Inter-Client Communication Conventions Manual
Version 2.0
X Consortium Standard
X Version 11, Release 6.3

David Rosenthal

Sun Microsystems, Inc.

Version 2 edited by Stuart W. Marks

SunSoft, Inc.

目 次

第 2 章 Peer-to-Peer Communication by Means of Selections 1

2.1 セレクションの所有権の獲得 2

2.2 セレクションオーナの責務 3

2.3 セレクションの所有権の放棄 5

2.3.1 自発的なセレクションの所有権の放棄 6

2.3.2 強制的なセレクション所有権の放棄 6

2.4 セレクションの要求 6

2.5 大量のデータの転送 8

2.6 セレクションアトムの使用 9

2.6.1 セレクションアトム 9

2.6.1.1 PRIMARY セレクション 10

2.6.1.2 SECONDARY セレクション 10

2.6.1.3 CLIPBOARD セレクション 10

2.6.2 ターゲットアトム 11

2.6.3 副作用をもつセレクションターゲット 14

2.6.3.1 DELETE 14

2.6.3.2 INSERT_SELECTION 14

2.6.3.3 INSERT_PROPERTY 15

2.7 セレクションプロパティの使用 15

2.7.1 テキストのプロパティ 16

2.7.2 タイプ INCR のプロパティ 17

2.7.3 ドローワブルのプロパティ 17

2.7.4 スパンのプロパティ 18

2.8 マネージャセレクション 18

第 4 章 Client-to-Window-Manager Communication 21

4.1 クライアントの行動 22

4.1.1 最上位ウィンドウの生成 . . . 22
4.1.2 クライアントのプロパティ . . . 23
4.1.2.1 プロパティWM_NAME . . . 24
4.1.2.2 プロパティWM_ICON_NAME . . . 24
4.1.2.3 プロパティWM_NORMAL_HINTS . . . 24
4.1.2.4 プロパティWM_HINTS . . . 26
4.1.2.5 プロパティWM_CLASS . . . 29
4.1.2.6 プロパティWM_TRANSIENT_FOR . . . 29
4.1.2.7 プロパティWM_PROTOCOLS . . . 30
4.1.2.8 プロパティWM_COLORMAP_WINDOWS . . . 30
4.1.2.9 プロパティWM_CLIENT_MACHINE . . . 30
4.1.3 ウィンドウマネージャのプロパティ . . . 31
4.1.3.1 プロパティWM_STATE . . . 31
4.1.3.2 プロパティWM_ICON_SIZE . . . 32
4.1.4 ウィンドウの状態の変更 . . . 32
4.1.5 ウィンドウの配置構成 . . . 35
4.1.6 ウィンドウの属性の変更 . . . 37
4.1.7 入力フォーカス . . . 38
4.1.8 カラーマップ . . . 41
4.1.9 アイコン . . . 43
4.1.10 ポップアップウィンドウ . . . 45
4.1.11 ウィンドウグループ . . . 46
4.2 ウィンドウマネージャの行動に対するクライアントの応答 . . . 46
4.2.1 親の変更 . . . 46
4.2.2 操作のリダイレクト . . . 48
4.2.3 ウィンドウの移動 . . . 49
4.2.4 ウィンドウのサイズの変更 . . . 50
4.2.5 アイコン化 . . . 50
4.2.6 カラーマップの変更 . . . 50
4.2.7 入力フォーカス . . . 51
4.2.8 ClientMessage イベント . . . 52
4.2.8.1 ウィンドウの消去 . . . 52
4.2.9 リクエストのリダイレクト . . . 53

4.3 セレクションによるウィンドウマネージャとの通信 54
4.4 ウィンドウマネージャのためのプロパティのタイプの要約 54

第 6 章 Manipulation of Shared Resources **55**

6.1 入力フォーカス 55
6.2 ポインタ 55
6.3 グラブ 56
6.4 カラーマップ 57
6.5 キーボードマッピング 58
6.6 モデファイヤマッピング 59

第2章　Peer-to-Peer Communication by Means of Selections

セレクションは、例えばウィンドウ間でカットアンドペーストするような、クライアント間の情報交換のために X バージョン 11 が定義している主要なメカニズムである。（それぞれアトムで指定される）任意の数のセレクションが存在し、それらはサーバに対してグローバルであることに注意。セクション 2.6 ではアトムの選択について説明する。それぞれのセレクションはクライアントに所有され、ウィンドウに所属している。

セレクションはオーナとリクエスタとの間で通信する。オーナは自分のセレクションの値を表すデータをもち、リクエスタはそれを受け取る。セレクションの値を取得したいリクエスタは、次のことを与える。

- セレクションの名前
- プロパティの名前
- ウィンドウ
- 必要とするデータのタイプを表すアトム
- オプショナルで、リクエストに対するいくつかのパラメータ

オーナは、現在セレクションを所有していれば、イベントを受け取り、次のことをするように期待されている。

- セレクションの内容を、要求されているデータのタイプへ変換する。
- このデータを、指定されているウィンドウの指定されているプロパティにおく。
- リクエスタにイベントを送り、プロパティが利用可能であると知らせる。

クライアントはこのメカニズムを使用するように強く推奨されている。特に、永続的なウィンドウにテキストを表示するにもかかわらず、そのテキストを選択して文字列に変換する能力を提供しなければ、確実に反社会的であるとみなされる。

オーナとリクエスタの間で転送されるすべてのデータは、X バージョン 11 の環境では、通常サーバを介して伝わるべきであることに注意。クライアントは、もうひとつのクライアントが同じファイルをオープンできると想定することができないし、あるいは直接通信できると想定することすらできない。そのほかのクライアントは完全に異なったネットワークのメカニズムでサーバと会話しているかもしれない（例えば、一方のクライアントは DECnet であり、もう一方のクライアントは TCP/IP であるかもしれない）。このように、（ファイル名、ホスト名、ポート番号のような）データへの間接的な参照を手渡すことは、両方のクライアントが特別に合意している場合に限り許される。

2.1　セレクションの所有権の獲得

特定のセレクションの所有権を獲得したいクライアントは、SetSelectionOwner を呼ばなければならない。それは次のように定義される。

SetSelectionOwner
selection: アトム
owner: ウィンドウ または `None`
time: タイムスタンプ または `CurrentTime`

クライアントは、*selection* フィールドをそのセレクションを表すアトムにセットし、*owner* フィールドを自分が作成したウィンドウにセットし、*time* フィールドを、そのセレクションに関する現在の最新変更時刻と現在のサーバの時刻の間の、ある時刻にセットしなければならない。この値 *time* は通常そのセレクションを獲得するきっかけとなったイベントのタイムスタンプから得られるものである。クライアントは値 *time* を `CurrentTime` にセットしてはならない。なぜなら、クライアントがそうしてしまうと、自分がセレクションの所有権を獲得した時刻を知る方法がなくなってしまうからである。クライアントは、リクエスタがイベントをセレクションオーナに配送できるように、自分が生成したウィンドウを使用するべきである[1]。

規約

セレクションを獲得しようとしているクライアントは、SetSelectionOwner リクエストの値 *time* を、`CurrentTime` ではなく、獲得しようとするきっかけとなったイベントのタイムスタンプにセットするべきである。プロパティに長さ 0 のデータを追加することが、この目的のためにタイムスタンプを取得する方法であり、タイムスタンプは対応する PropertyNotify イベントにある。

SetSelectionOwner リクエストの *time* が、サーバの現在時刻と比べて進んでいるか、引数 *selection* の持ち主が交替した最新時刻と比べて遅れている場合、SetSelectionOwner リクエストはクライアントには成功したようにみえるが、所有権は実際には移っていない。

クライアントは直接的にそのほかのクライアントを指定できないので、引数 *owner* のウィンドウが、GetSelectionOwner の返答や、SelectionRequest と SelectionClear イベントで所有権をもつクライアントを参照するために使用され、そしておそらく当のセレクションを記述するプロパティを置く場所として使用される。特定のセレクションのオーナを発見するためには、クライアントは GetSelectionOwner を行使しなければならない。それは次のように定義される。

GetSelectionOwner
selection: アトム
→
owner: ウィンドウ または `None`

[1] 現在のところ、リクエスタがイベントをセレクションオーナに送るようにプロトコルが要求する部分はいっさいない。この制限は、起こりうる将来の拡張を用意するために課されている。

規約

クライアントはセレクションの所有権に関して、なにか視覚的な確証を与えるように見込まれている。このフィードバックを堅実なものにするため、クライアントは次のような手順を実行しなければならない。

SetSelectionOwner(*selection*=PRIMARY, *owner*=ウィンドウ, *time*=タイムスタンプ)
owner = GetSelectionOwner(*selection*=PRIMARY)
if (owner != ウィンドウ) 失敗

SetSelectionOwner リクエストが（単に成功したように見えたのではなく）成功した場合、そのリクエストを発行したクライアントは、引数 *time* で始まる期間のセレクションオーナであるとサーバに記録される。

2.2 セレクションオーナの責務

セレクションの値をリクエスタが欲しいとき、オーナは SelectionRequest イベントを受け取る。それは次のように定義される。

SelectionRequest
owner: ウィンドウ
selection: アトム
target: アトム
property: アトム または `None`
requestor: ウィンドウ
time: タイムスタンプ または `CurrentTime`

owner フィールドと *selection* フィールドは、SetSelectionOwner リクエストで指定された値である。オーナは、セレクションを所有したときの時刻とそのタイムスタンプを比較しなければならない。もし *time* が範囲外である場合は、プロパティに `None` をセットした SelectionNotify イベントを（空のイベントマスクをもつ SendEvent リクエストで）ウィンドウ *requestor* へ送ることによって、その SelectionRequest を拒絶しなくてはならない。

より先進的なセレクションオーナは、セレクションの値の履歴を維持し、たとえ今は所有していなくてもオーナが所有していたときのセレクションの値をリクエスタに返答してもよいだろう。

property フィールドが `None` であれば、リクエスタは時代遅れのクライアントである。指定されているターゲットアトムをプロパティ名として使用し返答することによって、オーナはこうしたクライアントをサポートすることが推奨されている。

そうでなければオーナは *target* を、セレクションを変換するべきフォームを決定する目的で使用する。いくつかのターゲットは、リクエスタがリクエストと一緒にパラメータを渡すことができるように定義されているかもしれない。その SelectionRequest が指定する *property* の中に、オーナはこれらのパラメータを見つけるかもしれない。そのプロパティのタイプ、フォーマット、内容は、ターゲットの定義に依存している。そのターゲットがパラメータをもつように定義されていなければ、プロパティが存在していてもオーナはそれを無視するべきである。セレクションをそのターゲット（および、もしあればパラメータ）に基づいたフォームに変換できないときは、前述のようにオーナは SelectionRequest を拒絶しなくてはならない。

property フィールドが `None` でなければ、オーナはセレクションを変換した結果のデータを、ウィンドウ *requestor* の指定された *property* に置かなくてはならない。そしてそのプロパティのタイプをある適切な値にセットしなくてはならない。また、そのタイプは指定されている *target* と同じである必要はない。

規約

SelectionRequest イベントの返答に使われるすべてのプロパティはウィンドウ *requestor* に置かねばならない。

どちらの場合でも、セレクションを構成するデータをウィンドウ *requestor* に格納できないとき（例えば、サーバが十分なメモリをあてがうことができないという理由などで）、オーナは前述のように SelectionRequest を拒絶しなくてはならない。セクション 2.5 も参照すること。

もしプロパティをうまく格納できたら、オーナはウィンドウ *requestor* に SelectionNotify イベントを（空のマスクをもった SendEvent リクエストで）送ることによって、変換が成功したことを知らせなければならない。SelectionNotify は次のように定義される。

SelectionNotify
owner: ウィンドウ
selection, *target*: アトム
property: アトム または `None`
time: タイムスタンプ または `CurrentTime`

オーナは引数 *selection*, *target*, *time*, *property* を SelectionRequest イベントで受け取った値にセットしなくてはならない。（引数 *property* に `None` をセットすることは、要求された変換ができなかったことを示すことに注意。）

規約

SelectionNotify イベントの引数 *selection*, *target*, *time*, *property* は、SelectionRequest イベントで受け取った値にセットしなければならない。

もしオーナが、同じリクエスタ、セレクション、ターゲット、タイムスタンプをもつ 1 つ以上の SelectionRequest イベントを受け取った場合は、オーナは届いたのと同じ順序でそれらに応答しなければならない。

根本的理由

同一のリクエスタ、ウィンドウ、セレクション、ターゲット、タイムスタンプを使用し、プロパティだけは異なっている、複数の未決定なリクエストをリクエスタは要求できる。このとき、それらのリクエストのひとつが失敗した場合、その結果となる SelectionNotify イベントの引数 *property* は `None` にセットされる。このことから、順番にリクエストが応答されなければ、リクエスタはどのリクエストが失敗したのか決定できなくなるかもしれない。

プロパティに格納されたデータは、いつかは消去しなければならない。そうするためには、その責任を割り当てる規約が必要である。

規約

セレクションのリクエスタは、SelectionNotify イベントで自分が指定したプロパティ名のプロパティ（セクション 2.4 を見よ）、あるいはタイプ MULTIPLE のプロパティに含まれるプロパティを消去する責任がある。

セレクションオーナは、セレクションを構成するデータが実際に転送されたという裏づけを必要とすることが多いだろう。（例えば、その操作がオーナのもつ内部データ構造に対して副作用をもつ場合は、リクエスタがデータをうまく受け取ったことを示すまでは、オーナはその操作を行うべきではない。）オーナは、セレクションのデータが転送されたと決めてかかる前に、ウィンドウ *requestor* に関する PropertyNotify イベントへの関心を表明して、そしてその SelectionNotify イベントの *property* が消去されるまで待たなくてはならない。MULTIPLE リクエストでは、異なる変換が別々に確認を必要とする場合、セレクションオーナはまた、その SelectionNotify イベントの *property* が指定する個々のプロパティが消去されるのを監視することもできる。

何か別なクライアントがセレクションを獲得すると、以前のオーナは SelectionClear イベントを受け取る。それは次のように定義される。

SelectionClear
owner: ウィンドウ
selection: アトム
time: タイムスタンプ

引数 *time* は所有権の持ち主が変わった時刻であり、引数 *owner* は以前のオーナが自分の SetSelectionOwner リクエストで指定したウィンドウである。

オーナが転送の途中で（すなわち、リクエスタがすべてのデータを受け取ったという通知をオーナが受け取る前に）所有権を失う場合、進行中の転送サービスが完了するまで、オーナはそれを続行しなくてはならない。

セレクションの値はまったく変わってしまったが、オーナが同じクライアントになった場合（例えば、以前選択したのと同じ **xterm** で、まったく異なったテキストを選択したとき）、そのクライアントはまるで自分がオーナではなかったかのように、新しいタイムスタンプを使用してセレクションの所有権を再獲得するべきである。セレクションの値は変更されたが、依然として合理的にそれが同一の選択オブジェクトとして見ることができる場合[2]、そのオーナは何もするべきではない。

2.3 セレクションの所有権の放棄

クライアントはセレクションの所有権を自発的に放棄してもよいし、何かそのほかのクライアントによる行動の結果として、強制的に失うことになるかもしれない。

[2] これら 2 つのケースの境界線はソフトウェア開発者側の判断の問題である。

2.3.1　自発的なセレクションの所有権の放棄

自発的にセレクションの所有権を手放すためには、クライアントはそのセレクションのアトムに対して、*owner* に `None` を指定し、*time* にそのセレクションを獲得するために使用したタイムスタンプを指定して、SetSelectionOwner リクエストを実行しなければならない。

代わりに、そのクライアントが SetSelectionOwner リクエストの *owner* の値として使用したウィンドウを破壊してもよいし、あるいは終了してもよい。どちらの場合でも、そのウィンドウあるいはクライアントに関するセレクションの所有権は `None` にもどることになる。

2.3.2　強制的なセレクション所有権の放棄

クライアントがセレクションの所有権を放棄する場合や、何かそのほかのクライアントが SetSelectionOwner をそのセレクションに対して実行して、そのセレクションを強制的に再割り当てする場合、以前のオーナは SelectionClear イベントを受け取ることになる。SelectionClear イベントの定義については、セクション 2.2 を見よ。

そのタイムスタンプはセレクションの持ち主が変わった時刻である。*owner* フィールドは現在のオーナが SetSelectionOwner リクエストで指定したウィンドウである。

2.4　セレクションの要求

ある特別なフォームでセレクションの値を取得したいクライアント（リクエスタ）は、ConvertSelection リクエストを発行する。それは次のように定義される。

ConvertSelection
selection, *target*: アトム
property: アトム または `None`
requestor: ウィンドウ
time: タイムスタンプ または `CurrentTime`

引数 *selection* はその該当するセレクションを指定する。引数 *target* は必要とする情報のフォームを指定する。使用するべき適切なアトムを選択するための情報については、セクション 2.6 を見よ。リクエスタは引数 *requestor* を、自分が生成したウィンドウにセットしなければならず、オーナは返答のプロパティをそこへ配置することになる。リクエスタは引数 *time* を、そのセレクションの値を要求するきっかけとなったイベントのタイムスタンプにセットしなければならない。クライアントは `CurrentTime` を指定してはいけないことに注意。

規約

クライアントは ConvertSelection リクエストの引数 *time* に対して、`CurrentTime` を使用してはならない。代わりに、そのリクエストを発行する原因となったイベントのタイムスタンプを使用しなければならない。

リクエスタは引数 *property* を、オーナがセレクションの値を報告するために使用できるプロパティの名前にセットしなければならない。リクエスタは ConvertSelection リクエストを発行する前に、ウィンドウ *requestor* にその名前のプロパティが存在しないことを保証しなければならない[3]。この規則の例外は、リクエスタがリクエストと共にパラメータを渡そうと思っているときである（下記を見よ）。

根本的理由

将来互換性を保ったままターゲットがパラメータをとれるように拡張することが可能であるように、リクエスタはリクエストを発行する前にそのプロパティを消去する必要がある。また、セレクションのリクエスタは、そのセレクションを所有するクライアントも、そのセレクションを獲得したウィンドウも知る必要がないことに注意。

いくつかのターゲットは、リクエスタがリクエストと一緒にパラメータを渡すことができるように定義されているかもしれない。リクエスタがリクエストにパラメータを与えることを望むなら、ConvertSelection リクエストを発行する前に、ウィンドウ *requestor* の引数 *property* にそのパラメータを配置しなければならず、そしてそのプロパティをリクエストで指定しなければならない。

いくつかのターゲットは、パラメータがオプショナルであると定義されているかもしれない。そのようなターゲットのリクエストにパラメータがまったく与えられない場合、リクエスタは ConvertSelection リクエストを発行する前にそのプロパティが存在していないことを保証しなければならない。

プロトコルは *property* フィールドが `None` にセットされることを許していて、その場合はオーナがプロパティ名を選ぶことが見込まれている。しかしながら、オーナがこれを安全に選択することは困難である。

規約

1. リクエスタは ConvertSelection リクエストの引数 *property* に `None` を使用してはならない。
2. 引数 *property* が `None` である ConvertSelection リクエストを受け取ったオーナは、時代遅れのクライアントと会話している。そのようなオーナは、ターゲットアトムを返答に用いるべきプロパティ名として選ばなければならない。

ConvertSelection リクエストの結果は SelectionNotify イベントを受け取ることである。SelectionNotify イベントの定義についてはセクション 2.2 を見よ。

その引数 *requestor*, *selection*, *time*, *target* は、ConvertSelection リクエストのときのものと同じである。

引数 *property* が `None` であれば、その変換は拒絶された。これは、そのセレクションのオーナが存在しないか、もしくはそのターゲットで暗示された変換をオーナがサポートしていないか、あるいはデータを受け入れるのに十分なスペースをサーバがもっていないことを意味している。

引数 *property* が `None` でなければ、ウィンドウ *requestor* にそのプロパティが存在している。セレクションの値は、次のように定義されている GetProperty リクエストを使用することによって、このプロパティから取ってくることができる。

[3] この要求はバージョン 2.0 で新しくなった。そして一般的に、現存するクライアントはこの要求に順応しない。こうしたクライアントが壊れてしまうのを防ぐために、クライアントが更新されるのに十分な時間が経過するまで、決して現存するターゲットを拡張してパラメータをとれるようにしてはならない。ターゲット MULTIPLE はバージョン 1.0 でパラメータをとれるように定義されていたこと、そしてその定義が変わっていないことに注意。そのため、MULTIPLE に関して順応の問題はまったくない。

