インターネット・ユーティリティ・クラスの使用法

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

インターネット・ユーティリティ・クラスの使用法

InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

テーブル一覧

1

HTTP 要求の送信

ここでは、HTTP 要求 (POST や GET など) の送信方法について説明します。

1.1 HTTP 要求の概要

%Net.HttpRequest のインスタンスを作成して、さまざまな種類の HTTP 要求を送信し、応答を受信します。このオブジェクトは、Web ブラウザと同等であり、このオブジェクトを使用して、複数の要求を行うことができます。自動的に正しい Cookie が送信され、必要に応じて `Referer` ヘッダが設定されます。

HTTP 要求を作成するには、以下の一般的なプロセスを使用します。

1. %Net.HttpRequest のインスタンスを作成します。
2. このインスタンスのプロパティを設定し、通信する Web サーバを指定します。基本的なプロパティは以下のとおりです。
   - Server では、Web サーバの IP アドレスまたはマシン名を指定します。既定値は `localhost` です。

     注釈　Server の値の一部として `http://` や `https://` を含めないでください。これらを含めると、「`#6059 http:/ TCP/IP` 」が発生します。

   - Port では、接続先のポートを指定します。既定値は 80 です。
3. 必要に応じて、"HTTP 要求のその他のプロパティの指定" で説明されているとおり、HTTP 要求のその他のプロパティと呼び出しメソッドを設定します。
4. その後、"HTTP 要求の送信" で説明されているとおり、%Net.HttpRequest のインスタンスの Get() メソッドまたはその他のメソッドを呼び出し、HTTP 要求を送信します。

   インスタンスから複数の要求を行うことができ、それによって、Cookie および `Referer` ヘッダが自動的に処理されます。

   注釈　プロダクションの発信アダプタ (EnsLib.HTTP.OutboundAdapter) で使用するためにこの HTTP 要求を作成した場合は、代わりにそのアダプタのメソッドを使用して要求を送信します。
5. %Net.HttpRequest の同じインスタンスを使用して、必要に応じて追加の HTTP 要求を送信します。既定により、InterSystems IRIS® データ・プラットフォームは TCP/IP ソケットのオープン状態を維持するので、ソケットを閉じてから再び開くことなく、再利用できます。

   詳細は、"ソケット再利用の管理" を参照してください。

以下に簡単な例を示します。

ObjectScript

```
set request=##class(%Net.HttpRequest).%New()
set request.Server="tools.ietf.org"
set request.Https=1
set request.SSLConfiguration="TEST"
set status=request.Get("/html/rfc7158")
```

Https プロパティと SSLConfiguration プロパティの詳細は、このトピックで後述する "SSL を使用した接続" を参照してください。また、この例のより完全なバージョンは、"HTTP 応答へのアクセス" を参照してください。

1.2 認証の指定

宛先サーバがログイン資格情報を必要とする場合、資格情報を提供する HTTP Authorization ヘッダを HTTP 要求に含めることができます。以降の項で詳しく説明します。

- HTTP 1.0 を使用する場合の要求の認証
- HTTP 1.1 を使用する場合の要求の認証
- Authorization ヘッダを直接指定する方法
- HTTP 認証のログの有効化

プロキシ・サーバを使用している場合、プロキシ・サーバに対してもログイン資格情報を指定できます。そのためには、ProxyAuthorization プロパティを設定します。"プロキシ・サーバの使用法" を参照してください。詳細は、%Net.HttpRequest のクラス・ドキュメントを参照してください。

1.2.1 HTTP 1.0 を使用する場合の要求の認証

HTTP 1.0 の場合に HTTP 要求を認証するには、%Net.HttpRequest.のインスタンスの Username プロパティと Password プロパティを設定します。このインスタンスでは次に、Basic アクセス認証 (RFC 2617) を使用して、そのユーザ名とパスワードに基づいて HTTP Authorization ヘッダが作成されます。この %Net.HttpRequest から送信される以降の要求に、このヘッダが含まれるようになります。

重要　SSL も使用していることを確認してください ("SSL を使用した接続" を参照)。Basic 認証では、資格情報は Base 64 のエンコード形式で送信されるため、簡単に読み取ることができます。

1.2.2 HTTP 1.1 を使用する場合の要求の認証

HTTP 1.1 の場合に HTTP 要求を認証するには、ほとんどの場合、%Net.HttpRequest.のインスタンスの Username プロパティと Password プロパティを設定するだけです。%Net.HttpRequest のインスタンスが 401 HTTP ステータス・コードと WWW-Authenticate ヘッダを受け取ると、このインスタンスは、サポートされている認証スキームが含まれる Authorization ヘッダを使用して応答しようとします。サポートされており InterSystems IRIS 向けに構成されている最初のスキームが使用されます。既定では、以下の認証スキームがこの順序で検討されます。

1. Negotiate (RFC 4559 および RFC 4178 で説明されている SPNEGO および Kerberos)
2. NTLM (NT LAN Manager 認証プロトコル)
3. Basic (RFC 2617 で説明されている Basic アクセス認証)

重要　Basic 認証が使用される可能性がある場合、SSL も使用していることを確認してください ("SSL を使用した接続" を参照)。Basic 認証では、資格情報は Base 64 のエンコード形式で送信されるため、簡単に読み取ることができます。

Windows では、Username プロパティが指定されていない場合、InterSystems IRIS は代わりに現在のログイン・コンテキストを使用できます。具体的には、SPNEGO、Kerberos、または NTLM の場合にサーバが 401 ステータス・コードと `WWW-Authenticate` ヘッダを使用して応答すると、InterSystems IRIS は現在のオペレーティング・システムのユーザ名とパスワードを使用して、`Authorization` ヘッダを作成します。

以下に示すように、HTTP 1.0 の場合とは詳細が異なります。

- 認証が成功すると、最新の認証に使用された認証スキームを示すように、InterSystems IRIS は %Net.HttpRequest インスタンスの CurrentAuthenticationScheme プロパティを更新します。
- スキームの認証ハンドルまたはトークンを取得しようとして失敗した場合、InterSystems IRIS は原因となっているエラーを %Net.HttpRequest インスタンスの AuthenticationErrors プロパティに保存します。このプロパティの値は $LIST で、各項目が scheme `ERROR:` message の形式になっています。

Negotiate と NTLM は HTTP 1.1 でのみサポートされています。これらのスキームには複数回の往復が必要ですが、HTTP 1.0 では要求と応答の各ペアの後に接続を閉じる必要があるからです。

1.2.2.1 バリエーション

サーバで許可される認証スキームがわかっている場合は、選択したスキームのサーバ用の初期トークンが含まれる `Authorization` ヘッダを含めることによって、サーバからの最初の往復をバイパスできます。このためには、%Net.HttpRequest インスタンスの InitiateAuthentication プロパティを設定します。このプロパティの値として、サーバで許可されている 1 つの認証スキームの名前を指定します。以下の値のいずれかを使用します (大文字と小文字が区別されます)。

- `Negotiate`
- `NTLM`
- `Basic`

使用する認証スキームをカスタマイズする (または検討される順序を変更する) 場合は、%Net.HttpRequest インスタンスの AuthenticationSchemes を設定します。このプロパティの値として、認証スキーム名のコンマ区切りリストを指定します (前述のリストで示している正確な値を使用します)。

1.2.3 Authorization ヘッダを直接指定する方法

HTTP 1.0 と HTTP 1.1 (シナリオに適用できる場合) のいずれの場合でも、HTTP `Authorization` ヘッダを直接指定できます。具体的には、要求しているリソースのユーザ・エージェントに必要な認証情報に等しい値に Authorization プロパティを設定します。

Authorization プロパティを指定すると、Username プロパティと Password プロパティは無視されます。

1.2.4 HTTP 認証のログの有効化

HTTP 認証のログを有効にするには、ターミナルで以下を入力します。

```
set $namespace="%SYS"
kill ^ISCLOG
set ^%ISCLOG=2
set ^%ISCLOG("Category","HttpRequest")=5
```

ログ・エントリは、^ISCLOG グローバルに書き込まれます。ログをファイルに書き込むには (読みやすくするため)、以下を入力します (引き続き、%SYS ネームスペース内です)。

```
do ##class(%OAuth2.Utils).DisplayLog("filename")
```

filename は、作成するファイルの名前です。このディレクトリは既に存在している必要があります。ファイルが既に存在する場合は、そのファイルが上書きされます。

ログを停止するには、以下を入力します (引き続き、%SYS ネームスペース内です)。

```
set ^%ISCLOG=0
set ^%ISCLOG("Category","HttpRequest")=0
```

