ManipulationStarted**、**ManipulationDelta**、および **ManipulationCompleted** の 3 つの関数が含まれます。これらのコールバック関数は、**IManipulationProcessor** インターフェイスと **IInertiaProcessor** インターフェイスで使用され、**ProcessTime**、**ProcessUpWithTime**、**ProcessDownWithTime**、および **ProcessMoveWithTime** の各関数を呼び出すことによってプロセッサで計算された値を返します。

_IManipulationEvents インターフェイスは次のメソッドを定義します。

メソッド	説明
ManipulationStarted	操作または慣性が開始されるときのイベントを処理します。
ManipulationDelta	操作中のオブジェクトが変更された場合に発生するイベントを処理します。
ManipulationCompleted	操作または慣性が終了されるときのイベントを処理します。

COM オブジェクトを作成し、IManipulationProcessor インターフェイスと IInertiaProcessor インターフェイスを設定する

この API は、**IManipulationProcessor** インターフェイスと **IInertiaProcessor** インターフェイスの実装を提供します。**IManipulationEvents** イベント シンクの実装で作成した COM オブジェクトをインスタンス化して参照する必要があります。

WM_TOUCH メッセージを処理する

WM_TOUCH メッセージから入力データを抽出する必要があります。後でこのデータを適切な操作プロセッサに送って処理します。

TOUCHINPUT 構造体を適切なプロセッサに渡す

GetTouchInputInfo 関数を呼び出して WM_TOUCH メッセージからデータを抽出した後、**TOUCHINPUT** 構造体で設定されている **dwFlag** に応じて、**ProcessUpWithTime**、**ProcessDownWithTime**、**ProcessMoveWithTime** のいずれかの関数を呼び出して、データを操作プロセッサに送ります。

> **注**　複数の操作をサポートする場合に、適切な **IManipulationProcessor** オブジェクトにデータを送るために **TOUCHINPUT** 構造体で定義されている **dwID** を使用する必要があるときは、新しい操作プロセッサを作成する必要があります。

ManipulationCompleted 内で慣性を設定する

慣性を有効にするには、**ManipulationCompleted** メソッドを呼び出した後に、**IManipulationProcessor** オブジェクトで、**IManipulationProcessor** にリンクされた **IInertiaProcessor** オブジェクトの値を設定する必要があります。

IInertiaProcessor インターフェイスを **IManipulationProcessor** インターフェイスと組み合わせて使用すると、アニメーションを有効にし、オブジェクトの移動中もオブジェクトがユーザーの画面上に表示されるようにすることができます。

COM オブジェクトをクリーンアップする

アプリケーションを終了するときに、既存の COM オブジェクトをすべてクリーンアップする必要があります。

まとめ

このセクションで概説した手順の詳しい説明については、「操作と慣性のサンプル」を参照してください。

Windows 慣性プロセッサ

Windows タッチ アプリケーションには、オブジェクトから指を離したときにオブジェクトが唐突に停止するのではなくスムーズに停止するように、簡単な物理法則を組み込むのが一般的です。アプリケーションが他のアプリケーションと同様に動作するように、それらの簡単な物理法則に基づく動作の計算を実行するための慣性 API が用意されています。この API を使用すると、堅牢な物理法則機能を作成する手間も省けます。

慣性は、慣性プロセッサ (プロセッサの機能をカプセル化するクラス) を使用することで有効になります。慣性プロセッサは、オブジェクトの操作の完了時に渡されるイベントを処理し、オブジェクトの軌跡を他のアプリケーションと一貫した方法で処理することで機能します。

操作を使用するアプリケーションに慣性のサポートを簡単に追加できるように、慣性プロセッサは操作プロセッサと密接に結合されています。実際、慣性プロセッサは、操作プロセッサと同じイベントを生成します。唯一の違いは、慣性では、解釈されたメッセージが操作プロセッサではなく慣性プロセッサに渡され、慣性プロセッサがイベントを生成することです。