GetProperty
window: ウィンドウ
property: アトム
type: アトム または `AnyPropertyType`
long-offset, *long-length*: CARD32
delete: BOOL
→
type: アトム または `None`
format: {0, 8, 16, 32}
bytes-after: CARD32
value: LISTofINT8, LISTofINT16, LISTofINT32

GetProperty を使用してセレクションの値を取ってくるとき、その引数 *property* を SelectionNotify イベントの対応する値にセットしなくてはならない。リクエスタはセレクションオーナがどのタイプを用いるのか事前に知る方法がないので、引数 *type* には `AnyPropertyType` をセットしなくてはならない。セレクションのデータをすべて取ってくるためには、いくつかの GetProperty リクエストが必要になるかもしれない。その場合、それぞれ（のリクエスト）は引数 *long-offset* をその時点までに受け取っているデータの量にセットし、そして引数 *size* をある適正なバッファサイズ（セクション 2.5 を見よ）にセットしなくてはならない。戻り値 *bytes-after* が 0 であれば、プロパティ全体が転送された。

セレクションのデータをすべて取ってきたら（それにはいくつかのプロパティの値を得ることが必要になるかもしれない。セクション 2.7 を見よ。）、リクエスタは引数 *delete* を `True` にセットした GetProperty リクエストを使用して、SelectionNotify イベントのプロパティを消去しなければならない。前述のように、そのデータを消去しなければ、データがリクエスタに転送されたことをオーナが知る方法はない。

規約

リクエスタはデータをすべて取ってきたら、SelectionNotify で指定されているプロパティを消去しなくてはならない。リクエスタはセレクションのデータをすべてうまく取ってきた後で、DeleteProperty か GetProperty（*delete*==`True`）を行使しなければならない。より詳しい情報についてはセクション 2.5 を見よ。

2.5　大量のデータの転送

セレクションは大きくなることもできるが、それは次の 2 つの問題を引き起こす。

- サーバへ大量のデータを転送することは高くつく。
- すべてのサーバにはプロパティに格納できるデータの量に限界がある。この限界を越えれば、セレクションのオーナがデータを格納するときに使用する ChangeProperty リクエストの際に、Alloc エラーが生じることになる。

サーバの資源が有限であるという問題は、次の規約によって扱われる。

規約

1. セレクションオーナは、（接続時のハンドシェイクで受け取った最大リクエストサイズと比べて相対的に）大きなセレクションを表現するデータを INCR プロパティのメカニズムを使用して転送するべきである（セクション 2.7.2 を見よ）。
2. SetSelectionOwner を使用してセレクションの所有権を獲得するクライアントはすべて、プロパティを変更するリクエストの際の Alloc エラーを処理する手配をしておかなくてはならない。このことは、Xlib を使用するクライアントでは、関数 XSetErrorHandler を用いてデフォルトのハンドラをオーバライドすることと関係する。
3. セレクションのオーナは、プロパティの格納を承認するための SelectionNotify イベントで返答する前に、セレクションを表すプロパティを格納する間 Alloc エラーがいっさい生じなかったことを確かめなければならない。
4. （最大リクエストサイズと比べて相対的に）大量のデータを格納するとき、クライアントは適正な量のデータを求める一連の ChangeProperty（*mode*==Append）リクエストを使用しなければならない。これによってサーバのロックを回避することができ、Alloc エラーがもたらすデータの浪費を制限できる。
5. セレクションのデータを格納している間に Alloc エラーが発生したら、このセレクションのために格納したすべてのプロパティを消去しなければならず、そしてその ConvertSelection リクエストを拒絶しなければならない（セクション 2.2 を見よ）。
6. 極端に長い時間にわたってサーバをロックするのを避けるため、プロパティから大量のデータを取ってくるリクエスタは、それぞれが適正な量のデータを要求する一連の GetProperty リクエストを実行しなければならない。

実装者へのアドバイス

シングルスレッドのサーバは、大量のデータを転送する間、ロックしないように注意しなければならない。

2.6 セレクションアトムの使用

新しいアトムを定義するとサーバの資源は消費され、サーバを再び初期化するまでそれらは解放されない。すなわち、新しく造り出されるアトムの必要性を減らすことが、セレクションアトムを使用することの重要な目標である。

2.6.1 セレクションアトム

それぞれアトムによって指定される、不特定な数のセレクションがありうる。しかしながら、クライアント間の規約に従うためには、クライアントは次の 3 つのセレクションだけを扱う必要がある。

- PRIMARY
- SECONDARY

- CLIPBOARD

関連するクライアントのグループのなかで、プライベートなコミュニケーションのために、そのほかのセレクションを自由に使ってもよい。

2.6.1.1　PRIMARY セレクション

アトム PRIMARY で指定されるセレクションは、引数を 1 つだけとるすべてのコマンドに対して使用され、セレクションのメカニズムを用いるクライアント間のコミュニケーションの主要な手段である。

2.6.1.2　SECONDARY セレクション

アトム SECONDARY で指定されるセレクションは、次のように使用する。

- 2 つの引数をとるコマンドに対する第 2 の引数として（例えば「PRIMARY セレクションと SECONDARY セレクションを入れ換える」）。
- PRIMARY セレクションが存在して、ユーザがそれを乱したくないときに、データを獲得する手段として。

2.6.1.3　CLIPBOARD セレクション

アトム CLIPBOARD で指定されるセレクションは、クライアント間で転送するデータ、すなわち通常カットアンドペーストやコピーアンドペーストするデータを保持するのに用いる。

クライアントがデータをクリップボードに転送したいときは、

- クライアントは CLIPBOARD の所有権を主張しなければならない。
- 所有権を獲得するのに成功したら、CLIPBOARD の内容を求めるリクエストに対して、通常の方法で（データを保持してそれを返すことができるように）返答する準備をしなければならない。そのリクエストは下記のクリップボードクライアントが生成する。
- 所有権を獲得できなかったら、カットするクライアントは実際にカットを実行するべきではなく、またクライアントが実際にデータをクリップボードに転送してしまったと思わせるようなフィードバックを与えるべきではない。

転送するべきデータが変化するときはいつでも、オーナはこの過程を繰り返さなければならない。

クリップボードからデータをペーストしたいクライアントは、通常の方法で CLIPBOARD セレクションの内容を要求しなくてはならない。

クライアントが実際にデータを消去やコピーしている間を除き、CLIPBOARD セレクションのオーナは、その目的のために履行されるただ 1 つの特別なクライアントである。このクライアントは次のように、クリップボードの内容を最新のものに維持し、クリップボードのデータを求めるリクエストに応答する。

- クライアントは CLIPBOARD セレクションの所有権を主張し、クリップボードのデータが変化するときはいつでも、その所有権を再度主張しなくてはならない。
- クライアントは、（もうひとつのクライアントがクリップボードに何か新しいデータを与えるために）自分がセレクションを失う場合、
  - SelectionClear イベントのタイムスタンプを用いて、新しいオーナからセレクションの内容を取得しなければならない。
  - それと同じタイムスタンプを用いて、CLIPBOARD セレクションの所有権を再度主張しようとしなければならない。
  - この試みがうまくいかなかったら、新しく獲得されるタイムスタンプを用いてその処理を再スタートしなければならない。このタイムスタンプは、CLIPBOARD セレクションの現在のオーナに、そのセレクションを TIMESTAMP へ変換するように求めることによって獲得しなくてはならない。この変換が拒絶されるか、あるいは同一のタイムスタンプを 2 度受け取る場合は、そのクリップボードクライアントは新規のタイムスタンプを通常の方法（例えば長さ 0 のデータをプロパティに追加すること）で獲得しなければならない。
- CLIPBOARD の内容を求めるリクエストに通常の方法で応答しなければならない。

特別な CLIPBOARD クライアントは必要ではない。カットするクライアントと、ペーストするクライアントが使用するプロトコルは、CLIPBOARD クライアントが動作していてもいなくても同一である。その特別なクライアントを動作させる理由は、次のことを含んでいる。

- 持続 — カットするクライアントが堕ちたり終了するようなことがあっても、クリップボードの値は依然として利用可能である。
- フィードバック — クリップボードクライアントはクリップボードの内容を表示できる。
- 簡単 — データを消去するクライアントは、長い間そのデータを保持しておく必要がない。そのため、レース条件が問題を引き起こす機会を減せる。

クリップボードクライアントを動作させない理由は、次のことを含んでいる。

- 性能 — 実際に必要とするとき（すなわち、あるクライアントが実際にそのデータを望んだとき）にだけ、データを転送する。
- 適応性 — クリップボードのデータは、1 つより多くのターゲットで利用可能であるかもしれない。

2.6.2 ターゲットアトム

リクエスタが ConvertSelection リクエストのターゲットとして与えるアトムは、与えられるデータのフォームを決定する。そのようなアトムのセットは拡張可能であるが、一般的に受理されるターゲットアトムの基本セットが必要とされる。このための出発点として、次の表は現在までに提案されたものを含んでいる。

アトム	タイプ	受け取るデータ
ADOBE_PORTABLE_DOCUMENT_FORMAT	STRING	[1]
APPLE_PICT	APPLE_PICT	[2]
BACKGROUND	PIXEL	ピクセル値のリスト
BITMAP	BITMAP	ビットマップ資源識別子のリスト
CHARACTER_POSITION	SPAN	セレクションの最初と最後（バイト）
CLASS	TEXT	（セクション 4.1.2.5 を見よ）
CLIENT_WINDOW	WINDOW	セレクションオーナの最上位ウィンドウのいずれか
COLORMAP	COLORMAP	カラーマップ資源識別子のリスト
COLUMN_NUMBER	SPAN	カラム番号の最初と最後
COMPOUND_TEXT	COMPOUND_TEXT	コンパウンドテキスト
DELETE	NULL	（セクション 2.6.3.1 を見よ）
DRAWABLE	DRAWABLE	ドローワブル資源識別子のリスト
ENCAPSULATED_POSTSCRIPT	STRING	[3], Appendix H[4]
ENCAPSULATED_POSTSCRIPT_INTERCHANGE	STRING	[3], Appendix H
FILE_NAME	TEXT	ファイルの完全なパス名
FOREGROUND	PIXEL	ピクセル値のリスト
HOST_NAME	TEXT	（セクション 4.1.2.9 を見よ）
INSERT_PROPERTY	NULL	（セクション 2.6.3.3 を見よ）
INSERT_SELECTION	NULL	（セクション 2.6.3.2 を見よ）
LENGTH	INTEGER	セレクションのバイト数[5]
LINE_NUMBER	SPAN	行番号の最初と最後
LIST_LENGTH	INTEGER	セレクションの分割部分の数
MODULE	TEXT	選択されているプロシージャの名前
MULTIPLE	ATOM_PAIR	（以降の説明を見よ）
NAME	TEXT	（セクション 4.1.2.1 を見よ）
ODIF	TEXT	ISO オフィスドキュメント相互変換フォーマット
OWNER_OS	TEXT	オーナクライアントのオペレーティングシステム
PIXMAP	PIXMAP	ピクスマップ資源識別子のリスト[6]
POSTSCRIPT	STRING	[3]
PROCEDURE	TEXT	選択されているプロシージャの名前
PROCESS	INTEGER, TEXT	オーナのプロセス識別子
STRING	STRING	ISO Latin-1（+タブ+改行）テキスト
TARGETS	ATOM	有効なターゲットアトムのリスト
TASK	INTEGER, TEXT	オーナのタスク識別子
TEXT	TEXT	オーナが選択するエンコーディングのテキスト
TIMESTAMP	INTEGER	セレクション獲得に使用したタイムスタンプ
USER	TEXT	オーナを動作させているユーザの名前

参考文献:

[1] Adobe Systems, Incorporated. *Portable Document Format Reference Manual*. Reading, MA, Addison-Wesley, ISBN 0-201-62628-4.

[2] Apple Computer, Incorporated. *Inside Macintosh, Volume V*. Chapter 4, “Color QuickDraw,” Color Picture Format. ISBN 0-201-17719-6.

[4] ターゲット ENCAPSULATED_POSTSCRIPT と ENCAPSULATED_POSTSCRIPT_INTERCHANGE は（それぞれ）セレクションターゲットのレジストリに現れるターゲット _ADOBE_EPS と _ADOBE_EPSI に等しい。_ADOBE_ ターゲットの使用はやめることになっているが、クライアントは過去の互換性のために、それらのサポートを続けるように推奨されている。

[5] 異なる量のデータを返すようないくつかのターゲットのどれかひとつに、セレクションが変換されるかもしれないので、この定義はあいまいである。複数のターゲットがあるとすれば、これらの内のどれが LENGTH の結果に対応するのかをリクエスタが知る方法はない。LENGTH への変換の結果については、まったく何も保証できないとクライアントは勧告されている。すなわち、その使用はやめることになっている。

[6] このドキュメントの以前の版では誤って、ターゲット PIXMAP の変換は、タイプ PIXMAP ではなく、タイプ DRAWABLE のプロパティを返すと指定していた。実装者はこのことを承知していなければならないし、古いクライアントとの互換性を考慮して、タイプ DRAWABLE を同様にサポートするように望んでよい。

[3] Adobe Systems, Incorporated. *PostScript Language Reference Manual*. Reading, MA, Addison-Wesley, ISBN 0-201-18127-4.

時が経つにつれてこの表は大きくなっていくと思われる。

セレクションオーナは次のターゲットをサポートするように要求されている。そのほかのターゲットはすべてオプショナルである。

- TARGETS — オーナは、現在のセレクションを変換する試みが（Alloc エラーのような予期できない問題がなければ）成功することになるターゲットを表す、アトムのリストを返さなければならない。このリストは要求されるすべてのアトムを含んでいなければならない。

- MULTIPLE — ターゲットアトム MULTIPLE は、ConvertSelection リクエストでプロパティが指定されるときだけ有効である。SelectionRequest イベントの引数 *property* が None であり、*target* が MULTIPLE であれば、それを拒絶しなければならない。

  セレクションオーナが SelectionRequest（*target*==MULTIPLE）イベントを受け取るとき、そのリクエストで指定される *property* の内容は、2 つのアトムの組のリストである。1 番目のアトムはターゲットを指定し、2 番目のアトムはプロパティ（ここで None は有効ではない）を指定している。その効果は次のことを除き、（1 つのイベントがアトムのペアをそれぞれ表す）一連の SelectionRequest イベントをオーナが受け取ったかのようでなければならない。

  – オーナはリクエストされた変換をすべて実行したときだけ、SelectionNotify で返答しなければならない。
  – MULTIPLE プロパティに含まれるアトムが指定するターゲットをオーナが変換できない場合、プロパティに含まれるそのアトムを None で置換しなければならない。

  規約

  MULTIPLE プロパティのエントリは、それらがプロパティで出現する順序に従って処理しなければならない。より多くの情報についてはセクション 2.6.3 を見よ。

  リクエスタはその変換からデータをコピーしたとき、個々のプロパティをそれぞれ消去しなければならない。そして、すべてのデータをコピーしたら、MULTIPLE リクエストで指定した *property* を消去しなければならない。

  そのほか、それらのリクエストは独立に処理されるべきであり、独立に成功するか失敗するはずである。ターゲット MULTIPLE は、オーナとリクエスタの間のプロトコルトラフィックを減らす最適化であり、トランザクションのメカニズムではない。例えばクライアントは、データを求めるターゲットとターゲット DELETE の、2 つのターゲットを含む MULTIPLE リクエストを発行するかもしれない。仮にデータを求めるターゲットを変換できなくても、ターゲット DELETE は依然として処理されることになる。

- TIMESTAMP — いくつかのレース条件を避けるために、オーナが所有権を獲得するために使用したタイムスタンプを、リクエスタが発見できることは重要である。プロトコルが変更され、所有権を獲得するために使用したタイムスタンプを GetSelectionOwner リクエストが返すようになるまで、あるいはそうならないのであれば、セレクションオーナは TIMESTAMP への変換をサポートし、自分がセレクションを獲得するために用いたタイムスタンプを返せるようにするべきである。

2.6.3　副作用をもつセレクションターゲット

いくつかのターゲット（例えば DELETE）には副作用がある。これらのターゲットをあいまいでないようにするために、MULTIPLE プロパティのエントリをそのプロパティに出現する順序で処理しなければならない。

一般的に、副作用をもつターゲットは情報をいっさい返さない。すなわち、タイプ NULL で長さ 0 のプロパティを返す（タイプ NULL が意味するのは、文字列 “NULL” に対する InternAtom の結果であり、値 0 ではない。）。すべての場合において、変換を受理する前に、リクエストされている副作用を実行しなければならない。リクエストされている副作用を実行できない場合は、それに対応する変換リクエストを拒絶しなくてはならない。

規約

1. 副作用のあるターゲットはいっさい情報を返してはいけない（すなわち、タイプ NULL で長さ 0 のプロパティを返さなければならない。）。
2. ターゲットの副作用は変換を受理する前に実行しなければならない。
3. ターゲットの副作用を実行できない場合は、それに対応する変換を拒絶しなければならない。

問題

時間のかかる変換が成功するまで ConvertSelection リクエストへの応答を遅らす必要があることが、イントリンシックスのインタフェイスに身を入れてとりかかる必要のある問題を引き起こしている。

これらの副作用をもつターゲットは「PRIMARY セレクションと SECONDARY セレクションを交換する」のような操作を実装するために使用される。

2.6.3.1　DELETE

セレクションのオーナがセレクションを DELETE に変換するリクエストを受け取るとき、オーナはそれに対応するセレクションを消去しなければならず（そうすることがオーナの内部データ構造に対してどのようなことを意味するとしても）、消去が成功した場合はタイプ NULL で長さ 0 のプロパティを返さなければならない。

2.6.3.2　INSERT_SELECTION

セレクションのオーナがセレクションを INSERT_SELECTION に変換するリクエストを受け取るとき、指定されるプロパティはタイプ ATOM_PAIR である。1 番目のアトムはセレクションを指定し、2 番目はターゲットを指定する。オーナはセレクションのメカニズムを利用して、指定のセレクションを指定のターゲットに変換し、自分がその INSERT_SELECTION リクエストを受け取ったセレクションの位置にそれを挿入しなければならない（そうすることがオーナの内部データ構造に対してどのようなことを意味するとしても）。

2.6.3.3 INSERT_PROPERTY

セレクションのオーナがセレクションを INSERT_PROPERTY に変換するリクエストを受け取るとき、オーナはそのリクエストで指定されているプロパティを、自分がその INSERT_SELECTION リクエストを受け取ったセレクションの位置に挿入しなければならない（そうすることがオーナの内部データ構造に対してどのようなことを意味するとしても）。

2.7 セレクションプロパティの使用

セレクションによるデータ転送で使用するプロパティの名前はリクエスタが選択する。ConvertSelection リクエストにおいて *property* フィールドに None を用いる（セレクションオーナが名前を選ぶように要求する）ことは、これらの規約では許されない。

セレクションによるデータ転送では、いつでもセレクションオーナがプロパティのタイプを選択する。タイプのなかには、規約によって割り当てられている特別な意味をもつものもある。これらを以降のセクションで説明する。

すべての場合において、ターゲットへの変換を求めるリクエストはそのターゲットについて、前述の表に記載してあるいずれかのタイプのプロパティを返すか、タイプ INCR のプロパティを返してから記載してあるいずれかのタイプのプロパティを返さなければならない。

あるセレクションプロパティは資源識別子を含んでいるかもしれない。セレクションオーナはその資源が破壊されていないこと、そしてセレクションの転送が完了するまでその内容を変更しないことを保証しなければならない。資源が存在することや、資源の内容が適正であることに依存するリクエスタは、そのセレクションプロパティを消去する前に（例えば、ピクスマップの内容をコピーすることによって）その資源を処理しなければならない。

セレクションオーナは、プロパティのタイプが示すタイプの、0 個以上の項目からなるリストを返すことになる。一般的にそのリストの項目の個数は、セレクションのばらばらの部分の個数に対応している。いくつかのターゲット（例えば、副作用をもつターゲット）は、セレクションのばらばらの部分の個数と関係なく長さ 0 である。固定長の項目の場合、リクエスタは項目の個数をプロパティのサイズで決定するかもしれない。セレクションプロパティのタイプを下記の表に記載してある。テキストのような可変長の項目についてはセパレータも記載してある。