1.3 HTTP 要求のその他のプロパティの指定

HTTP 要求を送信する前に ("HTTP 要求の送信" を参照)、以下のセクションの説明に従って、プロパティを指定することができます。

- Location プロパティ
- インターネット・メディア・タイプと文字エンコードの指定
- プロキシ・サーバの使用法
- SSL を使用した接続
- HTTPVersion、Timeout、WriteTimeout、および FollowRedirect プロパティ
- HTTP 要求の既定値の指定

最後にリストされているセクションで説明しているとおり、%Net.HttpRequest のすべてのプロパティの既定値を指定できます。

1.3.1 Location プロパティ

Location プロパティでは、Web サーバから要求するリソースを指定します。このプロパティを設定すると、Get()、Head()、Post()、または Put() の各メソッドを呼び出す際、場所の引数を省略できます。

例えば、URL `http://machine_name/test/index.html` に HTTP 要求を送信しているとします。

この場合、以下の値を使用します。

テーブル 1-1: %Net.HttpRequest のプロパティの例

プロパティ	値
Server	machine_name
Location	test/index.html

1.3.2 インターネット・メディア・タイプと文字エンコードの指定

以下のプロパティを使用して、インターネット・メディア・タイプ (MIME タイプ とも呼ばれます)、および %Net.HttpRequest のインスタンスとそのレスポンス内の文字エンコードを指定できます。

- ContentType では、Content-Type ヘッダを指定します。このヘッダで、要求の本文のインターネット・メディア・タイプを指定します。既定タイプは、None です。

  指定可能な値は、application/json、application/pdf、application/postscript、image/jpeg、image/png、multipart/form-data、text/html、text/plain、text/xml などの他にも多数あります。詳細なリストは、http://www.iana.org/assignments/media-types を参照してください。

- ContentCharset プロパティは、コンテンツがテキスト・タイプ (text/html や text/xml など) の場合、要求のコンテンツのための目的の文字セットを制御します。このプロパティを指定しない場合、InterSystems IRIS では、InterSystems IRIS サーバ既定のエンコーディングが使用されます。

  注釈　このプロパティを設定する場合、まず、ContentType プロパティを設定する必要があります。

- NoDefaultContentCharset プロパティは、ContentCharset プロパティを設定していない場合、text タイプのコンテンツの明示的な文字セットを含めるかどうかを制御します。既定では、このプロパティは、False です。

  このプロパティが True の場合、text タイプのコンテンツがあり、ContentCharset プロパティが設定されていなければ、コンテンツ・タイプに文字セットは含まれません。つまり、iso-8859-1 文字セットがメッセージの出力に使用されます。

- WriteRawMode プロパティは、エンティティ本文に影響を与えます (含まれる場合)。これは要求の本文の記述方法を制御します。既定では、このプロパティは False であり、要求ヘッダで指定されたエンコーディングで本文に記述されます。このプロパティが True の場合、本文が RAW モードで記述されます (文字セットの変換は行いません)。

- ReadRawMode プロパティは、応答の本文の読み取り方法を制御します。既定では、このプロパティは False であり、本文が応答ヘッダで指定された文字セットであると見なされます。このプロパティが True の場合、本文が RAW モードで読み取られます (文字セットの変換は行いません)。

1.3.3 プロキシ・サーバの使用法

プロキシ・サーバにより、HTTP 要求を送信できます。これを設定するには、HTTP 要求の以下のプロパティを指定します。

- ProxyServer では、使用するプロキシ・サーバのホスト名を指定します。このプロパティが NULL でない場合、HTTP 要求はこのマシンに向けられます。

- ProxyPort では、プロキシ・サーバ上の接続するポートを指定します。

- ProxyAuthorization では、Proxy-Authorization ヘッダを指定します。ユーザ・エージェントがそれ自体をプロキシで認証する必要がある場合に設定する必要があります。値には、要求しているリソースのユーザ・エージェントが必要とする認証情報を使用します。"ログイン資格情報の提供" も参照してください。

- ProxyHTTPS は、HTTP 要求が通常の HTTP ページではなく、HTTPS ページに対するものであるかどうかを制御します。このプロパティは、プロキシ・サーバを指定していない場合、無視されます。このプロパティでは、ターゲット・システムの既定ポートをプロキシ・ポートである 443 に変更します。"SSL を使用した接続" も参照してください。

- ProxyTunnel では、プロキシ経由でターゲットの HTTP サーバへのトンネルを確立するかどうかを指定します。True の場合、要求は HTTP CONNECT コマンドを使用してトンネルを確立します。プロキシ・サーバのアドレスは、ProxyServer プロパティと ProxyPort プロパティから取得されます。ProxyHttps が True の場合、トンネルが確立されたときに InterSystems IRIS が SSL 接続をネゴシエートします。この場合、トンネルによってターゲット・システムとの直接接続が確立されるため、Https プロパティは無視されます。

詳細は、%Net.HttpRequest のクラス・ドキュメントを参照してください。

1.3.4 SSL を使用した接続

%Net.HttpRequest クラスは、SSL 接続をサポートします。SSL によって要求を送信する手順は以下のとおりです。

1. SSLConfiguration プロパティを使用する有効化した SSL/TLS 構成の名前に設定します。

   SSL/TLS 構成の作成と管理の詳細は、"InterSystems TCP ガイド" を参照してください。SSL/TLS 構成には、[構成名] と呼ばれるオプションが含まれています。これは、この設定で使用する文字列です。

2. プロキシ・サーバを使用しているかどうかによって、以下のいずれかを行います。

   - プロキシ・サーバを使用していない場合、Https プロパティを True に設定します。
   - プロキシ・サーバを使用している場合、ProxyHTTPS プロパティを True に設定します。

     この場合、プロキシ・サーバ自体への SSL 接続を使用するため、Https プロパティを True に設定します。

指定のサーバへの SSL 接続を使用する場合、そのサーバでの既定ポートは 443 (HTTPS ポート) と見なされます。例えば、プロキシ・サーバを使用しておらず、Https が True の場合、これによって、既定の Port プロパティが 443 に変更されます。

"プロキシ・サーバの使用法" も参照してください。

1.3.4.1 サーバ ID の確認

既定では、%Net.HttpRequest のインスタンスは、SSL/TLS でセキュリティ保護された Web サーバに接続するときに、証明書サーバ名と、そのサーバへの接続に使用した DNS 名が一致しているかどうかを確認します。これらの名前が一致していないと、接続は許可されません。この既定の動作は、“中間者” 攻撃を防止するもので、RFC 2818 のセクション 3.1 に説明があります。また、RFC 2595 のセクション 2.4 も参照してください。

この確認を無効にするには、SSLCheckServerIdentity プロパティを 0 に設定します。

1.3.5 HTTPVersion、Timeout、WriteTimeout、および FollowRedirect プロパティ

%Net.HttpRequest には、以下のプロパティもあります。

- HTTPVersion は、ページを要求するときに使用する HTTP のバージョンを指定します。既定値は "HTTP/1.1" です。また、"HTTP/1.0" も使用できます。
- Timeout では、Web サーバからの応答を待機する時間を秒単位で指定します。既定値は 30 秒です。
- WriteTimeout では、Web サーバの書き込み完了を待機する時間を秒単位で指定します。既定では無限に待機します。指定できる最小値は 2 秒です。
- FollowRedirect では、(300 ～ 399 の範囲の HTTP 状態コードによって示される) Web サーバからのリダイレクト要求に自動的に従うかどうかを指定します。既定値は、GET または HEAD を使用している場合は True、それ以外の場合は False です。

1.3.6 HTTP 要求の既定値の指定

%Net.HttpRequest のすべてのプロパティの既定値を指定できます。

- すべてのネームスペースに適用する既定値を指定するには、グローバル・ノード ^%SYS("HttpRequest","propname") を設定します。"propname" は、プロパティの名前です。
- 1 つのネームスペースに適用する既定値を指定するには、そのネームスペースに移動し、ノード ^SYS("HttpRequest","propname") を設定します。

(^%SYS グローバルはインストール全体に、^SYS グローバルは現在のネームスペースに影響を及ぼします。)

例えば、すべてのネームスペースに既定のプロキシ・サーバを指定するには、グローバル・ノード ^%SYS("HttpRequest","ProxyServer") を設定します。

1.4 HTTP ヘッダの設定と取得

HTTP ヘッダの値を設定および取得できます。

%Net.HttpRequest の以下の各プロパティには、対応する名前を持つ HTTP ヘッダの値が含まれます。これらのプロパティを設定しなかった場合、これらは自動的に計算されます。