IInertiaProcessor インターフェイスを **IManipulationProcessor** インターフェイスと組み合わせて使用すると、アニメーションを有効にし、オブジェクトの移動中もオブジェクトがユーザーの画面上に表示されるようにすることができます。

操作インターフェイスと同様、詳しい手順については、「操作と慣性のサンプル」を参照してください。

Windows 7 タッチに対応したその他の開発者プラットフォーム

Windows 7 SDK に含まれるネイティブの Windows タッチ サポートに加え、.NET Framework の Silverlight と Windows Presentation Foundation (WPF) もマルチタッチをサポートしています。Web アプリケーションやリッチ クライアント エクスペリエンスを開発するときは、その目的合わせてこれらのプラットフォームも使用できます。

小型 PC のパフォーマンスに関する考慮事項

どの開発者プラットフォームを使用する場合でも、既に述べたように、タッチ環境では即応性がきわめて重要なことに変わりはありません。タッチ操作には即座に反応が求められます。ユーザーが画面に触れて何らかの操作を実行したら、直ちに視覚的な反応を示さなければなりません。フィードバックがなければ、ユーザー

は対象アイテムを正しく捉えることができなかったと思い、反応が得られるまでタッチ操作を繰り返すかもしれません。

Windows 7 タッチに対応したアプリケーションを構築するときは、対象となるシナリオで求められるハードウェアの機能を考慮する必要があります。たとえば、軽量スレート PC とハイエンド ワークステーションでは、パフォーマンス特性はまったく異なります。アプリケーションだけでなく、対象のデバイスの CPU レベルまで考慮して、最適な開発プラットフォームを選択するようにしてください。

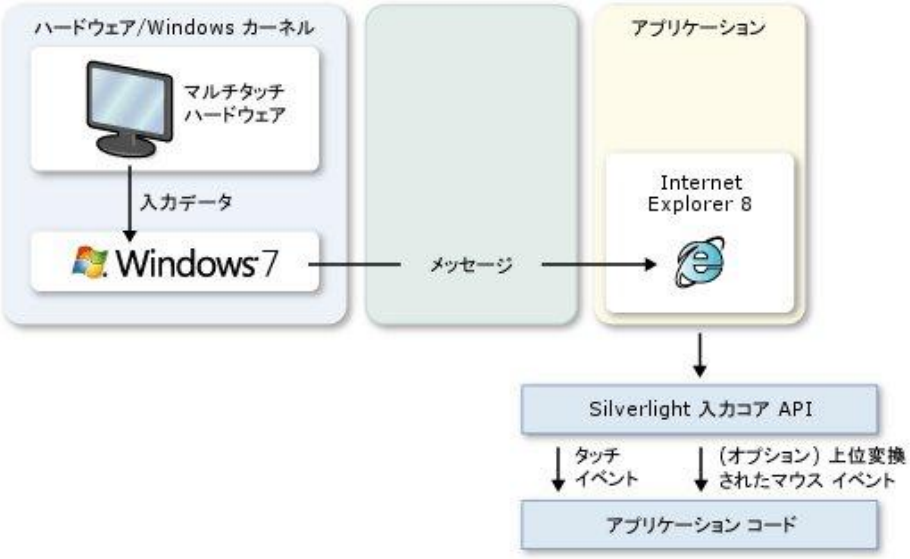
CPU/GPU の仕様が最も低いものに合わせてパフォーマンスを最適化しておけば、どの Windows 7 ハードウェアでもアプリケーションの応答性を確保することができます。MSDN ライブラリには、パフォーマンスに関する問題のトラブルシューティングに役立つ、パフォーマンス チューニングに関するドキュメントやツールがいくつか公開されています。

Silverlight 4

Microsoft Silverlight® では、プラットフォームの WM_TOUCH と同じように、マルチタッチ メッセージを未加工のメッセージのレベルで処理し、未加工のメッセージのレベルでタッチポイントの特性をキャプチャする API にアクセスします。その後、その特性が Silverlight API として公開されます。Silverlight 4 では、ジェスチャなどの高度なメタファーはサポートされません。