タイプアトム	フォーマット	セパレータ
APPLE_PICT	8	自己決定的なサイズ
ATOM	32	固定長
ATOM_PAIR	32	固定長
BITMAP	32	固定長
C_STRING	8	0
COLORMAP	32	固定長
COMPOUND_TEXT	8	0
DRAWABLE	32	固定長
INCR	32	固定長
INTEGER	32	固定長
PIXEL	32	固定長
PIXMAP	32	固定長
SPAN	32	固定長
STRING	8	0
WINDOW	32	固定長

時が経つにつれてこの表は大きくなっていくと思われる。

2.7.1　テキストのプロパティ

一般的に、テキスト文字列のプロパティにおける文字のエンコーディングは、そのタイプによって指定される。文字列のプロパティのタイプと、コンソーシアムが採用しているフォント指定標準においてフォント名内部に組み込まれている文字集合名の間に、簡単で逆変換可能なマッピングがあることが強く望まれている。

アトム TEXT は多形態のターゲットである。TEXT への変換要求は、オーナに都合のよいどんなエンコーディングにも変換されることになる。その選ばれるエンコーディングは、返されるプロパティのタイプが示すことになる。TEXT はタイプとしては定義されていないので、セレクションの変換リクエストから返されるタイプにはなり得ない。

リクエスタは、ある特定のエンコーディングでセレクションの内容を返すようにオーナに希望するのであれば、そのエンコーディング名への変換を要求しなければならない。

セクション 2.6.2 の表で、（タイプ欄の）TEXT という語は、登録されているエンコーディング名のひとつを示すために使用されている。タイプは正確には TEXT ではない。つまり、タイプはオーナによって選ばれるエンコーディングを指定する、STRING や何かそのほかのアトムである。

タイプあるいはターゲットとしての STRING は、ISO Latin-1 文字集合に制御文字のタブ（8 進数で 11）と改行（8 進数で 12）を加えたものを指定する。タブのスペーシング（字間をあけること）の解釈はコンテクスト依存である。現時点では、そのほかの ASCII 制御文字は STRING には明示的に含まれない。

タイプあるいはターゲットとしての COMPOUND_TEXT は、コンパウンドテキスト相互変換フォーマットを指定する。*Compound Text Encording* を見よ。

テキストオブジェクトのなかには、その出所あるいは意図をもつユーザが、場合によっては、そのテキストについて特定の文字集合をもたないが、代わりに 0 を終端とすること以外にはいっさい制限のない、連続するバイト列を単に必要とするものもある。すなわち、セレクションのメカニズムでは、いくつかのバイト値が許されないことを想定してよい要素や、2 つの異なるバイト列が同じ意味であることを想定してよい要素はいっさいない[7]。こうしたオブジェクトに対してはタイプ C_STRING を使用しなければならない。

根本的理由

C_STRING を必要とする例はファイル名を伝えることである。多くのオペレーティングシステムは、ファイル名に文字集合があるものとしてファイル名を解釈しない。例えば、同一の文字列は ASCII と EBCDIC では異なるバイト列を使用するが、大部分のオペレーティングシステムはそれらを異なるファイル名とみなしてしまうし、それらを同じものとして扱う方法をいっさい提供しない。したがって文字集合に基づくプロパティのタイプはまったく適切でない。

タイプ STRING, COMPOUND_TEXT と C_STRING のプロパティはヌル文字で区切られる構成要素のリストから成る。そのほかのエンコーディングでは適切なリストのフォーマットを指定する必要があるだろう。

[7] このことは、STRING とは、多くのバイト値が許されないという点で異なること、そして COMPOUND_TEXT とは、例えば、バイト列 27, 40, 66（ASCII から GL への指示）を先頭に挿入してもその意味が変わらないという点で異なることに注意。

2.7.2 タイプ INCR のプロパティ

リクエスタはセレクションのデータをもたらすあらゆるターゲットの応答として、タイプ INCR[8]のプロパティを受け取るかもしれない。これはオーナが実際のデータをインクリメンタルに送信しようとしていることを示す。INCR プロパティの内容は整数であり、そのセレクションのデータのバイト数に関して、より低い境界を表している。リクエスタとセレクションオーナはセレクションのデータを次のように転送する。

セレクションのリクエスタが、そのセレクションの返答を形成している（タイプ INCR の）プロパティを消去することにより、転送処理を開始する。

続いてセレクションオーナは、

- データを適切な大きさのチャンク（塊）で、そのセレクションの返答で用いたのと同じウィンドウの同じプロパティに、変換されるそのセレクションの実際のタイプに対応するタイプで追加する。そのサイズは X サーバとの接続ハンドシェイクで受け取る最大リクエストサイズよりも小さくなければならない。
- それぞれの追加の間に、リクエスタがデータを読んだことを示す PropertyNotify（*state*==`Deleted`）イベントを待つ。このようにする理由は、サーバのスペースの消費を制限するためである。
- （サーバにデータを全部転送した後、）リクエスタがデータを読み終えたことを示す PropertyNotify（*state*==`Deleted`）イベントを待ってから、長さ 0 のデータをプロパティに書く。

セレクションのリクエスタは、

- SelectionNotify イベントを待つ。
- 次を繰り返す。
  - 引数 *delete* が `True` の GetProperty を用いてデータを取ってくる。
  - 引数 *state* が `NewValue` の PropertyNotify を待つ。
- PropertyNotify イベントが指定するプロパティの長さが 0 になるまで待つ。
- 長さ 0 のプロパティを消去する。

変換されるそのセレクションのタイプは、最初の部分のプロパティのタイプである。残りの部分のプロパティも同じタイプでなければならない。

2.7.3 ドローワブルのプロパティ

リクエスタは適当な資源識別子を含むタイプ PIXMAP, BITMAP, DRAWABLE や WINDOW のプロパティを受け取るかもしれない。これらのドローワブルに関する情報は GetGeometry リクエストという手段でサーバから手に入れることができるが、次の項目はそうではない。

[8]初期の草稿ではこれらのプロパティは INCREMENTAL と呼ばれていた。それらを使用するプロトコルが変更されたので、混乱を避けるためその名前も変更した。

- 前景ピクセル値
- 背景ピクセル値
- カラーマップ資源識別子

一般的に、セクション 2.6.2 の表の、返されるタイプがドローワブルの種類のひとつであるターゲットに変換するリクエスタは、（MULTIPLE のメカニズムを使用して）次のターゲットにも変換することを見込む必要がある。

- FOREGROUND はピクセル値（PIXEL）を返す。
- BACKGROUND はピクセル値（PIXEL）を返す。
- COLORMAP はカラーマップ資源識別子を返す。

2.7.4　スパンのプロパティ

タイプ SPAN のプロパティは、そのフォーマットによって決まるカーディナル（基数）の長さに関する、2 つのカーディナルのリストを含んでいる。1 番目は開始位置を指定し、2 番目は最終位置に 1 を加えたものを指定する。ベースは 0 である。2 つが同じである場合、スパンは長さ 0 であり、指定の位置の直前にある。その単位は LINE_NUMBER や CHARACTER_POSITION といったターゲットアトムによって暗示される。

2.8　マネージャセレクション

マネージャと呼ばれることの多い、ある種のクライアントは共有資源を管理するための責務を負う。共有資源を管理するクライアントは、セクション 1.2.3 と 1.2.6 で記述している規約を用いて指定される、適切なセレクションの所有権を獲得しなければならない。複数の共有資源（あるいは資源のグループ）を管理するクライアントは、それぞれの資源を表すセレクションの所有権を獲得しなければならない。

マネージャはそのセレクションに対して様々なターゲットへの変換をサポートしてよい。クライアントがその管理される資源と相互作用するための主要な手段として、マネージャはこのテクニックを使用するように推奨されている。ウィンドウマネージャとの相互作用に関する規約は、このセクションよりも古くからあることに注意。結果としてウィンドウマネージャとの相互作用の多くは、そのほかのテクニックを使用する。

マネージャは、マネージャセレクションの所有権を獲得する前に、GetSelectionOwner リクエストを用いて、すでに別のクライアントがそのセレクションを所有していないか確認しなければならない。そして、場合によっては、新しいマネージャが古いマネージャを置き換えるべきか、そのマネージャはユーザに尋ねなければならない。置き換えるのであれば、そのマネージャはセレクションの所有権を獲得してもよい。マネージャは、特にこの目的のために作成されるウィンドウを使用して、セレクションを獲得しなければならない。マネージャはセクション 2.1 と 2.2 で記述しているセレクションオーナのための規則に従うべきである。そして、セクション 2.6.2 に記載している必須のターゲットをサポートするべきでもある。

マネージャがセレクションの所有権を失う場合、このことは新しいマネージャがその責務を引き継ごうとしていることを意味する。古いマネージャは自分が管理していたすべての資源を解放するべきであり、続

いてそのセレクションを所有していたウィンドウを破壊するべきである。例えば、WM_S2 の所有権を失うウィンドウマネージャは、WM_S2 を所有するウィンドウを破壊する前に、SubstructureRedirect を外して、スクリーン 2 のルートウィンドウに関するイベントの懇請をやめるべきである。

新しいマネージャは、そのセレクションを所有しているウィンドウが破壊されたことに気付いたら、続けて自分が管理するつもりの資源のコントロールをうまく始めることができると判断する。古いマネージャが適正な時間内にそのウィンドウを破壊しない場合は、新しいマネージャはそのウィンドウそのものを破壊したり、古いマネージャを殺したりする前に、ユーザに確認しなければならない。

マネージャが、セレクションによってコントロールされる共有資源の管理を、自分が所有しているときに放棄したい場合は、自分が管理している資源を解放し、続けてそのセレクションを所有しているウィンドウを破壊することによって、それを実行するべきである。新しいマネージャは最初にそのセレクションを放棄してはならない。そうすることによってレース条件が生じてしまうからである。

マネージャセレクションのオーナがその対応する共有資源を管理しなくなるのはいつであるのか、ということに関心があるクライアントは、そのセレクションを所有しているウィンドウに関する StructureNotify を懇請して、そのウィンドウが破壊されるときを自分自身に通知できるようにしなければならない。クライアントは、GetSelectionOwner を実行してから StructureNotify を懇請した後に、再び GetSelectionOwner を実行して、最初にセレクションオーナを取得してから StructureNotify を懇請するまでにオーナが変わらなかったことを確認するように警告されている。

マネージャは、マネージャセレクションの所有権をうまく獲得した直後に、ClientMessage イベントを送ることによって自分の出現をアナウンスしなければならない。SendEvent プロトコルリクエストを使用して、このイベントを次の引数と共に送らなければならない。

引数	値
destination	スクリーン 0 のルートウィンドウ、またはマネージャがスクリーン特有の資源を管理しているならば、適正なスクリーンのルートウィンドウ
propagate	`False`
event-mask	StructureNotify
event	ClientMessage
type	MANAGER
format	32
data[0][9]	タイムスタンプ
data[1]	マネージャセレクションのアトム
data[2]	セレクションを所有しているウィンドウ
data[3]	マネージャセレクション特有のデータ
data[4]	マネージャセレクション特有のデータ

特定のマネージャがいつから動作しているのかを知りたいクライアントは、適当なルートウィンドウに関して StructureNotify を懇請しなければならず、そして適当なタイプ MANAGER の ClientMessage を待ち構えていなければならない。

[9] *data*[*n*] という表記法を、ClientMessage の *format* フィールドに従って、その *data* フィールドにある LISTofINT8, LISTofINT16, あるいは LISTofINT32 の *n* 番目の要素を指すために使用する。このリストは 0 から始まっている。

第4章　Client-to-Window-Manager Communication

ウィンドウマネージャが、スクリーン空間といった競合する資源の要求を調停するという自分の役割を果たせるように、（ウィンドウマネージャによって）管理されているクライアントはいくつかの規約に忠実であるべきであり、ウィンドウマネージャも同じようにすると期待するべきである。ここではクライアントの視点からこれらの規約を扱う。

一般に、これらの規約はいくぶん複雑であり、ウィンドウ管理の新しいパラダイムが開発されれば、きっと変更されることになる。このように、本質的で、ウィンドウ管理のパラダイムすべてに全般的にあてはまる規約だけを定義しようとする強い傾向がある。ある特定のウィンドウマネージャと共に動作するように設計されたクライアントは、容易にプライベートなプロトコルを定義してこれらの規約へ加えることができる。しかし、そのプライベートなプロトコルの設計者が、自分はユーザインタフェイスの「真の光」を見たとどんなに確信しているとしても、そのようなクライアントのユーザが何か別なウィンドウマネージャを動作させようとすることになるかもしれないということを、そのクライアントは承知しているべきである。

どのウィンドウマネージャが動作しているのかを、あるいはそれどころか、ウィンドウマネージャが動作しているのかどうかさえも、一般的なクライアントが知っていたり気にしたりしてはならないということが、これらの規約の原則である。ウィンドウマネージャが動作していなければ、規約はクライアントの機能のすべてをサポートするとは限らない。例えば、アイコン化の概念をクライアントは直接サポートしない。ウィンドウマネージャがいっさい動作していなければ、アイコン化の概念はあてはまらない。この規約の目標は、機能上の損失なしにウィンドウマネージャを殺し、再始動できるようにすることである。

ウィンドウマネージャはそれぞれ、独自のウィンドウ管理方針をもつことになる。ユーザの環境に適切なウィンドウ管理方針は、個々のクライアントが選択するものではなく、ユーザやユーザのシステム管理者によって選択されるものである。このことは、プライベートなプロトコルを使用して、特定のウィンドウマネージャのもとでのみ操作できるように制限されたクライアントを書く可能性を除外しない。むしろ、一般的なユーティリティがそのようなプログラムをまったく要求しないと保証しているだけである。

例えば、「私が書いているクライアントは重要であり、（画面の）一番上に位置する必要がある。」と主張されることがよくある。おそらく重要なのは、そのクライアントを本格的に動作させているときである。そして、あるプライベートなプロトコルによって「重要な」ウィンドウを認識し、そのウィンドウが一番上に位置するように保証するウィンドウマネージャのコントロールのもとで、そのクライアントを動作させなければならない。しかしながら、例えば、その「重要な」クライアントがデバッグされているのを想像してみよ。その場合は、そのウィンドウが常に一番上に位置するように保証することは、もはや適切なウィンドウ管理方針ではないし、そのほかのウィンドウ（例えば、デバッガ）が一番上に現れることができるウィンドウマネージャのもとで、そのクライアントを動作させなければならない。

4.1　クライアントの行動

一般に、X バージョン 11 の設計の目標は、クライアントはウィンドウマネージャが不在のときにするのと同じことを、できる限り正確にしなければならないということである。ただし、次のことは除く。

- 自分が取得したい資源についてウィンドウマネージャにヒントを与えること。
- 仮に要求したものでなくても、自分に割り当てられた資源を受諾することによって、ウィンドウマネージャと協力すること。
- 資源の割り当てが変化することにいつでも備えていること。

4.1.1　最上位ウィンドウの生成

クライアントの**最上位ウィンドウ**は、オーバライドリダイレクト属性が `False` のウィンドウである。最上位ウィンドウはルートウィンドウの子であるか、あるいはウィンドウマネージャによって親が変更される直前まで、ルートウィンドウの子であった。クライアントがそのウィンドウの親をルートウィンドウでないウィンドウに変更すれば、そのウィンドウはもはや最上位ウィンドウではない。しかし、クライアントがそのウィンドウの親をルートウィンドウに戻せば、そのウィンドウは再び最上位ウィンドウになることができる。

クライアントは通常、次のようなある定型文を使用することにより、自分の複数の最上位ウィンドウを、1 つ以上のルートウィンドウの子として生成しようと見込んでいる。

```
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), xsh.x, xsh.y,
                          xsh.width, xsh.height, bw, bd, bg);
```

しかしながら、ある特定のルートウィンドウのひとつを必要とする場合は、クライアントは次のようなものを使用できる。

```
win = XCreateSimpleWindow(dpy, RootWindow(dpy, screen), xsh.x, xsh.y,
                          xsh.width, xsh.height, bw, bd, bg);
```

理想的には、ルートウィンドウの選択をオーバライドすること、そして（ウィンドウマネージャを含む）クライアントがルートウィンドウでないウィンドウを擬似的なルートウィンドウとして扱うことが可能でなければならない。例えば、このことによってウィンドウマネージャのテストができるようになり、アプリケーション特有のウィンドウマネージャを使用して、関連するクライアント一式のそれぞれが所有するサブウィンドウをコントロールできるようになる。適切にそれを行うには拡張が必要であり、そのデザインは研究中である。

クライアントの視点からは、ウィンドウマネージャはクライアントの最上位ウィンドウを、次の 3 つの状態のひとつにあるものとみなすことになる。

- Normal（通常の状態）

- Iconic（アイコン化された状態）
- Withdrawn（退避した状態）

新しく生成したウィンドウは Withdrawn 状態で始まる。最上位ウィンドウをマップ、アンマップするときと、ウィンドウマネージャがあるメッセージ（ClientMessage イベント）を受け取ったときに、状態間の遷移が生じる。より詳しいことについては、セクション 4.1.2.4 と 4.1.4 を見よ。

4.1.2 クライアントのプロパティ

クライアントは、ひとたび 1 つ以上の最上位ウィンドウを所有したら、その最上位ウィンドウにプロパティを置き、自分が望む挙動をウィンドウマネージャに通知しなければならない。ウィンドウマネージャは、与えられないプロパティがあれば、どれでもそれに自分に都合のよい値をとることになる。したがって、特定の値に依存するクライアントは明示的にそうした値を与えるべきである。クライアントが書いたプロパティをウィンドウマネージャが変更することはない。

ウィンドウが Withdrawn 状態から遷移したときに、ウィンドウマネージャはこれらのプロパティの内容を調べ、そのウィンドウが Iconic 状態か Normal 状態にある間、いくつかのプロパティをその変更について監視することになる。クライアントはそれらのプロパティのひとつを変更するときに、`Replace` モードを使用してプロパティ全体を新しいデータで上書きしなければならない。つまり、ウィンドウマネージャがプロパティの昔の値を覚えたりすることはない。プロパティのフィールドはすべて、1 回の `Replace` モードの ChangeProperty リクエストで、適切な値にセットするべきである。このことは、存在しているウィンドウマネージャが墜ちたり、それをシャットダウンして再始動したり、あるいはセッションマネージャがセッションをシャットダウンして再始動する必要がある場合に、プロパティのすべての内容を、新しいウィンドウマネージャが利用できるように保証する。

規約

ウィンドウマネージャのためのプロパティを書いたり書き換えたりするクライアントは、プロパティそれぞれの完全な内容がいつでも有効であるように保証しなければならない。

これらのプロパティのいくつかは、ウィンドウやピクスマップのような資源識別子を含んでいるかもしれない。少なくともそのプロパティが置かれているウィンドウが存在する間は、クライアントはそれらの資源が存在するように保証しなければならない。

これらのプロパティが期待されているものよりも長い場合、クライアントはそのプロパティの残りの部分を無視しなければならない。これらのプロパティの拡張は X コンソーシアムに予約されている。つまり、それらをプライベートに拡張することは禁止されている。プライベートに追加された、クライアントとウィンドウマネージャの間のコミュニケーションは、独立したプロパティを使用して行わなければならない。この規則の唯一の例外はプロパティ WM_PROTOCOLS であり、これは任意の長さで、プライベートなプロトコルを表すアトムを含んでもよい（セクション 4.1.2.7 を見よ）。

続くセクションでは、クライアントがセットする必要のあるプロパティをそれぞれ順に説明する。これらはセクション 4.4 の表に要約されている。

4.1.2.1　プロパティWM_NAME

プロパティWM_NAME は、クライアントがウィンドウマネージャに、そのウィンドウと関連するところ（例えばそのウィンドウのヘッドラインバー）に表示して欲しい、解釈されない文字列である。

この文字列（およびそのほかすべての解釈されない文字列のプロパティ）に使用するエンコーディングは、プロパティのタイプによって暗示される。この目的で使用する、タイプを表すアトムはセクション 2.7.1 で記述している。

ウィンドウマネージャはこの情報を表示しようと努力するように期待されている。単に WM_NAME を無視することは歓迎される振る舞いではない。クライアントは、少なくともこの文字列の最初の部分はユーザに対して可視であるとみなせるし、その情報がユーザに対して不可視であれば、その理由はユーザがそれを不可視にするような明示的な行動をとったとみなせる。

逆に、ウィンドウマネージャがウィンドウのヘッドラインを仮にサポートするとしても、ユーザが WM_NAME の文字列を見ることができるという保証はまったくない。ユーザはヘッドラインをスクリーンの外に配置したり、そのほかのウィンドウで覆い隠したりするかもしれない。WM_NAME を、アプリケーションにとって重大な情報を表すために使用したり、タイムリーなユーザの応答を要求するような、アプリケーションの非同期な状態変化をアナウンスするために使用したりしてはならない。見込まれている用途は、同じクライアントから成る多くのインスタンスのひとつをユーザが識別できるようにすること、そして重大でない状態の情報をユーザに与えることである。