- Authorization
- ContentEncoding
- ContentLength (このプロパティは読み取り専用です。)
- ContentType (Content-Type ヘッダのインターネット・メディア・タイプ (MIME タイプ) を指定します。)
- ContentCharset (Content-Type ヘッダの charset 部分を指定します。これを設定する場合、まず、ContentType プロパティを設定する必要があります。)
- Date
- From
- IfModifiedSince
- Pragma
- ProxyAuthorization
- Referer
- UserAgent

%Net.HttpRequest クラスは、主要 HTTP ヘッダの設定および取得に使用できる一般的なメソッドを提供します。これらのメソッドは、Content-Type およびその他のエンティティ・ヘッダを無視します。

ReturnHeaders()

この要求の主要 HTTP ヘッダを含む文字列を返します。

OutputHeaders()

現在のデバイスに主要 HTTP ヘッダを書き込みます。

GetHeader()

この要求に設定されているすべての主要 HTTP ヘッダの現在の値を取得します。このメソッドは、1 つの引数、すなわちヘッダ名を取ります (大文字と小文字を区別しない)。これは、Host や Date などの文字列です。

SetHeader()

ヘッダの値を設定します。通常は、これを使用して、標準以外のヘッダを設定します。通常のほとんどのヘッダは、Date などのプロパティによって設定されます。このメソッドは、以下の 2 つの引数を取ります。

1. コロン (:) の区切り文字のないヘッダの名前 (大文字と小文字を区別しない)。これは、Host や Date などの文字列です。
2. そのヘッダの値

このメソッドを使用して、エンティティ・ヘッダまたは読み取り専用のヘッダ (Content-Length および Connection) を設定することはできません。

詳細は、%Net.HttpRequest のクラス・ドキュメントを参照してください。

1.5 キープ・アライブ動作の管理

%Net.HttpRequest の同じインスタンスを再利用して、複数の HTTP 要求を送信する場合、既定により、InterSystems IRIS は TCP/IP ソケットのオープン状態を維持するので、閉じてから再び開く必要はありません。

TCP/IP ソケットを再利用する必要がない場合、SocketTimeout プロパティを 0 に設定します。

%Net.HttpRequest の SocketTimeout プロパティにより、指定されたソケットを InterSystems IRIS で再利用する予定期間の時間ウィンドウを秒単位で指定します。このタイムアウトには、ファイアウォールにより密かに閉じられた可能性のあるソケットの使用を避ける目的があります。このプロパティの既定値は 115 です。このプロパティには、別の値を設定できます。

1.6 HTTP 要求のパラメータの処理

HTTP 要求を送信する際 ("HTTP 要求の送信" を参照)、パラメータを location 引数で指定できます。例えば、"/test.html?PARAM=%25VALUE" は、PARAM を %VALUE に設定します。

以下のメソッドを使用して、%Net.HttpRequest のインスタンスがパラメータを処理する方法を制御することもできます。

InsertParam()

要求にパラメータを挿入します。このメソッドは、パラメータ名とパラメータの値の 2 つの文字列引数を取ります。以下はその例です。

ObjectScript

```
do req.InsertParam("arg1","1")
```

指定のパラメータに対して、複数の値を挿入できます。その場合、値は、1 で始まる添え字を受け取ります。その他のメソッド内で、これらの添え字を使用して、目的の値を参照します。

DeleteParam()

要求からパラメータを削除します。最初の引数は、パラメータの名前です。2 番目の引数は、削除する値の添え字です。同じパラメータに対して複数の値が要求に含まれている場合にのみこれを使用します。

CountParam()

指定のパラメータに関連付けられた値の数を数えます。

GetParam()

要求内の指定のパラメータの値を取得します。最初の引数は、パラメータの名前です。2 番目の引数は、要求にこの名前のパラメータがない場合に返す既定値です。この既定値の初期値は、NULL 値です。3 番目の引数は、取得する値の添え字です。同じパラメータに対して複数の値が要求に含まれている場合にのみこれを使用します。

IsParamDefined()

指定のパラメータが定義されているかどうかを確認します。このメソッドは、パラメータに値がある場合、True を返します。この引数は、DeleteParam() のものと同じです。

NextParam()

$Order() によってパラメータ名を並べ替えた後で、次にパラメータがある場合は、その名前を取得します。

ReturnParams()

この要求に含まれるパラメータのリストを返します。

詳細は、%Net.HttpRequest のクラス・ドキュメントを参照してください。

1.7 要求の本文を含める

HTTP 要求には、要求の本文またはフォーム・データのいずれかを含めることができます。要求の本文を含める手順は以下のとおりです。

1. %GlobalBinaryStream のインスタンスまたはサブクラスを作成します。HTTP 要求の EntityBody プロパティにはこのインスタンスを使用します。

   %GlobalBinaryStream は非推奨になっていますが、このような使用法は引き続きサポートされています。このユース・ケースで別のストリーム・クラスに置き換えることはお勧めできません。

2. データをこのストリームに書き込むには、標準のストリーム・インタフェースを使用します。以下はその例です。

```
ObjectScript

 Do oref.EntityBody.Write("Data into stream")
```

例えば、ファイルを読み取り、それをカスタム HTTP 要求のエンティティ本文として使用できます。

```
ObjectScript

set file=##class(%File).%New("G:\customer\catalog.xml")
set status=file.Open("RS")
if $$$ISERR(status) do $System.Status.DisplayError(status)
set hr=##class(%Net.HttpRequest).%New()
do hr.EntityBody.CopyFrom(file)
do file.Close()
```

1.7.1 チャンクされた要求の送信

HTTP 1.1 を使用すると、HTTP要求をチャンクで送信できます。これには、Transfer-Encoding を設定してメッセージがチャンクされていることを示し、0 サイズのチャンクを使用して完了を示す必要があります。

チャンク・エンコードは、サーバが大量のデータを返しており、要求が完全に処理されるまで応答の合計サイズが分からない場合に便利です。そのような場合、通常、コンテンツの長さが計算できるまで (これは、%Net.HttpRequest が自動的に行います)、メッセージ全体をバッファする必要があります。

チャンクされた要求を送信する手順は以下のとおりです。

1. データをチャンクで書き込むためのインタフェースを定義する抽象ストリーム・クラスである、%Net.ChunkedWriter のサブクラスを作成します。

   このサブクラス内に、OutputStream() メソッドを実装します。

2. %Net.HttpRequest のインスタンスで、%Net.ChunkedWriter サブクラスのインスタンスを作成し、送信する要求データを入力します。

3. %Net.HttpRequest インスタンスの EntityBody プロパティを %Net.ChunkedWriter のこのインスタンスと等しくなるように設定します。

   HTTP 要求を送信すると ("HTTP 要求の送信" を参照)、EntityBody プロパティの OutputStream() メソッドが呼び出されます。

%Net.ChunkedWriter のサブクラスで、OutputStream() メソッドはストリーム・データを調べ、これをチャンクするかどうかとその方法を決定し、クラスの継承されたメソッドを呼び出して出力を記述します。

以下のメソッドを使用できます。

WriteSingleChunk()

文字列の引数を受け入れ、この文字列を非チャンク出力として記述します。

WriteFirstChunk()

文字列引数を受け入れます。適切な Transfer-Encoding ヘッダを記述してチャンクされたメッセージを示し、続いて、その文字列を最初のチャンクとして記述します。

WriteChunk()

文字列の引数を受け入れ、この文字列をチャンクとして記述します。

WriteLastChunk()

文字列の引数を受け入れ、文字列をチャンクとして記述し、続いて、終了を表す長さが 0 のチャンクを記述します。

NULL でない場合、TranslateTable プロパティは、記述する際に各文字列の変換に使用する変換テーブルを指定します。前述のメソッドはすべてこのプロパティを確認します。

1.8 フォーム・データの送信

HTTP 要求には、要求の本文またはフォーム・データのいずれかを含めることができます。フォーム・データを含めるには、以下のメソッドを使用します。

InsertFormData()

要求にフォーム・データを挿入します。このメソッドは、フォーム項目名と関連付けられた値の 2 つの文字列引数を取ります。指定のフォーム項目に複数の値を挿入できます。その場合、値は、1 で始まる添え字を受け取ります。その他のメソッド内で、これらの添え字を使用して、目的の値を参照します。

DeleteFormData()

要求からフォーム・データを削除します。最初の引数は、フォーム項目の名前です。2 番目の引数は、削除する値の添え字です。同じフォーム項目に対して複数の値が要求に含まれている場合にのみこれを使用します。

CountFormData()

要求内の指定の名前に関連付けられた値の数を数えます。

IsFormDataDefined()

指定の名前が定義されているかどうかを確認します。

NextFormData()