ブラウザー ホストとしての Internet Explorer® 8 もマルチタッチ対応です。Internet Explorer 8 は、Internet Explorer 内で実行されるプラグイン (Silverlight など) にプラットフォームのマルチタッチ メッセージを転送し、これによって Silverlight アプリケーションでのマルチタッチ入力の操作が可能となります。

Silverlight のマルチタッチ入力 (Internet Explorer 8 ホストの場合)



マルチタッチは、Windows 7 上で動作する現在のバージョンの Firefox でホストされた Silverlight、および、Windows 7 上で動作するブラウザー外実行アプリケーションでもサポートされます。ただし、Silverlight アプリケーションが全画面表示モードで実行されている場合、マルチタッチ入力はサポートされません。

Silverlight 4 では、通常、レガシ サポートのために未加工のタッチ入力イベントをマウス メッセージに上位変換します。Silverlight ではタッチからジェスチャへの自動的な上位変換は行われないため、その効果を得るためにはプラットフォームの機能を使用するか、WM_GESTURE を処理する必要があります (Silverlight 4 では、WM_GESTURE をサポートしていません)。

ジェスチャのメタファーを使用したマルチタッチを処理する場合は、コードでタッチ イベントを処理し、Silverlight 4 で公開される API を使用して、イベント パラメーターをジェスチャに変換する必要があります。ジェスチャのためにプラットフォーム API を使用するかどうかに関係なく、これらすべてを行う必要があります。これは単純な作業ではありません。

マルチタッチ入力と Silverlight でサポートされている他の入力方法 (マウス、キーボード、スタイラス) の重要な相違点の 1 つは、ハンドラーを特定の入力要素 (UIElement オブジェクト) に追加せずに、マルチタッチ イベントをアプリケーション単位で登録することです。これは、Silverlight 全体がプラットフォームに登録される "アプリケーション" であるという考え方と一致します。

開発およびアーキテクチャについてのその他の情報やプラットフォームの制限など、Silverlight でのマルチタッチのサポートに関する詳細については、MSDN ライブラリの「マルチタッチ入力」を参照してください。

Windows Presentation Foundation (WPF) 4

Windows Presentation Foundation (WPF) 4 には、未加工のタッチ データを報告する単純な機能から、操作や慣性などの高度な機能まで、広範なマルチタッチがサポートされています。このサポートを使用することで、マウスやキーボードなどの他の入力に応答する場合と同様に、タッチが行われたときにイベントを生成する方法でタッチを検出して応答することができます。

WPF では、タッチが行われたときのイベントとして、タッチ イベントと操作イベントの 2 種類を公開します。タッチ イベントは、タッチスクリーン上の各指とその動きについての未加工のデータを提供します。操作イベントは、入力を特定のアクションとして解釈します。

タッチと操作のサポートは複数の要素クラスで提供されます。**UIElement** クラス、**UIElement3D** クラス、および **ContentElement** クラスは、ユーザーが要素をタッチしたときに発生するイベントを公開します。

UIElement は、タッチ イベントに加えて操作イベントもサポートします。操作は、**UIElement** の拡大縮小、回転、または平行移動として解釈されます。たとえば、写真を表示するアプリケーションでは、コンピューターの画面上で写真をタッチして、写真の移動、ズーム、サイズ変更、回転といった操作を行うことができます。