ヘッドラインバーをサポートするウィンドウマネージャでさえ、可視である WM_NAME の文字列の長さにある限界を設けることになる。したがって、この点では簡潔さが利益を生むことになる。

4.1.2.2　プロパティWM_ICON_NAME

プロパティWM_ICON_NAME は、ウィンドウがアイコン化されたときに、そのウィンドウと関連するところ（例えばアイコンのラベル）にクライアントが表示して欲しい、解釈されない文字列である。プロパティのタイプを含めて、そのほかの点では WM_NAME と類似している。明白な幾何学的理由から、WM_ICON_NAME では通常 WM_NAME よりも少ない数の文字が可視であることになる。

クライアントはこの文字列を自分のアイコンピクスマップやアイコンウィンドウに表示しようとしてはならない。正しく言えば、クライアントはウィンドウマネージャがそうすることに頼らなければならない。

4.1.2.3　プロパティWM_NORMAL_HINTS

プロパティWM_NORMAL_HINTS のタイプは WM_SIZE_HINTS である。その内容は次のようになる。

フィールド	データ型	コメント
flags	CARD32	（次の表を見よ）
pad	4 * CARD32	過去の互換性のため
min_width	INT32	なければ *base_width* であるとみなす
min_height	INT32	なければ *base_height* であるとみなす
max_width	INT32	
max_height	INT32	
width_inc	INT32	
height_inc	INT32	
min_aspect	(INT32,INT32)	
max_aspect	(INT32,INT32)	
base_width	INT32	なければ *min_width* であるとみなす
base_height	INT32	なければ *min_height* であるとみなす
win_gravity	INT32	なければ `NorthWest` であるとみなす

WM_SIZE_HINTS.*flags* のビットの定義は次のようになる。

名前	値	フィールド
`USPosition`	1	ユーザが指定した *x*, *y*
`USSize`	2	ユーザが指定した幅と高さ
`PPosition`	4	プログラムが指定した位置
`PSize`	8	プログラムが指定したサイズ
`PMinSize`	16	プログラムが指定した最小サイズ
`PMaxSize`	32	プログラムが指定した最大サイズ
`PResizeInc`	64	プログラムが指定したサイズ変更の増分
`PAspect`	128	プログラムが指定した最大と最小の縦横比
`PBaseSize`	256	プログラムが指定したベースサイズ
`PWinGravity`	512	プログラムが指定したウィンドウのグラビティ

ウィンドウのサイズと位置が（Withdrawn 状態からの遷移が生じたときに）ユーザによって指定されたことを示すためには、クライアントは `USPosition` フラグと `USSize` フラグをセットしなければならない。それによってウィンドウマネージャは、ウィンドウをどこに配置するべきか、どのようなサイズにするべきかをユーザが特別に要求したこと、そしてそれ以上の対話が不必要であることがわかる。ユーザといっさい関わり合うことなく、サイズと位置がクライアントによって指定されたことを示すためには、クライアントは `PPosition` と `PSize` をセットしなければならない。

サイズの指定は境界を除くクライアントのウィンドウの幅と高さに適用される。

win_gravity は `Unmap` を除いた、コアプロトコルの WINGRAVITY で指定される値 `NorthWest` (1), `North` (2), `NorthEast` (3), `West` (4), `Center` (5), `East` (6), `SouthWest` (7), `South` (8), `SouthEast` (9) のどれでもよい。これは、ウィンドウマネージャのフレームに対してスペースをあけるために、クライアントのウィンドウをどのように移して欲しいのか、あるいは移して欲しいかどうかを指定する。

win_gravity が `Static` であれば、ウィンドウマネージャのフレームの内側にあるクライアントのウィンドウの境界が、クライアントが Withdrawn 状態からの遷移を要求したときにそれがあった位置と、スクリーン上で同じ位置になるようにフレームが配置される。*win_gravity* の他の値は、ウィンドウの参照点を指定する。`NorthWest`, `NorthEast`, `SouthWest` と `SouthEast` では、参照点はウィンドウの指定したかどの外側（境界の端の外側）である。`North`, `South`, `East` と `West` では、参照点はウィンドウの境界の指定した端の外側の中心である。`Center` では、参照点はウィンドウの中心である。ウィンドウマネージャのフレームの参照点は、クライアントが Withdrawn 状態からの遷移を要求したときにクライアントのウィンドウの参照点があった、そのスクリーン上の位置に配置される。

min_width と *min_height* のフィールドは、クライアントが有効なまま、そのウィンドウが成りえる最小サイズを指定する。*max_width* と *max_height* のフィールドは、最大サイズを指定する。*width_inc* と *height_inc* に関連する *base_width* と *base_height* のフィールドは、指定したウィンドウの幅と高さに関する等差数列を定義し、非負の整数 *i, j* に対して、次のようになる。

$$\begin{aligned} \text{幅} &= base\_width + (i \times width\_inc) \\ \text{高さ} &= base\_height + (j \times height\_inc) \end{aligned}$$

ウィンドウマネージャは、ウィンドウのサイズをユーザに報告する際、幅と高さの代わりに *i* と *j* を用いるように推奨されている。ベースサイズ（*base_width* と *base_height*）が与えられなければ、その代わりに最小サイズを使用することになるし、逆の場合も同じである。

min_aspect と *max_aspect* フィールドは、前者が分子で後者が分母の分数であり、これらによって自分が望む縦横比の範囲をクライアントが指定できる。縦横比を守るウィンドウマネージャは、指定されたウィンドウのサイズを決定するときに、ベースサイズを考慮しなければならない。縦横比と一緒にベースサイズが与えられる場合は、その縦横比が範囲に収まるか確認する前に、ウィンドウのサイズからベースサイズを引き算しなければならない。ベースサイズが与えられない場合は、ウィンドウのサイズから何も引き算してはならない。（この目的でベースサイズの代わりに最小サイズを用いてはならない。）

4.1.2.4　プロパティWM_HINTS

プロパティWM_HINTS（そのタイプは WM_HINTS である）は、ウィンドウマネージャとコミュニケーションするために使用される。このプロパティは、ウィンドウそれ自身から利用可能なウィンドウのジオメトリ、WM_NORMAL_HINTS から利用可能なそのジオメトリに関する制約、そして WM_NAME のような個々のプロパティを必要とするさまざまな文字列以外の、ウィンドウマネージャが必要とする情報を伝える。そのプロパティの内容は次のようになる。

フィールド	データ型	コメント
flags	CARD32	（次の表を見よ）
input	CARD32	クライアントの入力モデル
initial_state	CARD32	最初にマップされたときの状態
icon_pixmap	ピクスマップ	アイコンの画像のピクスマップ
icon_window	ウィンドウ	アイコンの画像のウィンドウ
icon_x	INT32	アイコンの位置
icon_y	INT32	
icon_mask	ピクスマップ	アイコンの輪郭のマスク
window_group	ウィンドウ	グループリーダのウィンドウ資源識別子

WM_HINTS.*flags* のビットの定義は次のようになる。

名前	値	フィールド
`InputHint`	1	*input*
`StateHint`	2	*initial_state*
`IconPixmapHint`	4	*icon_pixmap*
`IconWindowHint`	8	*icon_window*
`IconPositionHint`	16	*icon_x* & *icon_y*
`IconMaskHint`	32	*icon_mask*
`WindowGroupHint`	64	*window_group*
`MessageHint`	128	（このビットは一般にはもう使われていない）
`UrgencyHint`	256	緊急

プロパティWM_HINTS なしにウィンドウがマップされた場合は、ウィンドウマネージャはプロパティWM_HINTS のすべてのフィールドに対して、都合のよい値を想定して構わない。

input フィールドは、クライアントが使用する入力フォーカスモデルを、ウィンドウマネージャに伝えるために使用する（セクション 4.1.7 を見よ）。

広域的に能動的な入力モデルと、入力なしモデルのクライアントは、*input* フィールドを `False` にセットしなければならない。受動的な入力モデルと局所的に能動的な入力モデルのクライアントは、*input* フィールドを `True` にセットしなければならない。

クライアントの視点からは、ウィンドウマネージャはクライアントの最上位ウィンドウを、次の 3 つの状態のいずれかにあるものとみなす。

- Normal（通常の状態）
- Iconic（アイコン化された状態）
- Withdrawn（退避した状態）

3 つの状態の語義は、セクション 4.1.4 で説明している。新しく生成したウィンドウは Withdrawn 状態で始まる。最上位ウィンドウをマップ、アンマップするときと、ウィンドウマネージャがあるメッセージを受け取ったときに、状態間の遷移が生じる。

initial_state フィールドの値は次の表が示すように、最上位ウィンドウを Withdrawn 状態からマップするときに、クライアントがなりたい状態を決定する。

状態	値	コメント
`NormalState`	1	ウィンドウが可視である
`IconicState`	3	アイコンが可視である

icon_pixmap フィールドは、アイコンとして使用するピクスマップを指定してよい。このピクスマップは、

- ルートウィンドウにプロパティWM_ICON_SIZE があれば、それが指定するサイズのひとつでなければならない。
- デプスが 1 でなければならない。ウィンドウマネージャはデフォルトのデータベースから、適切な背景色（ビット 0 を表す）と前景色（ビット 1 を表す）を選択することになる。もちろん、これらのデフォルトは、異なるクライアントのアイコンに対して異なる色を指定できる。

icon_mask は、*icon_pixmap* のどのピクセルをアイコンとして使用するべきか指定する。これによって、アイコンが矩形でなくなることが可能になる。

icon_window フィールドは、クライアントが自分のアイコンとして使用して欲しいウィンドウの資源識別子である。すべてではないが、ほとんどのウィンドウマネージャはアイコンウィンドウをサポートする。サポートしないものは、アイコンのように振る舞う小さなウィンドウがまったく不適切であるようなユーザインタフェイスをもっているのだろう。クライアントはなんとかしてこの機能の省略を補償しようとしてはならない。

単純な 2 色のビットマップよりも多くの能力をアイコンに必要とするクライアントは、アイコンウィンドウを用いるべきである。アイコンウィンドウを用いるクライアントのための規則は、セクション 4.1.9 で説明している。

座標 (*icon_x*, *icon_y*) は、アイコンを配置する場所に関するウィンドウマネージャへのヒントである。アイコンの配置はウィンドウマネージャの方針によってコントロールされるので、クライアントはウィンドウマネージャがこのヒントに注意を払うことに依存してはならない。

window_group フィールドは、このウィンドウがウィンドウグループに所属していることをクライアントに指定させる。ひとつの例は、クライアントが単独でルートウィンドウの複数の子を操作することである。

規約

1. *window_group* フィールドはグループリーダの資源識別子にセットされなければならない。ウィンドウグループのグループリーダはその目的だけのために存在するウィンドウであってもよい。この種のプレースホルダとしてのグループリーダをクライアントやウィンドウマネージャがマップすることは決してない。
2. ウィンドウグループのグループリーダのプロパティは、そのグループ全体のためのものである（例えば、グループ全体をアイコン化するときに表示されるアイコンなど）。

ウィンドウマネージャはグループ全体を操作するための機能を提供するかもしれない。現時点では、クライアントがグループ全体を操作する方法はない。

`MessageHint` のビットは、クライアントがそれを *flags* フィールドにセットしていれば、セクション 4.1.2.7 の WM_PROTOCOLS のメカニズムではなく、時代遅れのウィンドウマネージャ通信プロトコル[1]を使用していることを示す。

`UrgencyHint` フラグは、クライアントがそれを *flags* フィールドにセットしていれば、そのウィンドウの内容が緊急なものであると考えていて、ユーザのタイムリーな応答を要求していることを示す。ウィンドウマネージャは、このフラグがセットされている間、そのウィンドウに対してユーザの注意を引くために、何か努力しようとするべきである。ウィンドウマネージャはまた、ウィンドウが Normal 状態か Iconic 状態にある間は常に、このフラグの状態を監視するべきであり、フラグの状態が変化したときに適切な行動をとるべきである。そのほかの点では、このフラグはウィンドウの状態から独立している。特に、クライアントがこのフラグをアイコン化されたウィンドウにセットした場合に、ウィンドウマネージャはそのウィンドウを元に戻す必要はない。クライアントは、ユーザが `UrgencyHint` フラグを 0 にセットできたり、そのウィンドウを Withdrawn 状態にできるような、いくつかの手段を提供するべきである。ユーザの行動が、そのウィンドウを緊急ものにした現状を緩和できたり、あるいは単にその警告を消したりできる。

根本的理由

このメカニズムは警告のダイアログボックスや備忘録のウィンドウで、ウィンドウをマップするだけでは不十分な場合（例えば、複数のワークスペースや仮想デスクトップをもつウィンドウマネージャが存在する場合）や、オーバライドリダイレクト属性をセットしたウィンドウではあまりにも押しつけがましい場合に有用である。例えば、ウィンドウマネージャは、緊急なウィンドウのタイトルバーやアイコンに標識を加えることによって、そのウィンドウに注意を引きつけてよい。ウィンドウマネージャは新しく緊急になったウィンドウに対して、（それが Iconic 状態であれば）そのアイコンをフラッシュさせたり、そのウィンドウをスタックの一番上にライズするなどして、補足的な行動をとってもよい。

[1] この時代遅れのプロトコルは 1998 年 7 月 27 日に記述された、ICCCM の草稿である。そのプロトコルを使用しているウィンドウのプロパティ WM_HINTS は期待しているものより 4 バイト長いので、そうしたウィンドウを検出することもできる。ウィンドウマネージャは過去の互換性モードで、時代遅れのプロトコルを使用するクライアントをサポートして構わない。

4.1.2.5 プロパティWM_CLASS

（制御文字を除いたタイプ STRING の）プロパティWM_CLASS は、2 つの連続した、ヌルを終端とする文字列を含む。これらはインスタンス名とクラス名を指定する。クライアントとウィンドウマネージャの両方が、アプリケーション用のリソースを探し出すために使用したり、識別情報として使用したりする。このプロパティはウィンドウが Withdrawn 状態のままであるときに存在していなければならず、ウィンドウが Withdrawn 状態にある間だけ変更することができる。ウィンドウマネージャは、自分が起動するときと、ウィンドウが Withdrawn 状態にあるときに限り、そのプロパティを調べてもよい。しかし、クライアントが自分の状態を動的に変更する必要性はまったくないはずである。

2 つの文字列は、それぞれ、

- このウィンドウを所有するクライアントが所属しているアプリケーションに関する、特有のインスタンス名を指定する文字列である。インスタンス名で指定されるリソースは、クラス名で指定されるいかなるリソースもオーバライドする。オペレーティングシステム特有の方法で、ユーザがインスタンス名を指定できる。POSIX 準拠のシステムでは、次の規約が使用される。
  - コマンド行で "`-name` *NAME*" が与えられていれば、*NAME* をインスタンス名として使用する。
  - そうでなければ、環境変数 RESOURCE_NAME がセットされていれば、その値をインスタンス名として使用する。
  - そうでなければ、そのプログラムを起動するために使用された名前の最後の部分（ディレクトリ名をすべて取り除いた `argv[0]`）をインスタンス名として使用する。
- このウィンドウを所有するクライアントが所属しているアプリケーションの、一般的なクラス名を指定する文字列である。クラス名で指定されるリソースは、同じクラス名をもつすべてのアプリケーションに適用される。クラス名はアプリケーション制作者が指定する。一般に使用されるクラス名の例には、"Emacs", "XTerm", "XClock", "XLoad" などがある。

WM_CLASS の文字列はヌルを終端とするので、STRING プロパティがヌルで区切られるという一般的な規約と異なっていることに注意。この不一致は過去の互換性のために必要である。

4.1.2.6 プロパティWM_TRANSIENT_FOR

（タイプ WINDOW の）プロパティWM_TRANSIENT_FOR は、もうひとつの最上位ウィンドウの資源識別子を含んでいる。それはこのウィンドウが、指定されたウィンドウを代表するポップアップである、という意味をもつ。ウィンドウマネージャは一時的なウィンドウを装飾しないことにしてもよいし、別の方法で異なった扱いをしてもよい。特に、仮に通常は最上位ウィンドウをマップする際にユーザとの対話が必要であるとしても、ウィンドウマネージャは WM_TRANSIENT_FOR をもつ新しくマップされたウィンドウを、そうした対話をいっさい要求することなく提示しなければならない。例えば、ダイアログボックスは WM_TRANSIENT_FOR をセットしなければならないウィンドウの例である。

WM_TRANSIENT_FOR をオーバライドリダイレクト属性と取り違えないようにすることが重要である。ウィンドウをマップしている間にポインタをグラブしない場合に（言い換えれば、一時的なウィンドウが現れている間、そのほかのウィンドウが動作できる場合に）、WM_TRANSIENT_FOR を使用しなければならない。そのほかのウィンドウが入力を処理できないようにするべきである場合は（例えば、ポップアップメニューを実装する場合は）、オーバライドリダイレクト属性を使用して、ウィンドウをマップしている間はポインタをグラブすること。

4.1.2.7　プロパティWM_PROTOCOLS

（タイプ ATOM の）プロパティWM_PROTOCOLS はアトムのリストである。それぞれのアトムは、クライアントとウィンドウマネージャの間の、クライアントが関与したい通信プロトコルを決定する。アトムは標準のプロトコルと、個々のウィンドウマネージャに特有のプライベートなプロトコルの両方を決定できる。

クライアントが自発的に関与しようと申し出ることができるプロトコルはすべて、ウィンドウマネージャがクライアントに ClientMessage イベントを送ること、そしてクライアントが適切な行動をとることを必要とする。そのイベントの内容の詳細についてはセクション 4.2.8 を見よ。どの場合でも、ウィンドウマネージャがプロトコルのトランザクションを開始する。

プロパティWM_PROTOCOLS は必須ではない。それが存在しなければ、クライアントはどのウィンドウマネージャプロトコルにも関与したいと望んでいない。

X コンソーシアムは、名前空間の衝突を避けるために、プロトコルのレジストリ（登録簿）を維持している。次の表に現在までに定義されたプロトコルを記載する。

プロトコル	セクション	目的
WM_TAKE_FOCUS	4.1.7	入力フォーカスの割り当て
WM_SAVE_YOURSELF	Appendix C	クライアントの状態リクエストの保存（使用をやめること）
WM_DELETE_WINDOW	4.2.8.1	最上位ウィンドウを消去する要求

時が経つにつれてこの表は大きくなっていくと思われる。

4.1.2.8　プロパティWM_COLORMAP_WINDOWS

最上位ウィンドウの（タイプ WINDOW の）プロパティWM_COLORMAP_WINDOWS は、最上位ウィンドウのカラーマップとは異なるカラーマップをインストールする必要のある、ウィンドウの資源識別子のリストである。ウィンドウマネージャはこのウィンドウのリストを、カラーマップ属性の変化について監視する。最上位ウィンドウは（暗黙あるいは明示的に）常に監視リストにある。このメカニズムの詳細については、セクション 4.1.8 を見よ。

4.1.2.9　プロパティWM_CLIENT_MACHINE

クライアントは（タイプ TEXT のひとつである）プロパティWM_CLIENT_MACHINE を、サーバが動作している計算機からみた、クライアントが動作している計算機の名前を構成する文字列にセットしなければならない。

4.1.3 ウィンドウマネージャのプロパティ

前のセクションで説明したプロパティは、クライアントが自分の最上位ウィンドウで維持する責任をもつものである。このセクションでは、ウィンドウマネージャがクライアントの最上位ウィンドウとルートウィンドウに置くプロパティを説明する。

4.1.3.1 プロパティWM_STATE

ウィンドウマネージャは、（タイプ WM_STATE の）プロパティWM_STATE を、Withdrawn 状態にない最上位ウィンドウにそれぞれ置く。Withdrawn 状態にある最上位ウィンドウには、プロパティWM_STATE があってもなくてもよい。ひとたび最上位ウィンドウが Withdrawn 状態になったら、クライアントは別の目的でそれを再利用してよい。そのように再利用するクライアントは、依然としてプロパティWM_STATE が存在する場合は、それを消去しなければならない。