$Order() によってパラメータ名を並べ替えた後で、次にフォーム項目がある場合は、その名前を取得します。

これらのメソッドの詳細は、%Net.HttpRequest のクラス・ドキュメントを参照してください。

1.8.1 例 1

フォーム・データを挿入後、通常、Post() メソッドを呼び出します。以下はその例です。

ObjectScript

```
Do httprequest.InsertFormData("element","value")
Do httprequest.Post("/cgi-bin/script.CGI")
```

1.8.2 例 2

別の例を示します。

ObjectScript

```
Set httprequest=##class(%Net.HttpRequest).%New()
set httprequest.SSLConfiguration="MySSLConfiguration"
set httprequest.Https=1
set httprequest.Server="myserver.com"
set httprequest.Port=443
Do httprequest.InsertFormData("portalid","2000000")
set tSc = httprequest.Post("/url-path/")
Quit httprequest.HttpResponse
```

1.9 Cookie の挿入、リスト、および削除

%Net.HttpRequest は、サーバから送信された cookie を自動的に管理します。サーバが cookie を送信すると、%Net.HttpRequest のインスタンスが次の要求でこの cookie を返します (このメカニズムが機能するためには、%Net.HttpRequest の同じインスタンスを再利用する必要があります)。

%Net.HttpRequest のインスタンス内で cookie を管理するには、以下のメソッドを使用します。

InsertCookie()

要求に cookie を挿入します。以下の引数を指定します。

1. cookie 名。
2. cookie の値。

3. cookie を保存するパス。
4. cookie のダウンロード元のマシン名。
5. cookie の期限が切れる日付と時刻。

GetFullCookieList()

cookie の数と、cookie の配列を (参照によって) 返します。

DeleteCookie()

要求から cookie を削除します。

cookie は、HTTP サーバに固有であることに留意してください。cookie を挿入する際は、特定のサーバへの接続を使用しており、その cookie は他のサーバでは使用できません。

これらのメソッドの詳細は、%Net.HttpRequest のクラス・ドキュメントを参照してください。

1.10 HTTP 要求の送信

HTTP 要求を作成した後、以下のメソッドのいずれかを使用してこの HTTP 要求を送信します。

Delete()

```
method Delete(location As %String = "",
              test As %Integer = 0,
              reset As %Boolean = 1) as %Status
```

HTTP DELETE 要求を発行します。

Get()

```
method Get(location As %String = "",
           test As %Integer = 0,
           reset As %Boolean = 1) as %Status
```

HTTP GET 要求を発行します。このメソッドにより、Web サーバは要求されたページを返します。

Head()

```
method Head(location As %String,
            test As %Integer = 0,
            reset As %Boolean = 1) as %Status
```

HTTP HEAD 要求を発行します。このメソッドにより、Web サーバは応答のヘッダのみを返します。本文は返しません。

Patch()

```
method Patch(location As %String = "",
             test As %Integer = 0,
             reset As %Boolean = 1) as %Status
```

HTTP PATCH 要求を発行します。既存のリソースに部分的な変更を加える場合にこのメソッドを使用します。

Post()

```
method Post(location As %String = "",
            test As %Integer = 0,
            reset As %Boolean = 1) as %Status
```

HTTP POST 要求を発行します。フォームの結果などのデータを Web サーバに送信したり、ファイルをアップロードするには、このメソッドを使用します。使用例は、"フォーム・データの送信" を参照してください。

Put()

```
method Put(location As %String = "",
           test As %Integer = 0,
           reset As %Boolean = 1) as %Status
```

HTTP PUT 要求を発行します。データを Web サーバにアップロードするには、このメソッドを使用します。PUT 要求は通常ありません。

Send()

```
method Send(type As %String,
            location As %String,
            test As %Integer = 0,
            reset As %Boolean = 1) as %Status
```

指定されたタイプの HTTP 要求をサーバに送信します。このメソッドは、通常、他のメソッドによって呼び出されますが、異なる HTTP 動詞を使用したい場合に使用するために用意されています。ここでは、type は、"POST" などの HTTP 動詞を指定する文字列です。

いずれの場合も、

- 各メソッドはステータスを返すので、それを確認する必要があります。
- メソッドが正しく完了すると、この要求に対する応答が HttpResponse プロパティに追加されます。
- location 引数は、要求する URL です。例えば、"/test.html" です。
- location 引数では、既に URL エスケープされていると見なすことができるパラメータを指定できます。例えば、"/test.html?PARAM=%25VALUE" は、PARAM を %VALUE に設定します。
- 送信する予定のものを送信していることを確認するには、test 引数を使用します。
  - test が 1 の場合、メソッドは、リモート・マシンに接続するのではなく、単に Web サーバに送信したはずのものを現在のデバイスに出力します。
  - test が 2 の場合、HTTP 要求を発行後、応答を現在のデバイスに出力します。
- 各メソッドは、test=1 または、reset=0 の場合を除いて、応答をサーバから読み取った後に、自動的に Reset() メソッドを呼び出します。

  Reset() メソッドは、別の要求を発行できるように、%Net.HttpRequest インスタンスをリセットします。これは、このオブジェクトを閉じて、新規インスタンスを作成するよりもはるかに速いです。また、これは、Location ヘッダの値を Referer ヘッダに移動します。

以下はその例です。

ObjectScript

```
Set httprequest=##class(%Net.HttpRequest).%New()
Set httprequest.Server="www.intersystems.com"
Do httprequest.Get("/")
```

他の例については、%Net.HttpRequest のクラス・ドキュメントを参照してください。

1.11 マルチパート POST 要求の作成と送信

マルチパート POST 要求の作成と送信を行うには、%Net.MIMEPart クラスを使用します。詳細は、このドキュメントで後述します。以下の例では、2 つの部分を有する POST 要求を送信します。最初の部分はファイルのバイナリ・データ、2 番目の部分はファイル名を含んでいます。

Class Member

```
ClassMethod CorrectWriteMIMEMessage3(header As %String)
{
     // Create root MIMEPart
     Set RootMIMEPart=##class(%Net.MIMEPart).%New()

     //Create binary subpart and insert file data
     Set BinaryMIMEPart=##class(%Net.MIMEPart).%New()
     Set contentdisp="form-data; name="_$CHAR(34)_"file"_$CHAR(34)_"; filename="
                     _$CHAR(34)_"task4059.txt"_$CHAR(34)
     Do BinaryMIMEPart.SetHeader("Content-Disposition",contentdisp)

     Set stream=##class(%FileBinaryStream).%New()
     Set stream.Filename="/home/tabaiba/prueba.txt"
     Do stream.LinkToFile("/home/tabaiba/prueba.txt")

     Set BinaryMIMEPart.Body=stream
     Do BinaryMIMEPart.SetHeader("Content-Type","text/plain")

    // Create text subpart
    Set TextMIMEPart=##class(%Net.MIMEPart).%New()
    Set TextMIMEPart.Body=##class(%GlobalCharacterStream).%New()
    Do TextMIMEPart.Body.Write("/home/tabaiba/prueba.txt")

    // specify some headers
    Set TextMIMEPart.ContentType="text/plain"
    Set TextMIMEPart.ContentCharset="us-ascii"
    Do TextMIMEPart.SetHeader("Custom-header",header)

    // Insert both subparts into the root part
    Do RootMIMEPart.Parts.Insert(BinaryMIMEPart)
    Do RootMIMEPart.Parts.Insert(TextMIMEPart)

    // create MIME writer; write root MIME message
    Set writer=##class(%Net.MIMEWriter).%New()

    // Prepare outputting to the HttpRequestStream
    Set SentHttpRequest=##class(%Net.HttpRequest).%New()
    Set status=writer.OutputToStream(SentHttpRequest.EntityBody)
    if $$$ISERR(status) {do $SYSTEM.Status.DisplayError(status) Quit}

    // Now write down the content
    Set status=writer.WriteMIMEBody(RootMIMEPart)
    if $$$ISERR(status) {do $SYSTEM.Status.DisplayError(status) Quit}

    Set SentHttpRequest.Server="congrio"
    Set SentHttpRequest.Port = 8080

    Set ContentType= "multipart/form-data; boundary="_RootMIMEPart.Boundary
    Set SentHttpRequest.ContentType=ContentType

    set url="alfresco/service/sample/upload.json?"
            _"alf_ticket=TICKET_caee62bf36f0ea5bd51194fce161f99092b75f62"

    set status=SentHttpRequest.Post(url,0)
    if $$$ISERR(status) {do $SYSTEM.Status.DisplayError(status) Quit}
}
```

1.12 HTTP 応答へのアクセス

HTTP 要求を送信後、要求の HttpResponse プロパティが更新されます。このプロパティは、%Net.HttpResponse のインスタンスです。このオブジェクトの使用法の詳細は、"HTTP 応答の使用法" を参照してください。