タッチによるスクロールがサポートされている WPF コントロールを次に示します。

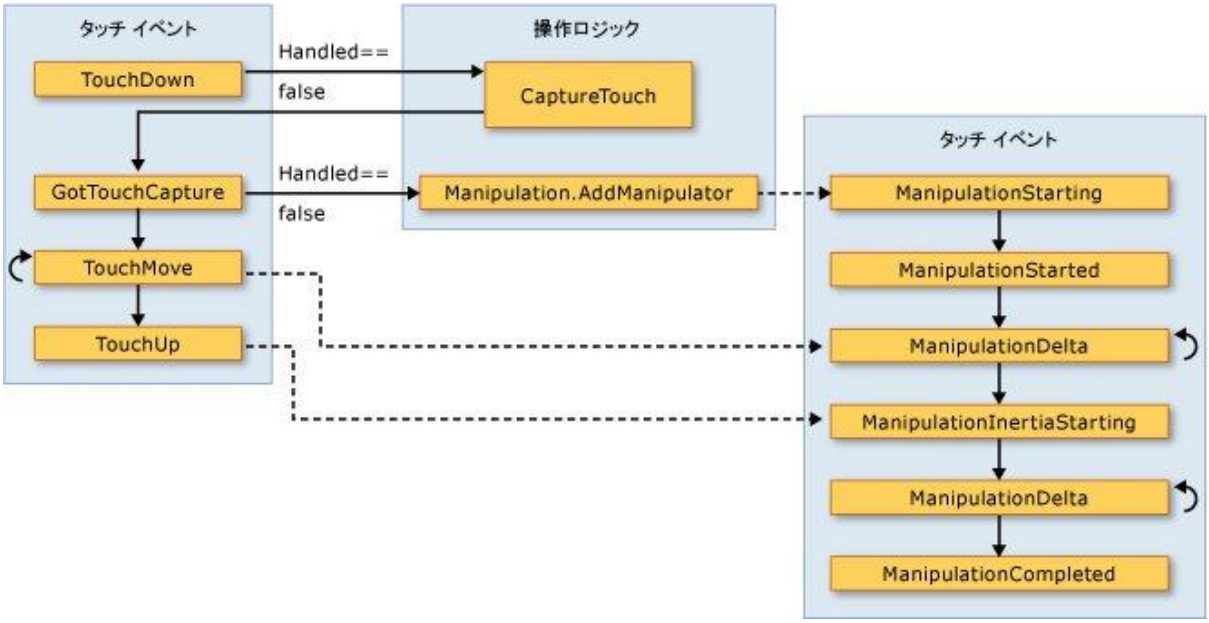
- ComboBox
- ContextMenu
- DataGrid
- ListBox
- ListView
- MenuItem
- TextBox
- ToolBar
- TreeView

ScrollViewer では、**ScrollViewer.PanningMode** 添付プロパティが定義されます。このプロパティでは、水平方向、垂直方向、またはその両方向のタッチによるパンを有効にするか、あるいはいずれも有効にしないかを指定できます。**ScrollViewer.PanningDeceleration** プロパティでは、ユーザーがタッチスクリーンから指を離したときのスクロール速度の低下率を指定します。**ScrollViewer.PanningRatio** 添付プロパティでは、スクロール オフセットと平行移動操作オフセットの比率を指定します。

タッチ イベントと操作イベントの関係

UIElement は、常にタッチ イベントを受け取ることができます。**IsManipulationEnabled** プロパティを true に設定すると、**UIElement** は、タッチ イベントに加えて操作イベントも受け取ることができます。**TouchDown** イベントが処理されない場合 (**Handled** プロパティが false の場合)、操作ロジックは要素に対するタッチをキャプチャし、操作イベントを生成します。**TouchDown** イベントの **Handled** プロパティが true に設定されている場合は、操作ロジックは操作イベントを生成しません。次の図に、タッチ イベントと操作イベントの関係を示します。

タッチ イベントと操作イベント



この図に示したタッチ イベントと操作イベントの関係は次のとおりです。

- デバイスへの最初のタッチによって **UIElement** で **TouchDown** イベントが生成されると、操作ロジックは **CaptureTouch** メソッドを呼び出し、**GotTouchCapture** イベントを生成します。
- **GotTouchCapture** が発生すると、操作ロジックは **Manipulation.AddManipulator** メソッドを呼び出し、**ManipulationStarting** イベントを生成します。
- **TouchMove** イベントが発生すると、操作ロジックは **ManipulationDelta** イベントを生成します。これは **ManipulationInertiaStarting** イベントの前に行われます。
- 要素へのタッチの最後に **TouchUp** イベントが生成されると、操作ロジックは **ManipulationInertiaStarting** イベントを生成します。