クライアントのなかには（**xprop** のように）、そのプログラムが操作しようとするウィンドウの上でユーザがクリックするように求めるものもある。典型的に、そのウィンドウが最上位ウィンドウであるというのが、その意図である。最上位ウィンドウを見つけるために、クライアントは選択された位置の真下のウィンドウ階層を、プロパティWM_STATE をもつウィンドウを求めて検索しなければならない。ウィンドウマネージャが親を変更する可能性をすべて考慮するために、この検索は再帰的でなければならない。プロパティWM_STATE をもつウィンドウがまったく見つからない場合は、選択された位置の真下にマップされているルートウィンドウの子があれば、プログラムはそれを使用するように推奨されている。

プロパティWM_STATE の内容は次のようになる。

フィールド	データ型	コメント
state	CARD32	（次の表を見よ）
icon	ウィンドウ	アイコンウィンドウの資源識別子

WM_STATE.*state* の値を次の表に記載する。

状態	値
`WithdrawnState`	0
`NormalState`	1
`IconicState`	3

このプロパティにそのほかのフィールドを加えることは X コンソーシアムに予約されている。上記の表で定義される値以外の *state* フィールドの値は、X コンソーシアムに予約されている。

state フィールドは、そのウィンドウの状態に関するウィンドウマネージャの考えを記述していて、それはプロパティWM_HINTS の *initial_state* フィールドで表現したクライアントの考えと合わないかもしれない（例えば、ユーザがウィンドウマネージャにウィンドウをアイコン化するように求めた場合）。*state* フィールドが `NormalState` であれば、クライアントが自分のウィンドウを動作させていると、ウィンドウマネージャは信じている。それが `IconicState` であれば、クライアントは自分のアイコンウィンドウを動作させなければならない。どちらの状態でも、クライアントは両方のウィンドウの Exposure イベントを処理する用意ができていなければならない。

ウィンドウが Withdrawn 状態になるとき、ウィンドウマネージャはそのウィンドウの *state* フィールドの値を `WithdrawnState` に変更するか、あるいは完全にプロパティWM_STATE を消去する。

icon フィールドは、このプロパティがセットされたウィンドウを表すアイコンとしてウィンドウマネージャが使用する、ウィンドウの資源識別子を含んでいなければならない。そのようなウィンドウがいっさい存在しない場合は、*icon* フィールドを None にセットしなければならない。このウィンドウは、クライアントが自分のプロパティWM_HINTS で指定したアイコンウィンドウと同じウィンドウであってもよいが、必ずしもそうなるとは限らない。WM_STATE のアイコンは、ウィンドウマネージャが与えたクライアントのアイコンピクスマップを含むウィンドウであってもよいし、クライアントのアイコンウィンドウのアンセスタであってもよい。

4.1.3.2　プロパティWM_ICON_SIZE

アイコンピクスマップやアイコンウィンドウのサイズを制約したいウィンドウマネージャは、WM_ICON_SIZE と呼ばれるプロパティをルートウィンドウに置かなければならない。このプロパティの内容を次の表に記載する。

フィールド	データ型	コメント
min_width	CARD32	一連のアイコンのサイズを表すデータ
min_height	CARD32	
max_width	CARD32	
max_height	CARD32	
width_inc	CARD32	
height_inc	CARD32	

より詳しいことについては、*Xlib — C Language X Interface* のセクション 14.1.12 を見よ。

4.1.4　ウィンドウの状態の変更

クライアントの視点からみると、ウィンドウマネージャはクライアントの最上位ウィンドウをそれぞれ、3つの状態のひとつにあるとみなす。それらの状態の意味は次のようになる。

- `NormalState`（Normal 状態）— クライアントの最上位ウィンドウは可視である。
- `IconicState`（Iconic 状態）— クライアントの最上位ウィンドウはアイコン化されている（そのことがこのウィンドウマネージャにとって何を意味しようとも）。クライアントは、自分の最上位ウィンドウが不可視であり、（もしあれば）自分のアイコンウィンドウが可視であり、それができなければ、（もしあれば）自分のアイコンピクスマップあるいは自分の WM_ICON_NAME が表示されているとみなすことができる。
- `WithdrawnState`（Withdrawn 状態）— クライアントの最上位ウィンドウとそのアイコンの両方が不可視である。

実際には、ウィンドウマネージャは上記以外の意味をもつ状態を実装してよい。例えば、ウィンドウマネージャは、あまり使用されないクライアントのウィンドウをメニューの文字列として表現するような、「Inactive」状態という概念を実装してもよい。しかし、この状態はクライアントにとって不可視であり、その結果クライアントは自分自身を単に Iconic 状態にあるものとしてみなすことになるだろう。

新しく生成された最上位ウィンドウは Withdrawn 状態にある。ひとたびそのウィンドウに適正なプロパティを与えてしまえば、クライアントは次のように、その状態を自由に変更できる。

- Withdrawn → Normal — クライアントは WM_HINTS.*initial_state* を `NormalState` にしてウィンドウをマップしなければならない。
- Withdrawn → Iconic — クライアントは WM_HINTS.*initial_state* を `IconicState` にしてウィンドウをマップしなければならない。
- Normal → Iconic — クライアントはこのセクションの後の方で説明するように、ClientMessage イベントを送らなければならない
- Normal → Withdrawn — クライアントはウィンドウをアンマップし、このセクションの後の方で説明するように、合成の UnmapNotify イベントでそれをフォローしなければならない。
- Iconic → Normal — クライアントはウィンドウをマップしなければならない。WM_HINTS.*initial_state* の内容は、この場合関係がない。
- Iconic → Withdrawn — クライアントはウィンドウをアンマップし、このセクションの後の方で説明するように、合成の UnmapNotify イベントでそれをフォローしなければならない。

クライアントは Withdrawn 状態への遷移か、Withdrawn 状態からの遷移をもたらすことしかできない。ひとたびクライアントのウィンドウが Withdrawn 状態から離れてしまうと、そのウィンドウは Normal 状態であればマップされるし、Iconic 状態であればアンマップされる。親を変更するウィンドウマネージャは、クライアントのウィンドウが Iconic 状態であるときは、仮にアンセスタのウィンドウがマップされていないためにクライアントのウィンドウが不可視になっているとしても、それをアンマップするべきである。反対に、親を変更するウィンドウマネージャがアンセスタをアンマップすることによって、クライアントのウィンドウを不可視にしている場合は、そのクライアントのウィンドウは定義により Iconic 状態にあり、またアンマップされるべきでもある。

実装者へのアドバイス

クライアントは、StructureNotify で自分の最上位ウィンドウに関するイベントを懇請して、Normal 状態と Iconic 状態の間の遷移をたどることができる。MapNotify イベントの受取は Normal 状態への遷移を示し、UnmapNotify イベントの受取は Iconic 状態への遷移を示す。

ウィンドウの状態を Withdrawn 状態に変更するとき、クライアントは（ウィンドウをアンマップすることに加えて）次の引数をもつ SendEvent リクエストを使用することによって、合成の UnmapNotify イベントを送るべきである。

引数	**値**
destination	ルートウィンドウ
propagate	`False`
event-mask	(SubstructureRedirect|SubstructureNotify)
event	UnmapNotify
event	ルートウィンドウ
window	ウィンドウそれ自身
from-configure	`False`

根本的理由

クライアントに合成の UnmapNotify イベントを送るように要求する理由は、クライアントが状態を変更しようと望んだという通知を、仮にその希望を示すときにはすでにウィンドウがアンマップされていたとしても、ウィンドウマネージャが得るように保証するためである。

実装者へのアドバイス

時代遅れのクライアントとの互換性のために、ウィンドウマネージャは合成の UnmapNotify を待つというよりはむしろ、本物の UnmapNotify に関して Withdrawn 状態への遷移を誘発しなければならない。ウィンドウマネージャはまた、まだ本物の UnmapNotify を受け取っていないウィンドウに関する合成の UnmapNotify を受け取る場合も、その遷移を誘発しなければならない。

クライアントがウィンドウを Withdrawn 状態にするとき、ウィンドウマネージャはセクション 4.1.3.1 の説明のように、そのプロパティ WM_STATE を更新あるいは消去する。クライアントのウィンドウを Withdrawn 状態にした後、それを（例えば、再びそれをマップしたり、その親をどこか違うところに変更したりすることによって）再利用したいクライアントは、処理に進む前に、それが完全に Withdrawn 状態になるまで待つべきである。これを行うための好ましい方法は、ウィンドウマネージャがプロパティ WM_STATE を更新あるいは消去するまで、クライアントが待つことである[2]。

Normal 状態から Iconic 状態へ遷移する場合、クライアントは ClientMessage イベントを次の引数と共に、ルートウィンドウへ送らなければならない。

引数	値
window	アイコン化するウィンドウ
type[3]	アトム WM_CHANGE_STATE
format	32
data[0]	`IconicState`

根本的理由

この ClientMessage イベントのフォーマットは、セクション 4.2.8 での ClientMessage のフォーマットと適合しない。これは、それらはウィンドウマネージャがクライアントに送るものであり、このメッセージはクライアントがウィンドウマネージャに送るものだからである。

data[0] のそのほかの値は、これらの規約への将来の拡張のために予約されている。その SendEvent リクエストのパラメータは、合成の UnmapNotify イベントを記述するものでなければならない。

[2] これらの規約の初期の版では、クライアントがプロパティ WM_STATE を読み出すことを禁止していた。初期の規約に従って動作するクライアントは、ReparentNotify イベントをたどるテクニックを使用して、最上位ウィンドウの親がルートウィンドウに戻るまで待った。これは依然として有効なテクニックである。しかしながら、これは親を変更するウィンドウマネージャに対してしか有効でないので、WM_STATE によるテクニックが望ましい。

[3] ClientMessage イベントの *type* フィールド（Xlib では *message_type* フィールドと呼ばれる）を、イベントそれ自身のコードを表すフィールドと混同してはならない。コードを表すフィールドは値 33（ClientMessage）をもつ。

実装者へのアドバイス

クライアントは VisibilityChange で自分の最上位ウィンドウやアイコンウィンドウに関するイベントを懇請することもできる。そうすると、問題となっているウィンドウをマップしていたとしても、それが完全に隠されたとき（したがって、おそらく更新するのは時間のむだである）、クライアントは VisibilityNotify（*state*==`FullyObscured`）イベントを受け取り、そしてそれが部分的にでも可視になるとき、クライアントは VisibilityNotify（*state*!=`FullyObscured`）イベントを受け取る。

実装者へのアドバイス

ウィンドウが Normal 状態から、Iconic 状態か Withdrawn 状態へ遷移したとき、ウィンドウマネージャがそのウィンドウのための一時的ウィンドウを利用できなくするかもしれないことを、クライアントは承知していなければならない。クライアントは、一時的ウィンドウを所有するウィンドウが Normal 状態ではないとき、ユーザがその一時的ウィンドウを利用できることに依存してはならない。ウィンドウを Withdrawn 状態にするとき、クライアントはそのウィンドウの一時的ウィンドウも Withdrawn 状態にするように勧告されている。

4.1.5 ウィンドウの配置構成

クライアントは自分の最上位ウィンドウのサイズや位置を、ConfigureWindow リクエストを使用して変更することができる。このリクエストで変更可能なウィンドウの属性は次のようになる。

- ウィンドウの左上隅の外側の位置 [*x*, *y*]
- ウィンドウの内側の領域（境界を除く）の幅と高さ
- ウィンドウの境界の幅
- スタックにおけるウィンドウの位置

位置を表す座標系は（親の変更が生じていたとしても、それにはいっさい関係なく）ルートウィンドウの座標系で表される。使用する境界の幅と、使用する *win_gravity* による位置のヒントは、クライアントがリクエストした最新のものである。クライアントによる配置構成のリクエストは、セクション 4.1.2.3 の説明のように、Withdrawn 状態からマップされるウィンドウの初期ジオメトリと同じように、ウィンドウマネージャによって解釈される。その要求したサイズや位置にウィンドウマネージャがウィンドウを割り当てる保証はいっさいないということを、クライアントは承知しているべきであり、どのようなサイズや位置でも扱えるように用意しておくべきである。ウィンドウマネージャが ConfigureWindow リクエストに応えると決めた場合は、

- ウィンドウのサイズ、位置、境界の幅や、スタックの順序をいっさい変更しない。

  クライアントは、ウィンドウの（変更されなかった）ジオメトリを記述する、合成の ConfigureNotify イベントを受け取る。座標 (*x*, *y*) は、親が変更されていたとしてもいっさい関係なく、ルートウィンドウの座標系にあり、クライアントが要求した境界の幅に対して調整されてある。*border_width* はクライアントが要求した境界の幅である。実際にはまったく変更が生じなかったので、クライアントは本物の ConfigureNotify を受け取らない。

- サイズや境界の幅を変更しないで、ウィンドウを移動したりそのスタックの順序を変更したりする。

  クライアントは、変更に伴ってウィンドウの新しいジオメトリを記述する、合成の ConfigureNotify イベントを受け取る。イベントの座標 (*x*, *y*) はルートウィンドウの座標系にあり、クライアントが要求した境界の幅に対して調整されてある。*border_width* はクライアントが要求した境界の幅である。ウィンドウマネージャが最上位ウィンドウの親を変更してしまったかもしれないので、クライアントはこの変更を記述する本物の ConfigureNotify を受け取らないかもしれない。クライアントが本物のイベントを受け取る場合は、合成のイベントが本物のイベントのあとに続く。

- ウィンドウのサイズを変更したり、その境界の幅を変更したりする（ウィンドウの位置やそのスタックの順序も変更したかどうかに関係なく）。

  StructureNotify でイベントを懇請したクライアントは、本物の ConfigureNotify イベントを受け取る。このイベントの座標は親ウィンドウに対して相対的であり、そのウィンドウの親が変更されていた場合は、その親ウィンドウはルートウィンドウではないかもしれないことに注意。座標は、ウィンドウの（ウィンドウマネージャが変更したかもしれない）実際の境界の幅を反映している。必要であれば、座標を変換するために TranslateCoordinates リクエストを使用することができる。

本物の ConfigureNotify イベントの座標が親ウィンドウの空間にあるのに対して、合成のイベントでは座標がルートウィンドウの空間にあるというのが、一般的な規則である。

実装者へのアドバイス

クライアントが、最上位ウィンドウのサイズと位置が変更されたケースを、そのサイズは変更されたが位置は変更されなかったケースと区別できないのは、両方のケースで本物の ConfigureNotify イベントを受け取るためである。最上位ウィンドウの絶対的な位置をたどり続けることを問題にするクライアントは、自分がその位置を確信しているどうかを示す状態を 1 つ保持しなければならない。最上位ウィンドウに関する本物の ConfigureNotify イベントを受け取ったとき、クライアントはその位置が不明であると記録しなければならない。合成の ConfigureNotify イベントを受け取ったとき、クライアントはこのイベントにある位置を使用して、その位置が既知であると記録しなければならない。クライアントがそのウィンドウ関する（あるいはあらゆるインフェリアに関する）KeyPress, KeyRelease, ButtonPress, ButtonRelease, MotionNotify, EnterNotify や LeaveNotify イベントを受け取った場合は、これらのイベントに含まれる座標 (*event-x*, *event-y*) と座標 (*root-x*, *root-y*) の差異から、クライアントは最上位ウィンドウの位置を導き出すことができる。その位置が不明な場合だけ、クライアントは TranslateCoordinates リクエストを使用して最上位ウィンドウの位置を見つける必要がある。

クライアントは自分の境界が可視ではないかもしれないことを承知していなければならない。ウィンドウマネージャは親を変更するテクニックを使用して、タイトル、画面上のさまざまな GUI オブジェクトと、そのほかの首尾一貫した look-and-feel を維持する項目を含む境界を用いて、クライアントの最上位ウィンドウを自由に装飾できる。ウィンドウマネージャがそうする場合は、クライアントがセットしようとした境界の幅をオーバライドして、それを 0 にセットすることが確実である。したがって、クライアントは最上位ウィンドウの境界が可視であることに依存してはならないし、決定的に重要な情報を表示するためにそれを使用してはならない。そのほかのウィンドウマネージャでは、最上位ウィンドウの境界は可視となる。

規約

クライアントはレース条件を避けるために、すべての ConfigureWindow リクエストに関する境界の幅の属性に、希望する値をセットしなくてはならない。

スタックにおける自分の位置を変更するクライアントは、自分の親が変更されたかもしれないことを承知しているべきである。それはかつて兄弟であったウィンドウが、もはやそうではないということを意味する。兄弟でないウィンドウを ConfigureWindow リクエストの *sibling* フィールドとして使用すると、エラーが生じることになる。

規約

ConfigureWindow リクエストを用いて、スタックにおける自分の位置を変更するように要求するクライアントは、*sibling* フィールドに `None` を用いて、そうしなければならない。

もともとは兄弟だったウィンドウに対して相対的に、スタックにおける自分の位置を変更しなければならないクライアントは、（クライアントが親を変更しないウィンドウマネージャのもとで動作している場合のために）ConfigureWindow リクエストを発行して、結果的に生じるエラーを扱えるように用意しておき、そして次の引数をもつ SendEvent リクエストを行使して、合成の ConfigureRequest イベントでフォローするべきである。

引数	値
destination	ルートウィンドウ
propagate	`False`
event-mask	(SubstructureRedirect|StructureNotify)
event	ConfigureRequest
event	ルートウィンドウ
window	ウィンドウそれ自身
...	ConfigureWindow リクエストからの残りのパラメータ

どのような場合でもウィンドウマネージャは、自分がふさわしいと思うように、ウィンドウのスタックにおける位置を変更して構わないので、クライアントは自分が要求したスタックの順序を受け取ることに依存してはならない。自分の最上位ウィンドウに届く本物と合成の ConfigureNotify イベントの両方に関する、上記の *sibling* フィールドは有用な情報を含んでいないかもしれないので、クライアントはこのフィールドを無視しなければならない。

4.1.6 ウィンドウの属性の変更

ウィンドウを生成するときに与えられる属性は、ChangeWindowAttributes リクエストを用いて変更できる。ウィンドウの属性を次の表に記載する。

属性	クライアントにプライベート
背景ピクスマップ	Yes
背景ピクセル値	Yes
境界のピクスマップ	Yes
境界のピクセル値	Yes
ビットグラビティ	Yes
ウィンドウグラビティ	No
バッキングストアのヒント	Yes
セーブアンダのヒント	No
イベントマスク	No
伝播させないイベントのマスク	Yes
オーバライドリダイレクトのフラグ	No
カラーマップ	Yes
カーソル	Yes

ほとんどの属性はクライアントに対してプライベートであり、ウィンドウマネージャが干渉することはいっさいない。クライアントに対してプライベートでない属性については、

- ウィンドウマネージャはウィンドウグラビティを自由にオーバライドできる。親を変更するウィンドウマネージャは最上位ウィンドウのウィンドウグラビティを、自分自身の目的のためにセットしようと望むかもしれない。

- クライアントは自分の最上位ウィンドウに関するセーブアンダのヒントをセットして構わない。しかし、ウィンドウマネージャがそのヒントをオーバライドするかもしれないことを、クライアントは承知しているべきである。

- 基本的には、ウィンドウはクライアントごとにイベントマスクをもつので、ウィンドウマネージャが懇請しているイベントにいっさい関係なく、クライアントは都合のよいどんなイベントを懇請してもよい。あるときただひとつのクライアントだけが懇請してもよいイベントもあるが、ウィンドウマネージャはどんなクライアントのウィンドウに関しても、そうしたイベントを懇請してはならない。

- クライアントは最上位ウィンドウにオーバライドリダイレクト属性をセットできるが、セクション 4.1.10 と 4.2.9 で記述しているような場合を除き、そうしないように推奨されている。

4.1.7　入力フォーカス

入力処理には、次の 4 つのフォーカスモデルがある。

- 入力なし — クライアントはキーボード入力をいっさい要求しない。**xload** やその他の出力だけのクライアントがその例である。

- 受動的な入力 — クライアントはキーボード入力を要求するが、入力フォーカスを明示的にセットすることをいっさいしない。サブウィンドウをもたない簡単なクライアントがその例で、`PointerRoot` モードか、ウィンドウマネージャが自分の最上位ウィンドウに入力フォーカスをセットしたとき（click-to-type モードのとき）に入力を受け付ける。

- 局所的に能動的な入力 — クライアントはキーボード入力を要求し、入力フォーカスを明示的にセットするが、それを行うのは自分に所属するウィンドウのひとつが既にフォーカスを取得しているときだけである。さまざまなデータを入力する領域となるサブウィンドウを複数もち、Next, Prev キーを使って入力フォーカスをそれらの領域間で移動できるクライアントがその例である。それを行うのは、自分の最上位ウィンドウが `PointerRoot` モードでフォーカスを獲得したときか、ウィンドウマネージャが自分の最上位ウィンドウに入力フォーカスをセットしたとき（click-to-type モードのとき）である。