1.13 関連項目

- HTTP 応答の使用法
- %Net.HttpRequest

2

HTTP 応答の使用法

ここでは、HTTP 応答オブジェクト (%Net.HttpResponse のインスタンス) の使用法について説明します。

2.1 応答のデータへのアクセス

HTTP 応答の本文は、応答の Data プロパティに含まれています。このプロパティには、ストリーム・オブジェクト、具体的には %GlobalBinaryStream が含まれています。%GlobalBinaryStream は非推奨になっていますが、このような使用法は引き続きサポートされています。このユース・ケースで別のストリーム・クラスに置き換えることはお勧めできません。

このストリームを使用するには、ストリーム・メソッド Write()、WriteLine()、Read()、ReadLine()、Rewind()、MoveToEnd()、および Clear() を使用します。また、ストリームの Size プロパティも使用できます。

要求 の ReadRawMode プロパティは、応答の本文を読み取る方法を制御します。

- 既定では、このプロパティは False であり、InterSystems IRIS では、本文が、応答の HTTP ヘッダで指定された文字セットである (かつ、それに応じて文字セットを変換している) と見なされます。
- このプロパティが True の場合、本文が RAW モードで読み取られます (文字セットの変換は行いません)。

また、OutputToDevice() メソッドを使用して、完全な応答を現在のデバイスに書き込むこともできます。ヘッダは、Web サーバによって生成された順序と同じ順序ではありません。

以下の簡単な例では、応答ストリームをファイルにコピーして保存します。

ObjectScript

```
set request=##class(%Net.HttpRequest).%New()
set request.Server="tools.ietf.org"
set request.Https=1
set request.SSLConfiguration="TEST"
set status=request.Get("/html/rfc7158")
if $$$ISERR(status) {
        do $system.OBJ.DisplayError()
} else {
        set response=request.HttpResponse
}

Set file=##class(%FileCharacterStream).%New()
set file.Filename="c:/temp/rfc7158.html"
set status=file.CopyFrom(response.Data)
if $$$ISERR(status) {
        do $system.OBJ.DisplayError()
}
do file.%Close()
```

2.2 名前による HTTP ヘッダの取得

%Net.HttpResponse クラスは、InterSystems IRIS 多次元配列内にその HTTP ヘッダを保存します。ヘッダにアクセスするには、以下のメソッドを使用します。

GetHeader()

指定したヘッダの値を返します。

GetNextHeader()

指定したヘッダの次のヘッダの名前を返します。

これらの各メソッドは、HTTP ヘッダの名前の文字列である単一の引数を取ります。

また、OutputHeaders() メソッドを使用して、HTTP ヘッダを現在のデバイスに書き込むこともできます (ただし、生成された順番と同じ順番ではありません)。

2.3 応答に関するその他の情報へのアクセス

%Net.HttpResponse クラスは、HTTP 応答のその他の特定の部分を保存するプロパティを提供します。

- StatusLine は、応答の第 1 行目の HTTP ステータス行を保存します。
- StatusCode は、HTTP ステータス・コードを保存します。
- ReasonPhrase は、StatusCode に相当する人間が読める形式の理由を保存します。
- ContentInfo は、応答本文についての追加情報を保存します。
- ContentType は、`Content-Type:` ヘッダの値を保存します。
- HttpVersion は、応答を送信した Web サーバがサポートする HTTP のバージョンを示します。

2.4 関連項目

- HTTP 要求の送信
- %Net.HttpRequest

3

InterSystems IRIS における電子メールのサポート

ここでは、InterSystems IRIS® データ・プラットフォームが MIME 電子メール・メッセージの送受信をどのようにサポートしているかについて説明します。

注釈　InterSystems IRIS は、メール・サーバを提供していません。その代わり、メール・サーバと接続し、やり取りする機能を提供します。

3.1 サポートされている電子メール・プロトコル

電子メールは、標準プロトコルを使用して、インターネット上でメッセージを送信します。InterSystems IRIS は、これらのプロトコルのうちの 3 つを以下のようにサポートします。

- InterSystems IRIS は、MIME 電子メール・メッセージのオブジェクト表現を提供します。テキストおよびテキスト以外の添付、シングルパートまたはマルチパートのメッセージ本文、および ASCII および非 ASCII 文字セットのヘッダをサポートします (MIME 部分のより一般的なサポートについては、"MIME メッセージの作成、記述および読み取り" を参照)。
- 電子メールを SMTP (Simple Mail Transport Protocol) サーバを介して送信できます。
- POP3 により電子メール・サーバから電子メールを受信することもできます。PSP3 は、電子メールをリモート・サーバから取得するための最も一般的な標準です。

3.2 InterSystems IRIS で MIME 電子メール・メッセージを表す方法

ここでは、InterSystems IRIS で MIME 電子メール・メッセージを表す方法について説明します。

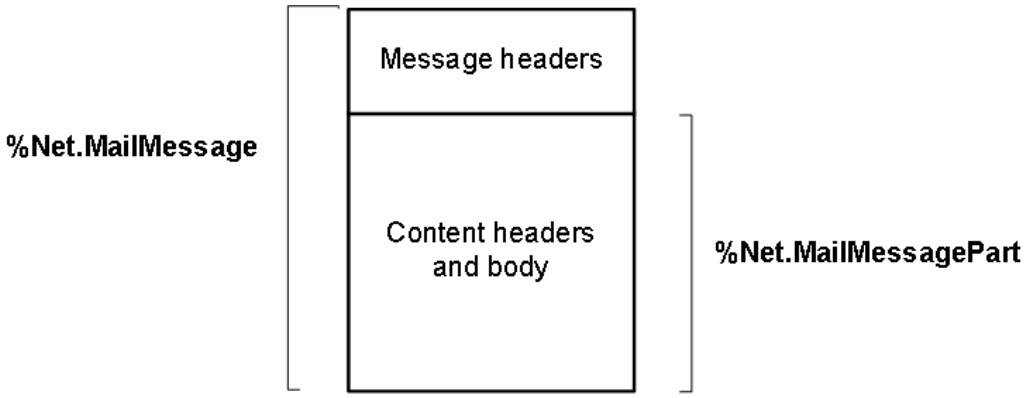
通常、マルチパート MIME メッセージは、以下の部分で構成されています。

- メッセージ・ヘッダ のセット。各メッセージ・ヘッダには、メッセージ送信先アドレスなどの情報が含まれています。これには、メッセージ全体の `Mime-Type` ヘッダおよび `Content-Type` ヘッダも含まれています。

マルチパート・メッセージの場合、Content-Type ヘッダは、multipart/mixed または multipart のその他のサブタイプである必要があります。MIME 標準には、多くの異形があります。

- それぞれが複数の部分からなる、複数のメッセージ部分。
  - Content-Type ヘッダおよびこの部分に特有なその他のヘッダが含まれるコンテンツ・ヘッダのセット。
  - 本文。テキストまたはバイナリのいずれかであり、他の部分の本文とは異なる文字セットを使用することができます。

InterSystems IRIS は、%Net.MailMessage および %Net.MailMessagePart (%Net.MailMessage のスーパークラス) の2 つのクラスを使用して、電子メール・メッセージを表します。以下のグラフィックに、これらのクラスの関係を示します。



一般に、

- 通常の 1 つの部分からなるメッセージを表すには、%Net.MailMessage を使用します。
- マルチパート・メッセージを表すには、親メッセージとして %Net.MailMessage を使用し、その部分として %Net.MailMessagePart の複数のインスタンスを使用します。

3.3 関連項目

- 電子メール・メッセージの作成
- SMTP を介した電子メールの送信
- POP3 を介した電子メールの取得
- 受信した電子メールの操作
- MIME メッセージの作成、記述および読み取り

4

電子メール・メッセージの作成

ここでは、InterSystems IRIS® データ・プラットフォームで電子メール・メッセージを作成する方法について説明します。

4.1 シングルパートの電子メール・メッセージの作成

シングルパート電子メール・メッセージを作成するには、%Net.MailMessage クラスを使用します。メール・メッセージを作成する手順は以下のとおりです。

1. %Net.MailMessage のインスタンスを作成します。

   Tipヒント　%New() への引数として文字セットを指定できます。このようにした場合、メッセージに対して Charset プロパティが設定されます。このメッセージへの影響については、"自動エンコードと文字変換" を参照してください。

2. インスタンスの To プロパティ、From プロパティ、および Subject プロパティを設定します。
   - To ― このメッセージの送信先の電子メール・アドレスのリスト。このプロパティは、標準の InterSystems IRIS リスト・クラスです。これを使用するには、標準のリスト・メソッドである Insert()、GetAt()、RemoveAt()、Count()、および Clear() を使用します。
   - From ― このメッセージの送信元の電子メール・アドレス。
   - Subject ― 使用している SMTP サーバで必要な場合、メッセージの件名。