タッチの詳細については、「チュートリアル: 初めてのタッチ アプリケーションの作成」および「タッチおよび操作」を参照してください。

ソフトウェア ロゴ プログラム

このガイドラインからもわかるように、タッチ対応 PC 向けの利用しやすい魅力あるアプリケーションを設計するための概念は実にさまざまです。タッチ操作向けに最適化されたアプリケーションを作成するときは、高い顧客満足度を達成するのはもちろん、信頼できる一貫した設計に従っていることが顧客にもわかるようにすることが根本的に重要です。

Windows 7 ソフトウェア ロゴ プログラムはこの適合性を保証する制度で、必要なテストに合格したアプリケーションに次のようなメリットを提供します。

- マイクロソフト パートナー ポイント
- マイクロソフト互換性センターへの優先的な掲載
- ロゴ アートワークとマーケティング ガイド
- Windows エラー報告

Windows 7 ソフトウェア ロゴ プログラムのテストは、無償で申し込むことができます。詳細については、「The Windows 7 Software Logo Program(英語)」を参照してください。

Windows 7 タッチのコード サンプルとアプリケーションの例

このドキュメントで説明した概念について開発者が詳細に確認できる Window 7 タッチのさまざまなコード サンプルが用意されています。また、タッチ操作向けに最適化されたアプリケーションの例も多数あります。このセクションの「タッチ アプリケーションの例」で紹介している Windows 7 タッチ パックもその 1 つです。

Windows 7 タッチのコード サンプルとチュートリアル

Windows SDK のコード サンプル

Windows 7 をサポートする Windows SDK には、オプションとして、タッチ機能向けの次のコード サンプルが含まれています。これらについては MSDN でも紹介しています。概要については、「Windows タッチのサンプル」を参照してください。

- 複数の接触点の検出と追跡
- 操作と慣性のサンプル
- Windows タッチ ジェスチャのサンプル (MTGestures)
- C# の Windows タッチ ジェスチャのサンプル (MTGesturesCS)
- Windows タッチ操作のサンプル (MTManipulation)
- Windows タッチ スクラッチ パッドのサンプル (MTScratchpadWMTouch)
- C# の Windows タッチ スクラッチ パッドのサンプル (MTScratchpadWMTouchCS)
- リアルタイム スタイラスを使用する Windows タッチ スクラッチ パッドのサンプル (MTScratchpadRTStylus)
- C# のリアルタイム スタイラスを使用する Windows タッチ スクラッチ パッドのサンプル (MTScratchpadRTStylusCS)
- ネイティブ C++ の記事とサンプル アプリケーション: プロジェクト 'Hilo' (英語)

Silverlight 4 のコード サンプル

- マルチタッチ入力
- Silverlight でのマルチタッチ サポートの詳細
  - 関連するダウンロード ページ (code.microsoft.com) (英語)
- Microsoft Surface Manipulations and Inertia Sample for Microsoft Silverlight (英語)

WPF 4 のコード サンプル

- WPF アプリケーションのパフォーマンスの最適化
- WPF におけるマルチタッチ操作イベント
- タッチと反応
- マルチタッチの慣性
- Surface Toolkit Beta (英語)

ハンズオン ラボ

ハンズオン ラボでは、チュートリアルのガイダンスに従って、特定の API の使用について学習することができます。Windows 7 タッチ向けのラボは次のものが提供されています。

マルチタッチ ジェスチャ

このマルチタッチ ジェスチャのラボ (言語は C++) では、アプリケーションでマルチタッチ ジェスチャに応答する方法について説明します。このラボでは、ジェスチャを使用して色付きの四角形を操作します。

高度なマルチタッチ ジェスチャ

この高度なマルチタッチ ジェスチャのラボ (言語は C++) は、上記のマルチタッチ ジェスチャのラボの上級編です。このラボでは、高度なジェスチャおよびレンダリングについて説明します。

マルチタッチ スクラッチ パッド

このマルチタッチ スクラッチ パッドのラボ (言語は C#) では、アプリケーションでマルチタッチ入力を読み取り、ウィンドウ内の接触点に線を描画して応答する方法について説明します。どの接触点の入力かがわかるように、描画する線を色分けします。