- 広域的に能動的な入力 — クライアントが所有していないウィンドウに入力フォーカスがあっても、クライアントはキーボード入力を要求し、入力フォーカスを明示的にセットする。スクロールバーをもち、入力フォーカスが何かそのほかのウィンドウにあっても入力フォーカスを攪乱することなく、ユーザがウィンドウをスクロールできるようにしたいクライアントがその例である。入力フォーカスを獲得したいと望むときは、スクロールする領域をユーザがクリックしたときであり、スクロールバーそれ自身をクリックしたときではない。したがってこのクライアントは、自分に所属するウィンドウのどれにもウィンドウマネージャが入力フォーカスをセットしないように期待する。

4つの入力モデルと、*input* フィールドの対応する値と、プロパティWM_PROTOCOLS にアトム WM_TAKE_FOCUS があるかないかを、次の表に載せた。

入力モデル	***input* フィールド**	WM_TAKE_FOCUS
入力なし	`False`	なし
受動的な入力	`True`	なし
局所的に能動的な入力	`True`	あり
広域的に能動的な入力	`False`	あり

受動的な入力と局所的に能動的な入力のクライアントは、WM_HINTS の *input* フィールドを `True` にセットし、それによって、クライアントが入力フォーカスを獲得する際にウィンドウマネージャの支援を必要とすることを示す。入力なしと広域的に能動的な入力のクライアントは、*input* フィールドを `False` にセットし、ウィンドウマネージャがクライアントの最上位ウィンドウに入力フォーカスをセットしないように要求する。

SetInputFocus リクエストを用いるクライアントは *time* フィールドを、そうしようとする原因となったイベントのタイプスタンプにセットするべきである。FocusIn イベントはタイムスタンプをもたないので、FocusIn イベントにはその資格がない。クライアントは対応する EnterNotify なしにフォーカスを獲得してもよい。クライアントは *time* フィールドに `CurrentTime` を使用するべきではないことに注意。

広域的に能動的な入力モデルを使用するクライアントは、次のイベントのひとつを受け取った際に入力フォーカスをまだもっていないときに限り、SetInputFocus リクエストを用いて入力フォーカスを獲得することができる。

- ButtonPress
- ButtonRelease
- 受動的なグラブによる KeyPress
- 受動的なグラブによる KeyRelease

一般にクライアントは、受動的なグラブによるキーボードイベントを、（例えばカット、コピーやペーストするキーに関する受動的なグラブを確立するセレクションのツールのように）避けられない場合を除き、この目的のために使用することを避けなければならない。

ユーザがウィンドウマネージャにフォーカスをセットするように命令する方法は、そのウィンドウマネージャしだいである。例えば、クライアントはフォーカスを移すクリックを自分が認識するかどうか決定することができない。

プロパティWM_PROTOCOLS にアトム WM_TAKE_FOCUS があるウィンドウは、（セクション 4.2.8 の説明のように）ウィンドウマネージャから、*data*[0] フィールドに WM_TAKE_FOCUS をもち、*data*[1] フィールドに有効なタイムスタンプ（すなわち、`CurrentTime` ではない）をもつ ClientMessage イベントを受け取るかもしれない。クライアントはフォーカスが欲しい場合、SetInputFocus リクエストで応えなければならない。その際、そのリクエストの *window* フィールドを、入力フォーカスを最後にもっていた自分のウィンドウ、あるいは自分のデフォルトの入力ウィンドウにセットし、*time* フィールドをそのメッセージ（ClientMessage イベント）に含まれるタイムスタンプにセットする。さらに多くの情報については、セクション 4.2.7 を見よ。

クライアントは、アイコンから開始する場合や、ユーザが最上位ウィンドウの外側の、ウィンドウマネージャにフォーカスを割り当てるように指示する領域をクリックした場合にも、WM_TAKE_FOCUS を受け取るかもしれない。（例えば、ヘッドラインバーへのクリックを利用してフォーカスを割り当てることが可能である）。

目標は、最上位ウィンドウが入力フォーカスを自分のサブウィンドウのひとつに割り当てることや、フォーカスの申し入れを断ることができるような方法で、入力フォーカスを最上位ウィンドウに割り当てたいウィンドウマネージャを支援することである。アイコンから復帰した後や、ライズした後に、クライアントは入力フォーカスを受け入れると、仮にウィンドウマネージャが概して信じているとしても、例えば、時計や、現在まったくフレームをオープンしていないテキストエディタは、フォーカスを受け入れたくないだろう。

入力フォーカスをセットするクライアントは、SetInputFocus リクエストの *revert-to* フィールドの値を決定する必要がある。この値は、フォーカスをセットしたウィンドウが不可視になる場合に、入力フォーカスがどのように振る舞うか決定する。この値は次のいずれかにすることができる。

- `Parent` — 一般に、自分のサブウィンドウのひとつにフォーカスを割り当てる場合は、クライアントはこの値を使用しなければならない。サブウィンドウをアンマップすることによって、フォーカスをその親にもどすことができる。また、これはおそらくあなたが望むものである。

- `PointerRoot` — click-to-type のフォーカス管理の方針でこの値を使用すると、ウィンドウが不可視になることが、ウィンドウマネージャがフォーカスをどこかよそに移動しようと決定することと同時に起こるかもしれないので、レース条件が生じる。

- `None` — ほとんどのウィンドウマネージャがそうするように、ウィンドウマネージャがウィンドウの親を変更する場合、この値を使用すると問題が生じ、墜ちてしまう。入力フォーカスが `None` になると、おそらくそれを変更する方法はない。

`PointerRoot` と `None` のどちらも、本当に安全に使用できないことに注意。

規約

SetInputFocus リクエストを行使するクライアントは、その引数 *revert-to* を `Parent` にセットしなければならない。

入力フォーカス放棄したいクライアントのためにも、また規約が必要である。そうしたクライアントが入力フォーカスをセットするべき安全な値はいっさい存在しない。それゆえ、クライアントは入力データを無視しなければならない。

規約

クライアントは入力フォーカスを自分から進んで放棄してはならない。代わりに自分が受け取るキーボード入力を無視しなければならない。

4.1.8 カラーマップ

ウィンドウマネージャが管理する最上位ウィンドウをもつクライアントに代わって、ウィンドウマネージャはカラーマップのインストールとアンインストールに対して責任を持つ。

どのカラーマップをインストール、アンインストールするかに関して、クライアントはウィンドウマネージャにヒントを与える。クライアントは（下記の状況を除き）自分自身でカラーマップをインストール、アンインストールするべきではない。（ウィンドウマネージャが実装するカラーマップフォーカスの方針がどのようなものであれ、その結果として）クライアントの最上位ウィンドウがカラーマップフォーカスを得たとき、ウィンドウマネージャはクライアントのカラーマップを 1 つ以上インストールするように請け負う。

自分の最上位ウィンドウとサブウィンドウのすべてに同じカラーマップを使用するクライアントは、そのカラーマップの資源識別子を最上位ウィンドウの属性の *colormap* フィールドにセットしなければならない。そうしたクライアントは最上位ウィンドウにプロパティ WM_COLORMAP_WINDOWS をセットしてはならない。そのようなクライアントがカラーマップを変更したい場合は、最上位ウィンドウのカラーマップ属性を変更しなければならない。ウィンドウマネージャはウィンドウのカラーマップ属性の変更をたどって、適切にカラーマップをインストールする。

ウィンドウを生成するクライアントは、値 `CopyFromParent` を使用して、自分の親のカラーマップを継承することができる。ウィンドウマネージャはルートウィンドウの *colormap* フィールドが、クライアントが継承するのに適したカラーマップを含むように保証する。特に、そのカラーマップは `BlackPixel` と `WhitePixel` に対して区別できるカラーを与えることになる。

サブウィンドウや、オーバライドリダイレクト属性をセットしたポップアップウィンドウをもつ最上位ウィンドウは、それらのウィンドウが最上位ウィンドウとは異なるカラーマップを必須とする場合、プロパティ WM_COLORMAP_WINDOWS をもたなければならない。ウィンドウマネージャが自分独自のカラーマップフォーカスの方針に従って、カラーマップフォーカスをその最上位ウィンドウに割り当てるとき、ウィンドウマネージャがカラーマップをインストールしようとするべきウィンドウの資源識別子のリストを、このプロパティは含んでいる（セクション 4.1.2.8 を見よ）。このリストはカラーマップをインストールさせるクライアントにとって重要な順に並んでいる。ウィンドウマネージャはこのプロパティの変更をたどり、そのプロパティに含まれるウィンドウのカラーマップ属性の変更もたどることになる。

カラーマップの相対的な重要度が変化する場合、クライアントは新しい順序を反映するようにプロパティ

WM_COLORMAP_WINDOWS を更新しなければならない。最上位ウィンドウがこのリストに現れない場合、ウィンドウマネージャはそれがこのリストにあるどのウィンドウよりも高い重要度をもつとみなすことになる。

WM_TRANSIENT_FOR をもつウィンドウは、自分自身のプロパティWM_COLORMAP_WINDOWS をもつことができるし、適切であれば、自分が一時的であると表しているウィンドウのプロパティに現れることもできる。

根本的理由

自分がカラーマップを要求することに関するヒントを、クライアントがウィンドウマネージャにどのようにして与えるかということに対して、代案となるデザインがあった。その代案となるデザインはウィンドウのリストではなく、カラーマップのリストを指定した。現在のデザイン、すなわちウィンドウのリストは、2 つの理由によって選ばれた。第一に、それによってウィンドウマネージャはカラーマップのビジュアルを見つけることができたので、ビジュアルに依存してカラーマップをインストールする方針が可能になった。第二に、それによってウィンドウマネージャは、VisibilityChange で当のウィンドウに関するイベントを懇請することができて、カラーマップのインストールが必要なウィンドウが可視の場合にだけ、カラーマップをインストールするように保証できる。代案となるデザインは、これらの方針のどちらも考慮していない。

実装者へのアドバイス

クライアントは接続セットアップ情報の *min-installed-maps* フィールドと *max-installed-maps* フィールドを知っていなければならないし、その最小値が、InstallColormap リクエストの説明にある X プロトコルで定義された「要求リスト」に影響を与えることを承知していなければならない。要するに、最近インストールした *min-installed-maps* 個のカラーマップはインストールされるように保証されている。この値は 1 であることが多いので、複数のカラーマップを必要とするクライアントは注意しなければならない。

可能であればいつでも、クライアントは上記のメカニズムを使用して、ウィンドウマネージャにカラーマップのインストールを処理させるようにしなければならない。しかしながら、クライアントはポインタをグラブしている間は、自分自身でカラーマップのインストールを実行することが許されている。カラーマップのインストールを実行するクライアントは、最初のインストールに先だって、ウィンドウマネージャに通知するべきである。そのクライアントはまた、自分のカラーマップのインストールが完了したときにも、ウィンドウマネージャに通知するべきである。次の引数をもつ SendEvent リクエストを発行することによって、クライアントはウィンドウマネージャに通知する。

引数	値
destination	カラーマップをインストールしているスクリーンのルートウィンドウ
propagate	`False`
event-mask	ColormapChange
event	ClientMessage
window	上記のルートウィンドウ
type	WM_COLORMAP_NOTIFY
format	32
data[0]	クライアントがカラーマップのインストールを開始、停止する原因となったイベントのタイムスタンプ
data[1]	クライアントがカラーマップのインストールを開始する場合は 1, 終える場合は 0
data[2]	予約済み, 0 にするべき
data[3]	予約済み, 0 にするべき
data[4]	予約済み, 0 にするべき

この特徴はこのドキュメントのバージョン 2.0 で導入された。すべてのウィンドウマネージャがこの特徴を実装することが期待できるようになるまで、かなりの期間がかかるだろう。クライアントはこの特徴を使用する前に、ウィンドウマネージャのコンプライアンスレベル（追従レベル）を（セクション 4.3 で記述しているメカニズムを使用して）調べ、そのウィンドウマネージャがこの特徴をサポートするか確認するべきである。このことは、クライアントと古いウィンドウマネージャの間で、カラーマップのインストールが衝突するのを防ぐために必要である。

クライアントがカラーマップのインストールのコントロールを要請している間ずっと、ウィンドウマネージャはカラーマップのインストールを差し控えなければならない。ウィンドウマネージャはクライアントがカラーマップのインストールを終えたときに、自分のカラーマップフォーカスの方針を回復できるように、インストールされるカラーマップ一式をたどり続けなければならない。

このテクニックにはレース条件があり、それはクライアントが自分の通知メッセージ（上記の ClientMessage イベント）を発行した後でさえも、カラーマップがインストールされ続けるという結果をもたらすかもしれない。例えば、クライアントの SendEvent リクエストと InstallColormap リクエストが終わるまで実行されない、いくつかの InstallColormap リクエストをウィンドウマネージャが発行するかもしれず、したがってウィンドウマネージャはクライアントのカラーマップをアンインストールしないかもしれない。クライアントが依然としてポインタをグラブしたままで、クライアントが「完了」メッセージを発行する前にこのことが起きれば、そのクライアントは望ましいカラーマップを再びインストールしてもよい。

実装者へのアドバイス

クライアントがこのメカニズムをポップアップウィンドウのようなものに対して、そしてオーバライドリダイレクト属性をセットしたウィンドウを使用するアニメーションに対して使用するように期待されている。

クライアントが「完了」メッセージを発行できなければ、ウィンドウマネージャはカラーマップをインストールする自分の方針が一時停止した状態に陥るかもしれない。ウィンドウマネージャの実装者は、カラーマップをインストールする方針をユーザからのコマンドに応えてリセットする特徴を実装したいと思うかもしれない。

4.1.9 アイコン

クライアントは次のようにすることにより、自分のアイコンの望ましい外観について、ウィンドウマネージャにヒントを与えることができる。

- WM_ICON_NAME の文字列をセットする。

  アイコンに関する概念がクライアントがもつ概念と著しく異なるウィンドウマネージャにとっては、この文字列は頼みの綱となるので、すべてのクライアントはこれをセットしなければならない。

- ピクスマップをプロパティ WM_HINTS の *icon_pixmap* フィールドにセットし、できる限りもうひとつのピクスマップを *icon_mask* フィールドにセットする。

  ウィンドウマネージャは、マスクとなるピクスマップでマスクしたピクスマップを表示するように期待されている。そのピクスマップはルートウィンドウのプロパティ WM_ICON_SIZE のなかに見つかるサイズのひとつでなければならない。このプロパティが見つからなければ、そのウィンドウマネー

ジャはアイコンピクスマップを表示しそうもない。ウィンドウマネージャは普通、WM_ICON_SIZE に合わないピクスマップをクリップするか、タイルパターンとして描画する。

- ウィンドウをプロパティ WM_HINTS の *icon_window* フィールドにセットする。

  クライアントが Iconic 状態にあるときはいつでも、そのウィンドウをマップするようにウィンドウマネージャは期待されている。一般に、アイコンウィンドウのサイズはルートウィンドウに WM_ICON_SIZE が存在すれば、それが指定するサイズのひとつでなければならない。ウィンドウマネージャはアイコンウィンドウのサイズを自由に変更できる。

Iconic 状態にあるとき、ウィンドウマネージャは通常次のことを保証する。

- そのウィンドウの WM_HINTS.*icon_window* がセットされていれば、それが指定するウィンドウが可視である。
- そのウィンドウの WM_HINTS.*icon_window* はセットされていないが、WM_HINTS.*icon_pixmap* がセットされていれば、それが指定するピクスマップが可視である。
- そうでなければ、そのウィンドウの WM_ICON_NAME の文字列が可視である。

クライアントは自分のアイコンウィンドウに関する次の規約に従わなければならない。

規約

1. アイコンウィンドウは、ルートウィンドウの子で `InputOutput` クラスでなければならない。
2. アイコンウィンドウのサイズは、ルートウィンドウにあるプロパティ WM_ICON_SIZE が指定するサイズのひとつでなければならない。
3. アイコンウィンドウは、該当するスクリーンの、ルートウィンドウのビジュアルとデフォルトのカラーマップを使用しなければならない。
4. クライアントは、自分のアイコンウィンドウをマップしてはならない。
5. クライアントは、自分のアイコンウィンドウをアンマップしてはならない。
6. クライアントは、自分のアイコンウィンドウの配置構成を変更してはならない。
7. クライアントは、自分のアイコンウィンドウにオーバライドリダイレクト属性をセットしてはならないし、ResizeRedirect で自分のアイコンウィンドウに関するイベントを懇請してはならない。
8. クライアントは、自分のアイコンウィンドウで入力イベントを受け取れることに依存するべきではない。
9. クライアントは、自分のアイコンウィンドウの境界を操作するべきではない。
10. クライアントは、Exposure で自分のアイコンウィンドウに関するイベントを懇請するべきであり、要求されたときはそれを再描画するべきである。

ウィンドウマネージャがクライアントのアイコンウィンドウに対し、キーボードやマウスの入力イベントをサポートするかどうかに関しては、ウィンドウマネージャによって異なる。ほとんどのウィンドウマネージャはクライアントがキーボードとマウスボタンのサブセットを受け取ることを許す。

アイコンウィンドウで見つけるプロパティWM_NAME, WM_ICON_NAME, WM_NORMAL_HINTS, WM_HINTS, WM_CLASS, WM_TRANSIENT_FOR, WM_PROTOCOLS, WM_COLORMAP_WINDOWS, WM_COMMAND や、WM_CLIENT_MACHINE を、ウィンドウマネージャはすべて無視する。

4.1.10 ポップアップウィンドウ

ウィンドウをポップアップしたいクライアントは、次の 3 つの事項のひとつを行うことができる。

1. クライアントはもうひとつの普通の最上位ウィンドウを生成し、マップすることができる。ウィンドウマネージャはそれを普通に装飾し、普通に管理することになる。続くウィンドウグループの解説を見よ。

2. そのウィンドウが比較的短い時間可視であり、いくらか軽い扱いを受けるのがふさわしい場合、クライアントはプロパティWM_TRANSIENT_FOR をセットすることができる。クライアントはより少ない装飾を期待できるが、そのウィンドウに関する通常のウインドウマネージャのためのプロパティをすべてセットできる。ダイアログボックスがその例である。

3. そのウィンドウがかなり短い時間可視であり、いっさい装飾するべきではない場合は、クライアントはオーバライドリダイレクト属性をそのウィンドウにセットすることができる。一般に、これを行う場合は、そのウィンドウをマップしている間ずっと、ポインタをグラブしなければならない。ウィンドウマネージャはこれらのウィンドウに関していっさい干渉しないので、注意して使用しなければならない。ポップアップメニューが適切な使用例である。

実装者へのアドバイス

ウィンドウマネージャはオーバライドリダイレクト属性がセットされたウィンドウを管理していないので、ユーザはそのウィンドウを移動したり、そのサイズやスタックの順序を変更したり、入力フォーカスをそれに移したりすることができない。クライアントがオーバライドリダイレクト属性をセットしたウィンドウでキーボードの打鍵を受け取る必要がある場合、クライアントはキーボードをグラブするか、あるいはオーバライドリダイレクト属性をセットしていない、局所的に能動的か広域的に能動的なフォーカスモデルを選んだ、もうひとつの最上位ウィンドウをもつべきである。残りのウィンドウが WM_TAKE_FOCUS メッセージ（ClientMessage イベント）を受け取った場合、またはセクション 4.1.7 の広域的に能動的なフォーカスモデルの説明で記載しているイベントのひとつを受け取った場合、クライアントはオーバライドリダイレクト属性をセットしたウィンドウに入力フォーカスをセットしてもよい。

自分が一時的であると表すウィンドウがアイコン化されるときに、WM_TRANSIENT_FOR をもつウィンドウがアイコン化されるべきかを、ウィンドウマネージャは自由に決定できる。WM_TRANSIENT_FOR をもつウィンドウを表示するクライアントが、それらが一時的であると表すウィンドウをアイコン化させる（あるいは、させるように要求する）場合は、そのクライアントが WM_TRANSIENT_FOR をもつウィンドウに関して同じ操作を実行するように要求する必要はない。すなわち、それがウィンドウマネージャが適用したい方針であれば、ウィンドウマネージャがその状態を変更することになる。

4.1.11　ウィンドウグループ

一連の最上位ウィンドウを（仮にそれらが多くのクライアントに所属していても）ユーザの視点からみて関連しているものとして扱うべきであれば、プロパティWM_HINTS の *window_group* フィールドを使用して、それらを関連させなければならない。