3. オプションで、Date プロパティ、Cc プロパティ、Bcc プロパティ、およびその他のプロパティを設定します。詳細は、"基本的な電子メール・ヘッダの指定" を参照してください。
4. メッセージが平文でない場合、以下のプロパティを設定して、作成しているメッセージの種類を示します。
   - これが、HTML メッセージの場合、IsHTML プロパティを 1 に設定します。
   - これがバイナリ・メッセージの場合、IsBinary プロパティを 1 に設定します。

5. メッセージおよびそのヘッダの文字セットを指定するには、必要に応じて、Charset プロパティを設定します (このメッセージへの影響については、"自動エンコードと文字変換" を参照してください)。

   重要　メッセージ・コンテンツを追加する前に文字セットを指定することが重要です。

6. メッセージ・コンテンツを追加します。

- 平文または HTML の場合は、%FileCharacterStream のインスタンスである、TextData プロパティを使用します。このストリームの TranslateTable プロパティを指定する必要はありません。メール・メッセージの文字セットを指定したときに自動的に指定されています。
- バイナリ・データの場合は、%FileBinaryStream のインスタンスである、BinaryData プロパティを使用します。

%FileCharacterStream および %FileBinaryStream は非推奨になっていますが、このような使用法は引き続きサポートされています。このユース・ケースで別のストリーム・クラスに置き換えることはお勧めできません。

Tipヒント ストリームの Filename プロパティを指定する際は、必ず、ユーザが書き込みアクセス権を持つディレクトリを使用してください。

これらのプロパティを使用するには、標準ストリーム・メソッドである Write()、WriteLine()、Read()、ReadLine()、Rewind()、MoveToEnd()、および Clear() を使用します。ストリームの Size プロパティも使用もできます。これによりメッセージ・コンテンツのサイズが分かります。

注釈 使用している SMTP サーバの要件を理解しておく必要があります。例えば、SMTP サーバの中には、`Subject` ヘッダを含める必要があるものがあります。同様に、SMTP サーバの中には、任意の `From` ヘッダを許可しないものもあります。

同様に、SMTP サーバの中には、`Priority` ヘッダを認識するものもあれば、代わりに `X-Priority` を認識するものもあります。

"マルチパートの電子メール・メッセージの作成" を参照してください。

4.1.1 例 1 : CreateTextMessage()

以下のメソッドは、単純なメッセージを作成し、そのアドレスを指定します。

Class Member

```
ClassMethod CreateTextMessage() As %Net.MailMessage
{
    Set msg = ##class(%Net.MailMessage).%New()
    set msg.From = "test@test.com"
    Do msg.To.Insert("xxx@xxx.com")
    Do msg.Cc.Insert("yyy@yyy.com")
    Do msg.Bcc.Insert("zzz@zzz.com")
    Set msg.Subject="subject line here"
    Set msg.IsBinary=0
    Set msg.IsHTML=0
    Do msg.TextData.Write("This is the message.")

    Quit msg
}
```

4.1.2 例 2 : SimpleMessage()

実際にメッセージを送信するときにアドレスを指定したい場合もあります ("SMTP サーバを使用した電子メールの送信" にある "例 3 : SendMessage()" を参照)。前の例の以下のバリエーションでは、アドレスのないテキスト・メッセージが生成されます。

Class Member

```
ClassMethod SimpleMessage() As %Net.MailMessage
{
  Set msg = ##class(%Net.MailMessage).%New()
  Set msg.Subject="Simple message "_$h
  Set msg.IsBinary=0
  Set msg.IsHTML=0
  Do msg.TextData.Write("This is the message.")
  Quit msg
}
```

4.2 マルチパートの電子メール・メッセージの作成

マルチパートの電子メール・メッセージを作成する手順は以下のとおりです。

1. %Net.MailMessage のインスタンスを作成し、その To プロパティ、From プロパティ、および Subject プロパティを設定します。オプションで、その他のプロパティを設定し、その他のメッセージ・ヘッダを指定します。
2. IsMultiPart プロパティを 1 に設定します。
3. MultiPartType プロパティを "related"、"alternative"、"mixed" のいずれかに設定します。これは、メッセージ全体の Content-Type ヘッダに影響します。
4. メッセージに含まれる各部分について、%Net.MailMessagePart のインスタンスを作成し、"シングルパートの電子メール・メッセージの作成" の説明のとおり (手順 4 から開始)、そのプロパティを指定します。
5. 親電子メール・メッセージの場合は、Parts プロパティを設定します。これは配列です。各子メッセージ部分をこの配列に挿入します。

メッセージを送信するときに、%Net.SMTP クラスは、必要に応じて (MultiPartType プロパティの値がある場合) 自動的にメッセージに対して Content-Type ヘッダを設定します。

4.3 電子メール・メッセージ・ヘッダの指定

前述のように、メッセージ自体とメッセージの各部分の両方に、ヘッダのセットがあります。

%Net.MailMessage および %Net.MailMessagePart クラスは、最も一般的に使用されるヘッダに簡単にアクセスできるプロパティを提供しますが、必要な任意のヘッダを追加することもできます。このセクションでは、すべてのヘッダについて説明し、さらにカスタム・ヘッダを作成する方法についても説明します。

指定したメッセージの部分のヘッダは、その部分の Charset プロパティによって指定された文字セットを使用します。

注釈　使用している SMTP サーバの要件を理解しておく必要があります。例えば、SMTP サーバの中には、Subject ヘッダを含める必要があるものがあります。同様に、SMTP サーバの中には、任意の From ヘッダを許可しないものもあります。

同様に、SMTP サーバの中には、Priority ヘッダを認識するものもあれば、代わりに X-Priority を認識するものもあります。

4.3.1 基本的な電子メール・ヘッダの指定

以下のプロパティを設定して (%Net.MailMessage 内のみ)、メッセージ自体の最も一般的に使用されるヘッダを設定します。

- To ― (必須) このメッセージの送信先の電子メール・アドレスのリスト。このプロパティは、標準の InterSystems IRIS リストです。これを使用するには、標準のリスト・メソッドである Insert()、GetAt()、RemoveAt()、Count()、および Clear() を使用します。
- From ― (必須) このメッセージの送信元の電子メール・アドレス。
- Date ― このメッセージの日付。
- Subject ― (必須) このメッセージの件名を含む文字列。
- Sender ― メッセージの実際の送信者。
- Cc ― このメッセージの送信先のカーボン・コピー・アドレスのリスト。
- Bcc ― このメッセージの送信先のブラインド・カーボン・コピー・アドレスのリスト。

4.3.2 Content-Type ヘッダ

メッセージを送信するときに、メッセージおよび各メッセージ部分の `Content-Type` ヘッダが、以下のとおりに自動的に設定されます。

- メッセージが、平文 (IsHTML が 0 で、IsBinary が 0) の場合、`Content-Type` ヘッダは `"text/plain"` に設定されます。
- メッセージが HTML (IsHTML が 1 で、IsBinary が 0) の場合、`Content-Type` ヘッダは `"text/html"` に設定されます。
- メッセージがバイナリ (IsBinary が 1) の場合、`Content-Type` ヘッダは `"application/octet-stream"` に設定されます。
- メッセージがマルチパートの場合、`Content-Type` ヘッダは MultiPartType プロパティの値に応じて設定されます。

%Net.MailMessage と %Net.MailMessagePart にはどちらも ContentType プロパティが用意されており、`Content-Type` ヘッダにアクセスできます。

4.3.3 Content-Transfer-Encoding ヘッダ

%Net.MailMessage と %Net.MailMessagePart にはどちらも、ContentTransferEncoding プロパティが用意されており、メッセージまたはメッセージの部分の `Content-Transfer-Encoding` ヘッダを簡単な方法で指定できます。

このプロパティは、`"base64"`、`"quoted-printable"`、`"7bit"`、`"8bit"` のいずれかです。

既定値は、以下のとおりです。

- バイナリ・メッセージまたはメッセージ部分の場合、`"base64"` です。

重要　コンテンツを `"base64"` でエンコードする場合、Unicode 文字を含めることはできません。送信するコンテンツに Unicode 文字が含まれている場合は必ず、$ZCONVERT を使用してコンテンツを UTF-8 に変換してから、base-64 でエンコードしてください。以下に例を示します。

```
set BinaryText=$ZCONVERT(UnicodeText,"O","UTF8")
set Base64Encoded=$system.Encryption.Base64Encode(BinaryText)
```

受信者は、逆のプロセスを使用してテキストをデコードする必要があります。

```
set BinaryText=$system.Encryption.Base64Decode(Base64Encoded)
set UnicodeText=$ZCONVERT(BinaryText,"I","UTF8")
```