Windows 7 オンライン トレーニング

Windows 7 オンライン トレーニング (Channel 9) にも、ジェスチャやマルチタッチの使用に関するハンズオン ラボが用意されています。

マルチタッチ - ネイティブ

このマルチタッチ C++ のラボでは、マルチタッチ ハードウェアがあるかどうかをテストする方法、WM_TOUCH メッセージを取得するようにウィンドウを構成する方法、およびシステムでタッチ ID がタッチ入力に関連付けられるしくみについて説明します。

マルチタッチ - MFC

このマルチタッチ MFC のラボでは、マルチタッチ ハードウェアがあるかどうかをテストする方法、マルチタッチ イベントを管理する方法、および複数のタッチ イベントを同時に操作した場合の影響について説明します。

マルチタッチ - WPF 3.5SP1

このマルチタッチ WPF 3.5SP1 のラボでは、マウスで操作する簡単な画像操作アプリケーションを最新のマルチタッチ アプリケーションにアップグレードする方法について説明します。また、PC のタッチ機能の検出方法、および操作プロセッサと慣性プロセッサの使用方法についても説明します。

ジェスチャ - ネイティブ

このジェスチャ C++ のラボでは、マルチタッチ ハードウェアがあるかどうかをテストする方法、ジェスチャ イベントを取得するようにウィンドウを構成する方法、およびジェスチャの引数を抽出して解釈する方法について説明します。

ジェスチャ - MFC

このジェスチャ MFC のラボでは、マルチタッチ ハードウェアがあるかどうかをテストする方法、マルチタッチ ジェスチャ イベントを管理する方法、およびジェスチャ イベントを含むオブジェクトを操作した場合の影響について説明します。

タッチ アプリケーションの例

Windows 7 の Microsoft® タッチ パック

Windows 7 の Microsoft タッチ パックには、マルチタッチ用に最適化された 6 つのアプリケーションとゲームが含まれており、Windows 7 の魅力的な操作性を実際に体験していただけます。このパックには、3 つのカジュアル ゲームと 3 つの Microsoft Surface® アプリケーションが含まれます。

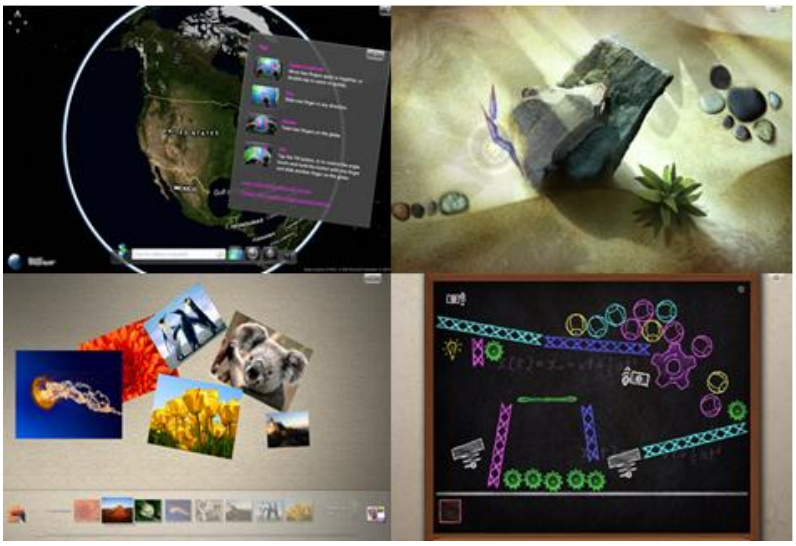


図 1: Windows 7 の Microsoft タッチ パック

Microsoft Windows Product Scout (英語のみ)

Microsoft Windows Product Scout は、Windows 7 PC におけるエクスペリエンスを高める Windows 7 ロゴ取得済みのアプリケーションやハードウェアを紹介するサイトです。Windows タッチについても、注目の分野の 1 つとして紹介されています。

プロジェクト 'Hilo'