それらのウィンドウのひとつ（すなわち、残りのウィンドウが指すひとつのウィンドウ）は、グループリーダであり、個々のプロパティとは対照的に、グループのプロパティを保持する。ウィンドウマネージャはグループリーダを、グループのそのほかのウィンドウとは異なるように扱ってよい。例えば、グループリーダは最大限の装飾が施されてもよいし、グループのそのほかのウィンドウはそれが制限されてもよい。

クライアントがグループリーダを常にマップする必要はない。すなわち、グループリーダは単にプレースホルダとして存在するだけのウィンドウであってもよい。

グループに含まれるウィンドウを扱うための方針を決定することは、ウィンドウマネージャしだいである。現在、個々の操作とは対照的に、グループの操作をクライアントが要求する方法はない。

4.2　ウィンドウマネージャの行動に対するクライアントの応答

ウィンドウマネージャは、主にクライアントの最上位ウィンドウにある、クライアントの資源に関して、多くの操作を実行する。クライアントはこのことに関して争おうとするべきではないが、ウィンドウマネージャのその操作に関して通知を受け取るように選んでもよい。

4.2.1　親の変更

ウィンドウマネージャのなかには、ルートウィンドウの子として生成されたウィンドウを、そのウィンドウマネージャに所属しているあるウィンドウの子として表示するために、クライアントの最上位ウィンドウの親を変更してしまうものもあるということを、クライアントは承知しておくべきである。この親の変更がクライアントに与える影響は、次の通りである。

- QueryTree リクエストの戻り値である親ウィンドウは、もはや親が変更されたそのウィンドウを生成した CreateWindow リクエストに与えたものではない。最上位ウィンドウの親がどのウインドウに変更されたのか、クライアントがそのウィンドウの正体に気づいている必要があってはならない。特に、さらに多くの最上位ウィンドウを生成したいクライアントは、それらの新しいウィンドウの親としてルートウィンドウを使用し続けなければならない。

- X サーバは ConfigureWindow リクエストの座標 (x, y) を新しい親の座標空間で解釈する。実際には、親を変更するウィンドウマネージャが普通はこれらの操作を横取りしてしまう（セクション 4.2.2 を見よ）ので、通常それらの座標を X サーバが解釈することはない。クライアントはこれらのリクエストに対して、ルートウィンドウの座標空間を使用しなければならない（セクション 4.1.5 を見よ）。

- 特定の兄弟ウィンドウを指定する ConfigureWindow リクエストは失敗するかもしれない。指定されるそのウィンドウがかつては兄弟であったとしても、親を変更する操作の後ではもはやそうではないからである（セクション 4.1.5 を見よ）。

- GetGeometry リクエストによって返された座標 (x, y) は、その親の座標空間で表されているため、親の変更操作の後では直接的に役に立つものではない。

- 背景ピクスマップに `ParentRelative` を指定した場合は予測できない結果となる。

- カーソルに `None` を指定した場合は予測できない結果となる。

自分のウィンドウの親が変更されたときに、それを通知して欲しいクライアントは、StructureNotify で自分の最上位ウィンドウに関するイベントを懇請することができる。クライアントは親の変更が生じた場合に ReparentNotify イベントを受け取ることになる。クライアントが最上位ウィンドウを Withdrawn 状態にするとき、ウィンドウマネージャは、そのウィンドウの親をかつてどこか別のウィンドウに変更していた場合、その親を変更してルートウィンドウにもどす。

ウィンドウマネージャがクライアントのウィンドウの親を変更する場合、親が変更されたウィンドウは親ウィンドウのセーブセットに置かれることになる。このことは、親が変更されたウィンドウが、ウィンドウマネージャが終了する場合に破壊されないこと、および、それをマップしていなかった場合に再びマップすることを意味する。このことは、一時的ウィンドウやクライアントのアイコンウィンドウを含む、ウィンドウマネージャが親を変更するクライアントのウィンドウすべてに適用されることに注意。

4.2.2　操作のリダイレクト

ウィンドウマネージャのなかには、クライアントのいくつかのリクエストを横取りし、リダイレクトするように調停するものもあるということを、クライアントは承知しているべきである。リダイレクトされるリクエストは実行されない。代わりに、そのリクエストによってイベントがウィンドウマネージャに送られることになる。イベントを受け取ったそのウィンドウマネージャは、何もしないことにするか、引数を変更するか、あるいはクライアントの代わりにリクエストを実行しようと判断するかもしれない。

リクエストがリダイレクトされるかもしれないということから、リダイレクト可能なリクエストを発行するとき、そうしたリクエストが実際に実行されるときでも、クライアントはあらゆるそうしたリクエストが実際に実行されるとみなせない、ということがわかる。リダイレクト可能なリクエストは、MapWindow, ConfigureWindow と CirculateWindow である。

実装者へのアドバイス

MapWindow リクエストが横取りされ、PolyLine がマップしていないウィンドウに出力するかもしれないので、次のことは正しくない。

MapWindow A
PolyLine A GC <point> <point> ...

クライアントはウィンドウを描画する前に Expose イベントを待つべきである[4]。

この次の例は、与えた引数で ConfigureWindow リクエストが実際に実行されると、間違って想定している。

ConfigureWindow 幅=N 高さ=M
<ウィンドウが $N \times M$ であると想定して出力する>

クライアントは StructureNotify で自分のウィンドウに関するイベントを懇請し、ConfigureNotify イベントをたどることによってウィンドウのサイズを監視しなければならない。

[4] 仮にクライアントがバッキングストア属性を `Always` にセットしている場合でも、このことはあてはまる。バッキングストア属性は単にヒントであり、X サーバはバッキングストアの内容を維持するのをいつやめてもよい。

クライアントは、自分がマップしたばかりのウィンドウにフォーカスをセットしようとするときに、特に注意しなければならない。この一連のリクエストは X プロトコルのエラーをもたらすかもしれない。

MapWindow B
SetInputFocus B

MapWindow リクエストが横取りされた場合、そのウィンドウは依然としてマップしていないので、SetInputFocus リクエストがエラーを発生させることになる。この問題の解決法は、VisibilityChange でクライアントがそのウィンドウに関するイベントを懇請して、ウィンドウが可視であることを示す VisibilityNotify イベントを自分が受け取るまで、SetInputFocus リクエストの発行を遅らすことである。

このテクニックは正しい操作を保証するものでない。SetInputFocus リクエストが X サーバに届く頃には、ユーザはそのウィンドウをアイコン化してしまっているかもしれず、やはりエラーを引き起こすかもしれない。また、ウィンドウマネージャは Iconic 状態でウィンドウをマップしようと思うかもしれず、その場合そのウィンドウは可視でないことになる。このことは VisibilityNotify イベントの発生を不定に遅らすことになる。クライアントはこれらのケースを扱えるように用意しておくべきである。

オーバライドリダイレクト属性をセットしたウィンドウは、リダイレクトから免れることができる。しかし、その属性を最上位ウィンドウにセットするのは、その属性をセットしたウィンドウをマップしている間ずっと、残りのウィンドウの入力処理を妨げる必要がある場合（セクション 4.1.10）と、ResizeRequest イベントに応答する間（セクション 4.2.9）に限るべきである。

Withdrawn 状態でない最上位ウィンドウをいっさいもたないで、オーバライドリダイレクト属性をセットした最上位ウィンドウをマップするクライアントは、その系の状態のすべての責務を引き継いでいる。そうしたクライアントの責務は、次の通りである。

- どんなものであれウィンドウマネージャが以前から存在していれば、それが自分の行動に干渉するのを妨げること。
- 自分がそのウィンドウをアンマップした後、どんなものであれウィンドウマネージャが以前から存在していれば、それが混乱しないように現状を正確に元へもどすこと。

事実上、この種のクライアントは臨時のウィンドウマネージャとして行動する。そうしたクライアントは、ウィンドウマネージャが維持しようとしているユーザインタフェイスの方針について知らないし、そうしたクライアントのユーザインタフェイスの振る舞いは要求の少ないクライアントのユーザインタフェイスと衝突しがちなので、そうするのは断固として思いとどまるべきである。

4.2.3 ウィンドウの移動

ウィンドウマネージャがサイズの変更を伴わずに最上位ウィンドウを移動させる場合、クライアントはその移動に伴う、合成の ConfigureNotify イベントを受け取ることになる。そのイベントは、ルートウィン

ドウの座標空間で、その新しい位置を記述する。クライアントは移動させられることに対して、より良い位置に移動しようと試みることで応えるべきではない。

最上位ウィンドウの親は変更されてしまっているかもしれないので、仮に最上位ウィンドウに関する本物の ConfigureNotify イベントが、そのウィンドウの新しい位置は自分の親に対して変化していないと報告しても、そうした本物の ConfigureNotify イベントはすべて、ルートウィンドウに関するそのウィンドウの位置が変わってしまったかもしれない、という意味を含む。そのイベントに含まれる座標は、この場合、直接的には役に立たないことに注意。

ウィンドウマネージャは、次の引数をもつ SendEvent リクエストを用いて、そうしたイベントを送ることになる。

引数	値
destination	クライアントのウィンドウ
propagate	`False`
event-mask	StructureNotify

4.2.4　ウィンドウのサイズの変更

クライアントは StructureNotify で自分の最上位ウィンドウに関するイベントを懇請することによって、サイズが変更されるという通知を受け取るように選ぶことができる。そのイベントのサイズの情報は正しいが、位置はその親ウィンドウに対するものである。（その親ウィンドウはルートウィンドウではないかもしれない）。

サイズが変更されることに対してクライアントは、自分に与えられたサイズを受諾し、それに最善を尽くすことで応えなければならない。サイズが変更されることに対してクライアントは、自分自身をより良いサイズに変更しようとすることで応えるべきではない。そのサイズで動作するのが不可能であれば、クライアントは Iconic 状態へ変化するように要求して構わない。

4.2.5　アイコン化

Withdrawn 状態にない最上位ウィンドウは、マップされていれば Normal 状態にあり、マップされていなければ Iconic 状態にある。仮にそのウィンドウの親が変更されていたとしても、このことはあてはまり、ウィンドウマネージャはそのウィンドウを Iconic 状態に切り替えるときに、その親と同様にそのウィンドウをアンマップすることになる。

クライアントは、StructureNotify で自分の最上位ウィンドウに関するイベントを懇請することにより、こうした状態の変化が通知されるように選ぶことができる。最上位ウィンドウは、Iconic 状態になるときは UnmapNotify イベントを受け取り、Noamal 状態になるときは MapNotify イベントを受け取ることになる。

4.2.6　カラーマップの変更

自分のカラーマップがインストール、アンインストールされるのを通知して欲しいクライアントは、ColormapChange で自分の最上位ウィンドウに関するイベントと、自分の最上位ウィンドウのプロパティ

WM_COLORMAP_WINDOWS で自分が指定したウィンドウすべてに関するイベントを懇請しなければならない。そうしたウィンドウにカラーマップがインストール、アンインストールされるときに、そのウィンドウは *new* フィールドが `False` の ColormapNotify イベントを受け取ることになる。

4.2.7 入力フォーカス

クライアントは FocusChange で自分の最上位ウィンドウに関するイベントを懇請することにより、自分が入力フォーカスをもつという通知を要求することができる。その場合、クライアントは FocusIn と FocusOut イベントを受け取ることになる。最上位ウィンドウのプロパティ WM_PROTOCOLS に WM_TAKE_FOCUS をセットしていなかった場合や、次の事項のひとつを満たしていなかった場合は、クライアントは自分のサブウィンドウのひとつに入力フォーカスをセットする必要があっても、そうしてはならない。

- WM_HINTS の *input* フィールドを `True` にセットし、自分の最上位ウィンドウのひとつに実際に入力フォーカスをもっている。
- WM_HINTS の *input* フィールドを `False` にセットし、セクション 4.1.7 で記述しているようなふさわしいイベントを受け取った。
- セクション 4.1.7 で記述しているような WM_TAKE_FOCUS メッセージ（ClientMessage イベント）を受け取った。

クライアントはフォーカスを移そうとする際、ポインタをワープさせてはならない。つまり、フォーカスをセットしてもポインタはそのままにしておかなければならない。さらに多くの情報については、セクション 6.2 を見よ。

ひとたびクライアントがこれらの条件を満たせば、SetInputFocus リクエストを用いることによって、クライアントはフォーカスを自分の別のウィンドウに移してもよい。そのリクエストは次のように定義される。

SetInputFocus
focus: ウィンドウ or `PointerRoot` or `None`
revert-to: {`Parent`, `PointerRoot`, `None`}
time: タイムスタンプ または `CurrentTime`

規約

1. SetInputFocus リクエストを用いるクライアントは *time* フィールドを、そうしようとする原因となったイベントのタイプスタンプにセットするべきである。FocusIn イベントはタイムスタンプをもたないので、FocusIn イベントにはその資格がない。クライアントは対応する EnterNotify イベントなしにフォーカスを獲得してもよい。クライアントは引数 *time* に `CurrentTime` を使用するべきではない。
2. SetInputFocus リクエストを使用して、フォーカスを自分のウィンドウのひとつにセットするクライアントは、その *revert-to* フィールドを `Parent` にセットするべきである。

4.2.8　ClientMessage イベント

ClientMessage イベントが自分自身に送られてくるのをクライアントが妨げる方法はない。

プロパティWM_PROTOCOLS をもつ最上位ウィンドウには、そのプロパティに含まれるアトムが指定するプロトコルに固有な ClientMessage イベントが送られてくるかもしれない（セクション 4.1.2.7 を見よ）。すべてのプロトコルで、ClientMessage イベントは次のようになる。

引数	値
type	WM_PROTOCOLS
format	32
data[0]	自分のプロトコルを指定するアトム
data[1]	タイムスタンプ

イベントの残りのフィールドは、*window* フィールドを含め、そのプロトコルが決定する。

次の引数をもつ SendEvent リクエストを使用して、これらのイベントを送ることになる。

引数	値
destination	クライアントのウィンドウ
propagate	`False`
event-mask	()（空のイベントマスク）
event	プロトコルが指定するもの

4.2.8.1　ウィンドウの消去

通常複数の最上位ウィンドウをもち、自分の最上位ウィンドウのいくつかが消去された後もサーバと接続されたままであるべきクライアントは、そのようなウィンドウ関してそれぞれ、アトム WM_DELETE_WINDOW をプロパティWM_PROTOCOLS に含めなければならない。そうしたクライアントは上記のような *data*[0] フィールドが WM_DELETE_WINDOW である ClientMessage イベントを受け取ることになる。

WM_DELETE_WINDOW メッセージを受け取ったクライアントは、ユーザが仮定上のメニューから「ウィンドウの消去」を選択したかのように振る舞わなければならない。クライアントはユーザと、どんなものでも確認のための対話を実行しなければならず、クライアントがそのウィンドウの消去を完結すると決めた場合は、次のことをしなければならない。

- そのウィンドウの状態を（セクション 4.1.4 の説明のように）Withdrawn 状態に変更するか、そのウィンドウを破壊する。
- そのウィンドウと結びついている内部状態をすべて破壊する。

その対話でユーザが消去を中断する場合は、クライアントはそのメッセージを無視しなければならない。

クライアントはユーザと対話して、例えば、消去しようとしているウィンドウと結びついているファイルを保存するべきか、あるいはそのウィンドウの消去を取り消すべきかどうか尋ねることができる。クライアントはウィンドウそれ自身を破壊するように要求されているのではない。したがって、その資源は再利用してもよいが、それと結びついている状態（例えば、バッキングストア）はすべて解放しなければならない。

クライアントが破壊を中断して、それからユーザが再び「ウィンドウの消去」を選択した場合、ウィンドウマネージャは WM_DELETE_WINDOW のプロトコルを再び始めなければならない。プロパティ WM_PROTOCOLS に WM_DELETE_WINDOW を含むウィンドウに関して、ウィンドウマネージャは DeleteWindow リクエストを使用してはならない。

プロパティ WM_PROTOCOLS に WM_WINDOW_DELETE を含めないように選択したクライアントは、ユーザがそのクライアントの最上位ウィンドウのひとつを消去するように求めた場合、X サーバとの接続を断たれるかもしれない。

4.2.9 リクエストのリダイレクト

普通のクライアントも、ちょうどウィンドウマネージャがするように、SubstructureRedirect で親ウィンドウに関するイベントを懇請するか、ResizeRedirect でウィンドウそれ自身に関するイベントを懇請することにより、リダイレクトのメカニズムを使用することができる。しかしながら、これらのイベントは 1 つのウィンドウにつき 1 つのクライアントしか懇請できないので、衝突しないようにするためには規約が必要である。

規約

（ウィンドウマネージャを含む）クライアントは、SubstructureRedirect と ResizeRedirect で自分が所有するウィンドウに関するイベントだけを懇請しなければならない。

特に、自分のサイズが変更される場合に何か特別な行動をとる必要のあるクライアントは、ResizeRedirect で自分の最上位ウィンドウに関するイベントを懇請することができる。そのようなクライアントは、自分のウィンドウのサイズをウィンドウマネージャが変更する場合に ResizeRequest イベントを受け取ることになり、サイズの変更は、実際には行われないことになる。ウィンドウマネージャがクライアントのウィンドウのサイズを変更したいという情報を、クライアントは自分の好きなように利用して構わない。しかし、クライアントはそのウィンドウを、そのイベントが指定する幅と高さに、なるべく早く構成するべきである。サイズの変更を、ウィンドウマネージャが横取りし実行する（そうしてこの過程が再び始まる）のではなく、実際にこの段階でサイズが変更されるのを保証するために、クライアントは一時的にオーバライドリダイレクト属性をそのウィンドウにセットする必要がある。

規約

ResizeRequest イベントを受け取ったクライアントは、次のようにすることで応えるべきである。

- そのイベントが指定するウィンドウのオーバライドリダイレクト属性をセットする。
- できるだけ早く、そのほかのあらゆるジオメトリに関するリクエストを使用する前に、そのイベントが指定するウィンドウを、そのイベントが指定する幅と高さに構成する。
- そのイベントが指定するウィンドウのオーバライドリダイレクト属性をクリアする。

ウィンドウマネージャは、クライアントがこの規約に従っていないのを検出したら、どんな処置であれ、自分がそのクライアントを扱うのに適すると考える処置をとって構わない。

4.3　セレクションによるウィンドウマネージャとの通信

自分が管理するスクリーンについてそれぞれ、ウィンドウマネージャは WM_S*n* という名前のセレクションを獲得することになる。セクション 1.2.6 の説明のように *n* はスクリーン番号である。ウィンドウマネージャはセクション 2.8 で記述している「マネージャセレクション」の規約に従わなければならない。クライアントがこのセレクションに関する変換リクエストを発行することによって、いろいろな情報やサービスを要求できるというのが、その趣旨である。ウィンドウマネージャは自分のマネージャセレクションに関して、次のターゲットへの変換をサポートしなければならない。

アトム	タイプ	受け取るデータ
VERSION	INTEGER	2 つの整数。それらはそれぞれ、そのウィンドウマネージャが従う ICCCM のメジャーリリース番号とマイナーリリース番号である。このバージョンの ICCCM では、それらの番号は 2 と 0 である[5]。

4.4　ウィンドウマネージャのためのプロパティのタイプの要約

ウィンドウマネージャに関するプロパティは、次の表に要約されている（*Xlib — C Language X Interface* のセクション 14.1 も見よ）。

プロパティ名	タイプ	フォーマット	セクション
WM_CLASS	STRING	8	4.1.2.5
WM_CLIENT_MACHINE	TEXT		4.1.2.9
WM_COLORMAP_WINDOWS	WINDOW	32	4.1.2.8
WM_HINTS	WM_HINTS	32	4.1.2.4
WM_ICON_NAME	TEXT		4.1.2.2
WM_ICON_SIZE	WM_ICON_SIZE	32	4.1.3.2
WM_NAME	TEXT		4.1.2.1
WM_NORMAL_HINTS	WM_SIZE_HINTS	32	4.1.2.3
WM_PROTOCOLS	ATOM	32	4.1.2.7
WM_STATE	WM_STATE	32	4.1.3.1
WM_TRANSIENT_FOR	WINDOW	32	4.1.2.6

[5]特別な場合として、セレクションに関するリクエストを実装したくないクライアントは、単に適切な WM_S*n* セレクションに関して GetSelectionOwner リクエストを発行してもよい。このセレクションが所有されていれば、クライアントはそのウィンドウマネージャがバージョン 2.0 かそれ以降の ICCCM に従うとみなしてよい。