- テキスト・メッセージまたはメッセージ部分の場合、"quoted-printable" です。

また、"自動エンコードと文字変換" も参照してください。

4.3.4 カスタム・ヘッダ

%Net.MailMessage と %Net.MailMessagePart の両方で、以下の構造を持つ配列である Headers プロパティにアクセスして、カスタム・ヘッダを設定または取得できます。

配列キー	配列値
"Priority" などのヘッダ名	ヘッダの値

このプロパティを使用して、X-Priority など、追加のヘッダを含めます。以下はその例です。

ObjectScript

```
do msg.Headers.SetAt(1,"X-Priority")
do msg.Headers.SetAt("High","X-MSMail-Priority")
do msg.Headers.SetAt("High","Importance")
```

各種の電子メール・クライアントおよびサーバでは認識するヘッダも異なるので、複数の類似ヘッダを設定して、確実に認識できるヘッダの付いたメッセージを受信したほうが便利となります。

4.4 メッセージへの添付の追加

添付を電子メール・メッセージまたはメッセージの部分 (具体的には、%Net.MailMessagePart または %Net.MailMessage のインスタンス) に追加できます。この操作には、以下のメソッドを使用します。

これらのメソッドは、それぞれ、添付を元のメッセージ (またはメッセージの部分) の Parts 配列に追加し、自動的に IsMultiPart プロパティを 1 に設定します。

AttachFile()

```
method AttachFile(Dir As %String,
                  File As %String,
                  isBinary As %Boolean = 1,
                  charset As %String = "",
                  ByRef count As %Integer) as %Status
```

指定したファイルを電子メール・メッセージに添付します。既定では、ファイルはバイナリの添付として送信されますが、代わりにテキストであると指定することができます。テキストの場合、そのファイルが使用する文字セットを指定することもできます。

具体的には、このメソッドは、%Net.MailMessagePart のインスタンスを作成し、適宜、ファイルのコンテンツを BinaryData プロパティまたは TextData プロパティに配置し、必要に応じて、Charset プロパティおよび TextData.TranslateTable プロパティを設定します。メソッドは、Parts 配列内のこの新規メッセージ部分の位置を示す整数を参照によって返します。

このメソッドは、メッセージまたはメッセージの部分の Dir プロパティおよび FileName プロパティも設定します。

AttachStream()

```
method AttachStream(stream As %Stream.Object,
                    Filename As %String,
                    isBinary As %Boolean = 1,
                    charset As %String = "",
                    ByRef count As %Integer) as %Status
```

指定したストリームを電子メール・メッセージに添付します。Filename が指定されている場合、この添付はファイル添付と見なされます。指定されていない場合、インライン添付と見なされます。AttachFile() の説明を参照してください。

AttachNewMessage()

```
method AttachNewMessage() as %Net.MailMessagePart
```

%Net.MailMessage の新規インスタンスを作成し、それをメッセージに追加して、新たに変更された親メッセージまたはメッセージの部分を返します。

AttachEmail()

```
method AttachEmail(mailmsg As %Net.MailMessage)
```

電子メール・メッセージ (%Net.MailMessage のインスタンス) がある場合、このメソッドは、それをメッセージに追加します。このメソッドは、メッセージまたはメッセージの部分の Dir プロパティおよび FileName プロパティも設定します。

注釈　このメソッドは、ContentType を "message/rfc822" に設定します。この場合、その他の添付は追加できません。

4.4.1 例 : MessageWithAttachment()

以下の例は、ハードコードされた添付が 1 つある簡単な電子メール・メッセージを生成します。メッセージのアドレスは提供しません。実際にメッセージを送信するときにこの情報を設定できます ("SMTP サーバを使用して電子メールを送信" の "例 3 : SendMessage()" を参照)。

Class Member

```
ClassMethod MessageWithAttachment() As %Net.MailMessage
{
  Set msg = ##class(%Net.MailMessage).%New()
  Set msg.Subject="Message with attachment "_$h
  Set msg.IsBinary=0
  Set msg.IsHTML=0
  Do msg.TextData.Write("This is the main message body.")

  //add an attachment
  Set status=msg.AttachFile("c:\", "GNET.pdf")
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Quit $$$NULLOREF
  }

  Quit msg
}
```

その他の例については、%Net.MailMessagePart クラスのクラス・リファレンスを参照してください。

4.5 エンコードと文字変換

電子メールのメッセージ部分には、使用されている文字セットおよび使用されている Content-Transfer-Encoding（存在する場合）の両方についての情報が含まれています。参照のため、このセクションでは、この情報の使用方法について説明します。

文字セットおよび変換テーブルの詳細は、"変換テーブル" を参照してください。

%Net.SMTP は、各部分の Charset プロパティを確認し、適切な変換テーブルを適用します。

指定した部分に対して Charset プロパティを指定しない場合、InterSystems IRIS では UTF-8 が使用されます。

%Net.SMTP も ContentTransferEncoding プロパティを確認します。このプロパティが `"base64"` または `"quoted-printable"` の場合、メッセージを作成するときに、%Net.SMTP が、必要に応じて本文をエンコードします。(コンテンツ転送エンコードが `"7bit"` または `"7bit"` の場合、エンコードは必要ありません。)

重要　コンテンツを `"base64"` でエンコードする場合、Unicode 文字を含めることはできません。送信するコンテンツに Unicode 文字が含まれている場合は必ず、$ZCONVERT を使用してコンテンツを UTF-8 に変換してください。

4.6 関連項目

- InterSystems IRIS における電子メールのサポート
- SMTP を介した電子メールの送信
- MIME メッセージの作成、記述および読み取り

5

SMTP を介した電子メールの送信

ここでは、%Net.SMTP を使用して MIME 電子メール・メッセージを送信する方法について説明します。

5.1 概要

SMTP サーバにアクセスできる場合、電子メール・メッセージを送信できます。SMTP サーバが稼動しており、それを使用するために必要な権限を持っている必要があります。電子メールを送信するには、以下の手順を実行します。

1. %Net.SMTP のインスタンスを作成し、必要に応じてそのプロパティを設定します。特に、以下のプロパティを設定します。
   - smtpserver は、使用している SMTP サーバの名前です。
   - port は、SMTP サーバで使用しているポートです。既定値は、25 です。
   - timezone では、RFC 822 によって指定されたとおりに、サーバのタイム・ゾーンを指定します。例えば、"EST"、"-0400"、"LOCAL" などです。これを設定しない場合、メッセージは協定世界時を使用します。

   このオブジェクトは、使用する SMTP サーバを記述します。
2. SMTP サーバが認証を必要とする場合、必要な資格情報を指定します。そのためには、以下の操作を実行します。
   a. %Net.Authenticator のインスタンスを作成します。
   b. このオブジェクトの UserName プロパティおよび Password プロパティを設定します。
   c. %Net.SMTP インスタンスの authenticator プロパティをこのオブジェクトと等しくなるように設定します。
   d. メッセージ自体に許可されたセンダがある場合、%Net.SMTP インスタンスの AuthFrom プロパティを設定します。
   e. MechanismList を設定します。ここでは、試行する認証方法を指定します。メール・サーバには使用できる認証メカニズムのリストが保持されていて、MechanismList にある最初のメカニズムがメール・サーバとの接続に使用されます。アプリケーションの要件とサーバの設定に適した MechanismList を定義します。
3. SMTP サーバへの SSL/TLS 接続を使用するには
   a. SSLConfiguration プロパティを使用する有効化した SSL/TLS 構成の名前に設定します。

      SSL/TLS 構成の作成と管理の詳細は、インターシステムズの "TLS ガイド" を参照してください。SSL/TLS 構成には、[構成名] と呼ばれるオプションが含まれています。これは、この設定で使用する文字列です。
   b. UseSTARTTLS プロパティを 0 か 1 に設定します。

ほとんどの場合、0 の値を使用します。通常の TCP ソケットでサーバのやり取りが開始した後、通常のソケットとして同じポートの TLS に切り替わる場合、1 の値を使用します。詳細は、"RFC 3207" を参照してください。

c. 必要に応じて、SSLCheckServerIdentity プロパティを 1 に設定します。これは、証明書のホスト・サーバ名を確認する場合に設定します。

4. 送信する電子メール・メッセージを作成します（手順については、"シングルパートの電子メール・メッセージの作成" および "マルチパートの電子メール・メッセージの作成" を参照）。
5. SMTP インスタンスの Send() メソッドを呼び出します。このメソッドはステータスを返すので、それを確認する必要があります。
6. 返されたステータスがエラーを示している場合、エラー・メッセージを含んでいる Error プロパティを確認します。
7. 送信アクションに失敗した電子メールのアドレスのリストを含んでいる FailedSend プロパティを確認します。