'Hilo' とは、Windows 7、Visual Studio 2010、および Visual C++ を活用して、優れたパフォーマンスと応答性を備えたリッチ クライアント アプリケーションを構築する方法を紹介している一連の記事およびサンプル アプリケーションのことです。Hilo は、魅力的なタッチ対応 Windows アプリケーションの設計および開発に役立つソース コードとガイダンスの両方を提供します。

次のステップ

Windows タッチを使用すれば、より自然に、楽しく、直感的に使えるアプリケーションを開発できますが、アプリケーションに求められるのはそれだけではありません。優れた Windows 7 アプリケーションを開発するには、インターフェイス設計、パフォーマンス、セキュリティ、および管理性についても考慮する必要があります。

エクスペリエンス全体がタッチ操作向けに最適化されたマルチタッチ アプリケーションを構築するために、設計の段階で、設計原則やソフトウェア ロゴの要件を見直し、推奨事項について検討するようにしてください。

参考資料

マイクロソフトは、タッチ対応のソフトウェアやハードウェアに関連する各社と協力し、高品質な Windows 7 のユーザー エクスペリエンスの提供に向けて精力的に取り組んでいます。実装に関する包括的なガイダンスとして、以下の参考資料を紹介します。

タイトル	入手場所
Windows タッチの紹介	http://msdn.microsoft.com/ja-jp/windows/hardware/gg487480
Windows タッチ ラーニング パス	http://msdn.microsoft.com/ja-jp/windows/ee633448
スレート PC に関する Windows 7 エンジニアリング ガイダンス	http://msdn.microsoft.com/ja-jp/windows/hardware/gg487456
Tablet and Touch SDK	http://msdn.microsoft.com/en-us/library/ms704849(VS.85).aspx
Windows デベロッパー センター	http://msdn.microsoft.com/ja-jp/windows/default.aspx
MSDN マガジン: Windows 7 のマルチタッチ機能	http://msdn.microsoft.com/ja-jp/magazine/ee336016.aspx
Windows タッチ (MSDN)	http://msdn.microsoft.com/ja-jp/library/dd562197(v=VS.85).aspx
Windows Touch Developer Resources	http://code.msdn.microsoft.com/WindowsTouch/
Windows ユーザー エクスペリエンス ガイドライン	http://msdn.microsoft.com/ja-jp/library/aa511258.aspx
Windows タッチのフォーラム	http://social.msdn.microsoft.com/Forums/ja-JP/tabletandtouch/threads
小型 PC 用の DPI 構成	http://msdn.microsoft.com/ja-jp/windows/hardware/gg487326
Sensor Development Kit for Windows 7	http://code.msdn.microsoft.com/Project/Download/FileDownload.aspx?ProjectName=SensorsAndLocation&DownloadId=5856
Engineering Windows 7 ブログ: Windows 7 でのタッチ	http://blogs.msdn.com/b/e7jp/archive/2009/05/04/9585946.aspx

タイトル	入手場所
Engineering Windows 7 ブログ: インク入力と Tablet PC	http://blogs.msdn.com/b/e7jp/archive/2009/05/21/9633408.aspx
WPF 4 でのタッチのサポート	http://msdn.microsoft.com/ja-jp/library/bb613588.aspx#touch_and_manipulation
Windows Product Scout	http://www.microsoft.com/windows/product-scout/
Windows タッチ アプリケーションのモバイル ユーザー向け強化	http://msdn.microsoft.com/ja-jp/magazine/ee819130.aspx
入力と HID - アーキテクチャとドライバー サポート	http://msdn.microsoft.com/ja-jp/windows/hardware/gg487435
Multi-Touch in Windows 7 [WinHEC 2008、5.3 MB]	http://download.microsoft.com/download/5/E/6/5E66B27B-988B-4F50-AF3A-C2FF1E62180F/MBL-T527_WH08.pptx
MsDev のタッチ向けトレーニング	http://www.msdev.com/Directory/SearchResults.aspx?keyword=touch
プロジェクト 'Hilo'	http://code.msdn.microsoft.com/Hilo