第6章　Manipulation of Shared Resources

X バージョン 11 では、例えば入力フォーカス、ポインタ、カラーマップといった、多くの共有資源を操作することをクライアントは許されている。クライアントが秩序正しく資源を共有するために、規約が必要である。

6.1　入力フォーカス

明示的に入力フォーカスをセットするクライアントは、次の 2 つのモードのひとつを守るべきである。

- Locally active mode
- Globally active mode

規約

1. 局所的に能動的なクライアントは、入力フォーカスがすでに自分のウィンドウのひとつにあるとき、または自分が WM_TAKE_FOCUS メッセージを受け取ったときにだけ、入力フォーカスを自分のウィンドウのひとつにセットするべきである。このようなクライアントは WM_HINTS の *input* フィールドを `True` にセットしなくてはならない。
2. 広域的に能動的なクライアントは、自分がマウスボタンイベントか受動的にグラブされたキーボードイベントを受け取ったとき、または WM_TAKE_FOCUS メッセージを受け取ったときにだけ、入力フォーカスを自分のウィンドウのひとつにセットするべきである。このようなクライアントは WM_HINTS の *input* フィールドを `False` にセットしなくてはならない。
3. 加えてクライアントは、`CurrentTime` ではなく、自分が入力フォーカスをセットしようとする原因となったイベントのタイムスタンプを、SetInputFocus リクエストの *time* フィールドとして、使用しなければならない。

6.2　ポインタ

一般にクライアントは、ポインタをワープさせてはならない。しかしながら、ウィンドウマネージャはそうするかもしれない。(例えば、いつでも入力フォーカスのあるウィンドウにポインタがあることを変わらず維持する目的で)。そのほかのウィンドウマネージャは、ユーザがポインタを唯一コントロールしているという幻想を維持したいかもしれない。

規約

1. クライアントはポインタをワープさせてはならない。
2. ポインタをワープさせることに固執するクライアントは、WarpPointer リクエストの引数 *src-window* を、自分のウィンドウのひとつにセットするときに限り、そうしなければならない。

6.3　グラブ

クライアントがあるウィンドウに関してボタンやキーのグラブを確立しようとしても、なにかそのほかのクライアントが既にその同じウィンドウ関して相反するグラブを確立していた場合は、そのグラブの確立は失敗することになる。それゆえ、グラブは共有資源であり、それらの利用には規約が必要である。

クライアントはウィンドウマネージャが動作していても、可能な限りそれが動作していない場合と同様に振る舞うべきであるという原則に準拠して、入力フォーカスを持ったクライアントは、利用可能なキーとボタンをすべて、自分が受け取ることができるとみなしてよい。

規約

ウィンドウマネージャの機能のためのものとして登録されているキーシム（例えば、hypothetical WINDOW キーシム）をもつキーに関するイベントを除き、自分のクライアントがすべてのキーとすべてのボタンからイベントを受け取れるようなメカニズムを、ウィンドウマネージャは確実に提供しなければならない。

言い換えれば、X コンソーシアムがいくつかのキーシムをウィンドウマネージャの機能のために予約されているものとして登録しなければ、あるいは登録するまで、ウィンドウマネージャは、クライアントがどのキーやボタンからも（修飾キーにかかわらず）イベントを受け取れるようなメカニズムを提供するべきである。現在、ウィンドウマネージャの機能のために登録されているキーシムはいっさいない。

そうであるとしても、ウィンドウマネージャのなかにはこの規約に従わないことを選ぶものもあるかもしれないし、ユーザがいくつかのキーイベントを伝えるために都合のよい手段を提供しないものもあるかもしれないので、クライアントはプログラムの動作を喚起するために使用するキーとボタンの組み合わせを変更できるように勧告されている。

規約

クライアントは自分が所有するウィンドウに関してだけ、ボタンとキーのグラブを確立するべきである。

特に、クライアントの最上位ウィンドウでグラブを確立したいウィンドウマネージャが、ルートウィンドウでグラブを確立するか、そのウィンドウの親を変更して適切なアンセスタでグラブを確立するべきであるということを、この規約は意味している。いくつかのケースでは、ウィンドウマネージャは受け取るイベントを消費して、それに続くイベントがクライアントに向かう状態にウィンドウを置きたいかもしれない。例:

- ウィンドウでのクリックは、フォーカスをセットするが、そのクリックはそのクライアントに供給されない。
- 埋もれているウィンドウでのクリックは、それをライズするが、やはりそのクリックはそのクライアントに供給されない。

より典型的には、ウィンドウマネージャは、自分がイベントを処理したあとで、そのイベントをクライアントが使用できるようにすることによって、キーとボタンの組み合わせに対するクライアントのセマンティクスを置換するというよりはむしろ、増やさなければならない。これを確実に行うには、ウィンドウマネージャはその親ウィンドウで、次のようにグラブを確立しなければならない。

pointer/keyboard-mode == `Synchronous`

それから、ウィンドウマネージャは次のような指定の AllowEvents リクエストを使用して、そのグラブを解放しなければならない。

mode == `ReplayPointer/Keyboard`

このような方法では、クライアントはまるでそれが横取りされなかったかのように、イベントを受け取ることになる。

明らかに、これらの規約は可能なユーザインタフェイスの方針に、いくつかの制限を置く。ウィンドウマネージャが自分のユーザインタフェイスの方針を実装することの自由と、クライアントがそれを実装することの自由との間には、トレードオフがある。そのジレンマは次のように解決される:

- クライアントが所定のどのキーやボタンからもイベントを受け取るかどうか、受け取るときを、ウィンドウマネージャに決定させる。
- 恐らく「Quote」キーのような、ユーザがどのキーやボタンからもイベントをクライアントに送ることができるメカニズムを提供するように、ウィンドウマネージャに要件を置く。

6.4 カラーマップ

セクション 4.1.8 は、クライアントが自分のカラーマップの必要性について、ウィンドウマネージャとコミュニケーションするための規約を prescribe している。あなたのクライアントが DirectColor 型のアプリケーションであれば、標準カラーマップを共有することに関連する規約について、あなたは Xlib — C Language X Interface のセクション 14.3 を consult しなければならない。そうしたクライアントは、そこで記述している適切なスクリーンのルートウィンドウにあるプロパティを探し、生成しなければならない。

タイプ RGB_COLOR_MAP のプロパティの内容は、次のようになる。

———————-

RGB_COLOR_MAP を消去または置換するとき、そのプロパティを消去するだけでは十分ではない。それに関連するカラーマップ資源を解放することが同様に重要である。*kill_id* が 1 より大きい場合、引数として *kill_id*

をもつ KillClient リクエストを発行することで、その資源を解放しなければならない。*kill_id* が 1 の場合、引数 *colormap* に（RGB_COLOR_MAP の）colormap をもつ FreeColormap リクエストを発行することで、その資源を解放しなければならない。kill_id が 0 の場合、その資源を解放してはならない。RGB_COLOR_MAP を生成するクライアント、特にこの目的のためにカラーマップ資源が生成される、kill_id を 1 にセットしなければならない。（そして単一の接続を使用して、そのような標準カラーマップを 1 つ以上生成できる。）RGB_COLOR_MAP を生成するクライアント、ある方法でカラーマップ資源が共有される、（例えば、ルートウィンドウに対するデフォルトのカラーマップである）、arbitary 資源を生成し、その資源識別子を *kill_id* に使用しなければならない。（そしてその接続では、そのほかの標準カラーマップをいっさい生成してはならない。）

規約

RGB_COLOR_MAP プロパティが *visual_id* を含めないほど短い場合、その *visual_id* が適切なスクリーンのルートウィンドウのビジュアルであるとみなせる。RGB_COLOR_MAP プロパティが *kill_id* フィルードを含めないほど短い場合、値 0 と見なせる。

接続ハンドシェイクの間、サーバはクライアントに inform する。それぞれのスクリーンのデフォルトのカラーマップを。これはルートウィンドウのビジュアルのカラーマップであり、クライアントはそれを使用してカラーマップの共有の程度を改善することができる。ルートウィンドウのビジュアルを使用するならば。

6.5　キーボードマッピング

X サーバはテーブルを保持している。（それは GetKeyboardMapping リクエストによって読まれる）そのテーブルは記述する。シンボルの集合を。サーバによって作成されるキーコードをそれぞれ表すその対応するキーに appearing on。

このテーブルはサーバの操作に影響しない。in any way。すなわち、それは単に自分が受け取るキーコードを理解しようとするクライアントが用いるデータベースである。しかしながら、それは共有リソースであり、規約を必要とする。

クライアントがこのテーブルを変更することが可能である。ChangeKeyboardMapping リクエストを用いることにより。一般にクライアントはこれをおこなってはならない。特に、これはキーバインディングやキーの再割り当てをクライアントが実装するべき方法ではない。

サーバから受け取られる一連のキーコードと特別なエンコーディングで表現される文字列の間の変換は、クライアントそれぞれについてプライベートな事項である。（それが in a world であるべきであるように。アプリケーションが異なるエンコーディングを使用して異なる言語とフォントをサポートしているかもしれない。）

キーボードイベントをテキストに変換するためには、Xlib のリファレンスを見よ。

ChangeKeyboardMapping リクエストを用いる正当な理由は唯一、キーにかかれているシンボルが変更されてしまうときだけである。例えばドボラク（Dvorak）キー変換キットや、APL のキーキャプス集合がインストールされたときのように。

もちろん、クライアントはキーキャプスへの変更を on trust に受け取らなければならない。

次のことはクライアントとユーザの間の permissible な対話を illustrate している。

クライアント:「あなたは Pause キーのないサーバで私をちょうど始めました。Pause キーとするべきキーを選び、それを今押してください。」

ユーザ:Scroll Lock キーを押す。

クライアント:「Scroll Lock キーのシンボルに Pause を加えます。Confirm または中止。」

ユーザ:Confirms。

クライアント:ChangeKeyboardMapping リクエストを使用して Pause を、すでに Scroll Lock をもつキーコードに加え、このリクエストを発行する。「Scroll Lock キーに Pause と描いてください。」

規約

クライアントは ChangeKeyboardMapping リクエストを利用しなてはならない。

クライアントがキーボードマッピングのテーブルを変更するのに成功すれば、すべてのクライアントは MappingNotify（request==Keyboard）イベントを受け取ることになる。このイベントを受け取ることを避けるためのメカニズムはいっさいない。

規約

MappingNotify（request==Keyboard）イベントを受け取るクライアントは、自分が使用している any 内部のキーコード変換テーブルを更新しなくてはならない。

6.6 モデファイヤマッピング

X バージョン 11 は 8 つのモデファイヤビットをサポートしている。そのうち 3 つは予め割り当てられている。Shift、Lock、Control に。

それぞれのモデファイヤビットはコントロールされる。キーのセットは指定される。GetModifierMapping と SetModifierMapping リクエストによって、アクセスされるテーブルにおいて。

このテーブルは共有リソースであり、規約が必要である。

予め割り当てられているモデファイヤのひとつを用いる必要のあるクライアントは、モデファイヤテーブルは正しくセットアップされていて、これらのモデファイヤをコントロールできるとみなさなければならない。

Lock モデファイヤは解釈されなければならない。CapsLock か ShiftLock として。

キーコード in its controlling set が XK_Caps_Lock か XK_Shift_Lock を含むように。according as。

規約

クライアントはモデファイヤビットの意味を決定しなければならない。それをコントロールするためには使用されているキーシムから。

エキストラなモデファイヤ（例えば、META）を用いる必要のあるクライアントは、次のことをしなければならない。

- 存在するモデファイヤマッピングをスキャンする。キーシムのセットが XK_Meta_L や XK_Meta_R を含むキーコードを保持するモデファイヤを見つける場合は、そのモデァイヤビットを使用しなければならない。
- XK_Meta_L か XK_Meta_R によってコントロールされるモディファイアがいっさい存在しない場合、クライアントは未使用のモディファイアビットを選び（空の controlling set をもつモディファイアビット）、次のことを行う。
  - そのキーシムセットに XL_Meta_L をもつキーコードがあれば、そのキーコードを選んだモディファイアビットに加える。
  - そのキーシムセットに XL_Meta_R をもつキーコードがあれば、そのキーコードを選んだモディファイアビットに加える。
  - controlling set が依然として空であれば、ユーザと対話して、META として使用するべき 1 つ以上のキーを選ぶ。
- 未使用のモディファイアビットがいっさいない場合は、corrective な行動をとるようにユーザに求める。

規約

1. 現在使用されていないモディファイアを必要とするクライアントは、適正なキーシムを carry しているキーコードを、未使用のモディファイアビットに割り当てなければならない。
2. 自分自身のモディファイアビットを割り当てるクライアントは、自分の SetModifierMapping リクエストが Busy ステータスを返す場合は、そのユーザに politely に問題となっているキーから手を離すように求めなければならない。

よい解決方法はない。問題に対する。5 つの割り当て済みモディファイアへの割り当ての reclaiming という。それらがもはや使用されていないときに。

規約

ユーザは xmodmap やなにかそのほかのユーテリティを使用して、時代遅れのモディファイアマッピングを by hand で deassign するべきである。

クライアントが SetModifierMapping リクエストの実行に成功するとき、すべてのクライアントは MappingNotify（request==Modifier）イベントを受け取ることになる。これらのイベントを受け取ることを妨げるメカニズムはない。割り当て済みでないモディファイアのひとつを使用するクライアントが、これらのイベントのひとつを受け取る場合は、GetModifierMapping リクエストで新しいマッピングを発見し、自分が使用しているモディファイアがクリアされていたら、そのクライアントはそのモディファイアを再インストールしなければならない。

GrabServer リクエストは使用されるべきであることに注意。GetModifierMapping と SetModifierMapping の組を atomic に make するために。これらのトランザクションにおける。

索引

-name, 29
ADOBE, 12
_ADOBE_EPS, 12
_ADOBE_EPSI, 12

ADOBE_PORTABLE_DOCUMENT_FORMAT, 12
AllowEvents, 57
Always, 48
AnyPropertyType, 8
Append, 9
APPLE_PICT, 12, 15
argv[0], 29
ATOM, 12, 15, 30, 54
ATOM_PAIR, 12, 14, 15

BACKGROUND, 12, 18
BITMAP, 12, 15, 17
BlackPixel, 41
ButtonPress, 36
ButtonRelease, 36

C_STRING, 15, 16
Center, 25
ChangeProperty, 8, 9, 23
ChangeWindowAttributes, 37
CHARACTER_POSITION, 12, 18
CirculateWindow, 48
CLASS, 12
CLIENT_WINDOW, 12
ClientMessage, 19, 30, 33, 34, 40, 42, 43, 45, 51, 52
CLIPBOARD, 10, 11
COLORMAP, 12, 15, 18
ColormapChange, 42, 50
ColormapNotify, 51
COLUMN_NUMBER, 12
COMPOUND_TEXT, 12, 15, 16
ConfigureNotify, 35–37, 48–50
ConfigureRequest, 37
ConfigureWindow, 35, 37, 46, 48
ConvertSelection, 6, 7, 9, 11, 13–15
CopyFromParent, 41
CreateWindow, 46
CurrentTime, 2–4, 6, 39, 40, 51, 55

DELETE, 12–14
Deleted, 17
DeleteProperty, 8
DeleteWindow, 53
DRAWABLE, 12, 15, 17

East, 25
ENCAPSULATED_POSTSCRIPT, 12
ENCAPSULATED_POSTSCRIPT_INTERCHANGE, 12
EnterNotify, 36, 39, 51
Expose, 48
Exposure, 44

False, 19, 22, 27, 33, 37, 39, 42, 50–52, 55
FILE_NAME, 12
FocusChange, 51
FocusIn, 39, 51
FocusOut, 51
FOREGROUND, 12, 18
FreeColormap, 58
FullyObscured, 35

GetGeometry, 17, 47
GetProperty, 7–9, 17
GetSelectionOwner, 2, 13, 18, 19, 54

HOST_NAME, 12

IconicState, 27, 31–34
IconMaskHint, 26

IconPixmapHint, 26
IconPositionHint, 26
IconWindowHint, 26
INCR, 9, 15, 17
INCREMENTAL, 17
InputHint, 26
InputOutput, 44
INSERT_PROPERTY, 12, 15
INSERT_SELECTION, 12, 14, 15
InstallColormap, 42, 43
INTEGER, 12, 15, 54
InternAtom, 14

KeyPress, 36
KeyRelease, 36
kill_id, 58
KillClient, 58

LeaveNotify, 36
LENGTH, 12
LINE_NUMBER, 12, 18
LIST_LENGTH, 12

MANAGER, 19
MapNotify, 33, 50
MapWindow, 48, 49
MessageHint, 26, 28
MODULE, 12
MotionNotify, 36
MULTIPLE, 5, 7, 12–14, 18

NAME, 12
NewValue, 17
None, 2–4, 6–8, 13, 15, 32, 37, 40, 47, 51
NormalState, 27, 31–33
North, 25
NorthEast, 25
NorthWest, 25
NULL, 12, 14

ODIF, 12
OWNER_OS, 12

Parent, 40, 41, 51
ParentRelative, 47
PAspect, 25
PBaseSize, 25
PIXEL, 12, 15, 18
PIXMAP, 12, 15, 17
PMaxSize, 25
PMinSize, 25
PointerRoot, 38–40, 51
PolyLine, 48
POSTSCRIPT, 12
PPosition, 25
PResizeInc, 25
PRIMARY, 3, 9, 10, 14
PROCEDURE, 12
PROCESS, 12
PropertyNotify, 17
PropertyNotify, 2, 5, 17
PSize, 25
PWinGravity, 25

QueryTree, 46

ReparentNotify, 34, 48
Replace, 23
ReplayPointer/Keyboard, 57
ResizeRedirect, 44, 53
ResizeRequest, 49, 53
RESOURCE_NAME, 29
RGB_COLOR_MAP, 57, 58

SECONDARY, 9, 10, 14
SelectionRequest, 4
SelectionClear, 2, 5, 6, 11
SelectionNotify, 3–5, 7–9, 13, 17
SelectionRequest, 2–4, 13
SendEvent, 3, 4, 19, 33, 34, 37, 42, 43, 50, 52
SetInputFocus, 39–41, 49, 51, 55
SetSelectionOwner, 2, 3, 5, 6, 9
South, 25
SouthEast, 25
SouthWest, 25
SPAN, 12, 15, 18
StateHint, 26
Static, 25
STRING, 12, 15, 16, 29, 54

StructureNotify, 19, 33, 36, 37, 48, 50
SubstructureNotify, 33
SubstructureRedirect, 19, 33, 37, 53
`Synchronous`, 57

TARGETS, 12, 13
TASK, 12
TEXT, 12, 16, 30, 54
TIMESTAMP, 11–13
TranslateCoordinates, 36
`True`, 8, 17, 27, 39, 51, 55

`Unmap`, 25
UnmapNotify, 33, 34, 50
`UrgencyHint`, 26, 28
USER, 12
`USPosition`, 25
`USSize`, 25

VERSION, 54
VisibilityChange, 35, 42, 49
VisibilityNotify, 35, 49

WarpPointer, 56
`West`, 25
`WhitePixel`, 41
WINDOW, 12, 15, 17, 29, 30, 54
`WindowGroupHint`, 26
`WithdrawnState`, 31, 32
WM_CHANGE_STATE, 34
WM_CLASS, 29, 45, 54
WM_CLIENT_MACHINE, 30, 45, 54
WM_COLORMAP_NOTIFY, 42
WM_COLORMAP_WINDOWS, 30, 41, 42, 45, 51, 54
WM_COMMAND, 45
WM_DELETE_WINDOW, 30, 52, 53
WM_HINTS, 26–28, 31–33, 39, 43–46, 51, 54, 55
WM_ICON_NAME, 24, 32, 43–45, 54
WM_ICON_SIZE, 27, 32, 43, 44, 54
WM_NAME, 24, 26, 45, 54
WM_NORMAL_HINTS, 24, 26, 45, 54
WM_PROTOCOLS, 23, 28, 30, 39, 40, 45, 51–54
WM_S*n*, 54
WM_S2, 19
WM_SAVE_YOURSELF, 30
WM_SIZE_HINTS, 24, 25, 54
WM_STATE, 31, 32, 34, 54
WM_TAKE_FOCUS, 30, 39, 40, 45, 51, 55
WM_TRANSIENT_FOR, 29, 30, 42, 45, 54
WM_WINDOW_DELETE, 53

xload, 38
xprop, 31
xterm, 5