以下のセクションの例は、このドキュメント執筆時点で使用可能な 2 つの異なる無料 SMTP サービスを使用しています。これらのサービスの選択は、特に推奨を意味しているわけではありません。また、例で示しているのは、実際のパスワードではありません。

重要

%Net.SMTP は、メッセージ本文を一時ファイル・ストリームに書き込みます。既定では、このファイルはネームスペースのディレクトリに書き込まれますが、ディレクトリで特別な書き込み許可が必要な場合、ファイルは作成されずに空のメッセージ本文を取得することになります。

これらの一時ファイルに新規のパスを定義して、書き込みアクセスを制限しないパスを選択することができます（/tmp など）。そのためには、グローバル・ノード %SYS("StreamLocation",namespace) を設定します。ここで、namespace はお使いのコードが実行されているネームスペースです。以下はその例です。

```
Set ^%SYS("StreamLocation","SAMPLES")="/tmp"
```

%SYS("StreamLocation",namespace) が NULL の場合は、InterSystems IRIS では %SYS("TempDir",namespace) で指定されたディレクトリが使用されます。%SYS("TempDir",namespace) が設定されていない場合は、InterSystems IRIS では %SYS("TempDir") で指定されたディレクトリが使用されます。

5.2 XOAUTH2 認証の有効化

%Net.SMTP では、認証方法として XOAUTH2 をサポートしています。

%Net.Authenticator にはアクセス・トークン・プロパティがあるので、それを XOAUTH2 によってメカニズムのリストに追加できます。XOAUTH2 を追加すると、それが %Net.Authenticator で認証方法として使用されます。追加しない場合は使用されません。%Net.SMTP で XOAUTH2 の使用を望まない場合は、何もする必要はありません。

5.3 例 1 : HotPOPAsSMTP() および SendSimpleMessage()

この例は、一緒に使用する 2 つのメソッドで構成されます。最初のメソッドは、HotPOP SMTP サーバで既に設定されているテスト・アカウントを使用する %Net.SMTP のインスタンスを作成します。

Class Member

```
ClassMethod HotPOPAsSMTP() As %Net.SMTP
{
  Set server=##class(%Net.SMTP).%New()
  Set server.smtpserver="smtp.hotpop.com"
  //HotPOP SMTP server uses the default port (25)
  Set server.port=25

  //Create object to carry authentication
  Set auth=##class(%Net.Authenticator).%New()
  Set auth.UserName="isctest@hotpop.com"
  Set auth.Password="123pass"

  Set server.authenticator=auth
  Set server.AuthFrom=auth.UserName
  Quit server
}
```

次のメソッドは、引数として指定する SMTP サーバを使用して、簡単な一意のメッセージを送信します。

Class Member

```
ClassMethod SendSimpleMessage(server As %Net.SMTP) As %List
{
  Set msg = ##class(%Net.MailMessage).%New()
  Set From=server.authenticator.UserName
  Set:From="" From="xxx@xxx.com"
  Set msg.From = From

  Do msg.To.Insert("xxx@xxx.com")
  //Do msg.Cc.Insert("yyy@yyy.com")
  //Do msg.Bcc.Insert("zzz@zzz.com")
  Set msg.Subject="Unique subject line here "_$H
  Set msg.IsBinary=0
  Set msg.IsHTML=0
  Do msg.TextData.Write("This is the message.")

  Set status=server.Send(msg)
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Write server.Error
    Quit ""
  }
  Quit server.FailedSend
}
```

5.4 例 2 : YPOPsAsSMTP()

この例は、Yahoo 電子メール・アカウントへの SMTP および POP3 アクセスを提供するクライアント・ソフトウェアである YPOP を使用する %Net.SMTP のインスタンスを作成します。これは、この目的で既に設定されているテスト・アカウントを使用します。

Class Member

```
ClassMethod YPOPsAsSMTP() As %Net.SMTP
{
  Set server=##class(%Net.SMTP).%New()
  //local host acts as the server
  Set server.smtpserver="127.0.0.1"
  //YPOPs uses default port, apparently
  Set server.port=25

  //Create object to carry authentication
  Set auth=##class(%Net.Authenticator).%New()
  //YPOPs works with a Yahoo email account
  Set auth.UserName="isc.test@yahoo.com"
  Set auth.Password="123pass"

  Set server.authenticator=auth
  Set server.AuthFrom=auth.UserName
  Quit server
}
```

前の例で示した SendSimpleMessage メソッドでこれを使用できます。

5.5 例 3 : SendMessage()

以下の、より柔軟なメソッドは、SMTP サーバと電子メール・メッセージの両方を受け入れます。電子メール・メッセージには、既に件名の行が含まれている必要があります (SMTP サーバで必要な場合) が、アドレスを含める必要はありません。このメソッドは、その後、電子メール・メッセージをハードコードされたテストの宛先一式に送信します。

Class Member

```
ClassMethod SendMessage(server As %Net.SMTP, msg as %Net.MailMessage) as %Status
{
  Set From=server.authenticator.UserName
  //make sure From: user is same as used in authentication
  Set msg.From = From

  //finish addressing the message
  Do msg.To.Insert("xxx@xxx.com")
  //send the message to various test email addresses
  Do msg.To.Insert("isctest@hotpop.com")
  Do msg.To.Insert("isc_test@hotmail.com")
  Do msg.To.Insert("isctest001@gmail.com")
  Do msg.To.Insert("isc.test@yahoo.com")

  Set status=server.Send(msg)
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Write server.Error
    Quit $$$ERROR($$$GeneralError,"Failed to send message")
  }
  Quit $$$OK
}
```

この例は、"メッセージへの添付の追加" で説明されている SimpleMessage メソッド例および MessageWithAttachment メソッド例と共に使用するためのものです。

5.6 %Net.SMTP のその他のプロパティ

%Net.SMTP クラスには、使用している SMTP サーバによっては必要な場合がある以下のプロパティもあります。

- AllowHeaderEncoding は、Send() メソッドが非 ASCII ヘッダ・テキストをエンコードするかどうかを指定します。既定値は 1 で、これは、非 ASCII ヘッダ・テキストが RFC 2047 によって指定されたとおりにエンコードされることを意味します。
- ContinueAfterBadSend は、送信に失敗した電子メール・アドレスを検出した後、メッセージの送信を継続して試行するかどうかを指定します。ContinueAfterBadSend が 1 の場合、システムは、送信に失敗した電子メール・アドレスを FailedSend プロパティのリストに追加します。既定値は 0 です。
- ShowBcc は、Bcc ヘッダを電子メール・メッセージに記述するかどうかを指定します。これらは通常、SMTP サーバによって、フィルタで除外されます。

5.7 関連項目

- InterSystems IRIS における電子メールのサポート

- 電子メール・メッセージの作成

6

POP3 を介した電子メールの取得

ここでは、%Net.POP3 クラスを使用して、電子メール・メッセージを取得する方法について説明します。

6.1 概要

必要な権限があり、メール・サーバが稼動している場合、POP3 プロトコルを使用して、メール・サーバから電子メール・メッセージをダウンロードし、処理することができます。一般に、POP3 を介してメール・サーバと通信するには、ログインし、メールボックスに影響を与える一連のアクションを実行し、変更をコミットまたはロールバックします。InterSystems IRIS 内でこれを行うには、以下を実行します。

1. %Net.POP3 のインスタンスを作成します。このオブジェクトは、使用するメール・サーバを記述します。
2. オプションで、%Net.POP3 のインスタンスの以下のプロパティを指定します。
   - port ― 使用するポートを指定します。既定値は 110 です。
   - timeout ― 読み取りタイムアウトを秒単位で指定します。既定値は 30 秒です。
   - StoreAttachToFile ― メッセージを読み取る際に、各添付をファイルに保存するかどうかを指定します (メッセージが `content-disposition; attachment` ヘッダを含んでいる場合)。既定値は False です。

     この設定は、AttachDir も設定しないと、何も実行しないことに注意してください。
   - StoreInlineToFile ― メッセージを読み取る際に、各インライン添付をファイルに保存するかどうかを指定します (メッセージが `content-disposition; inline` ヘッダを含んでいる場合)。既定値は False です。

     この設定は、AttachDir も設定しないと、何も実行しないことに注意してください。
   - AttachDir ― 添付を保存するディレクトリを指定します。既定値はありません。ディレクトリの名前は、オペレーティング・システムに応じて適切に、スラッシュ (/) またはバックスラッシュ (\) で終わるようにします。また、これが既に存在するディレクトリで、ユーザがこれに対するアクセス権を持っていることを確認してください。
   - IgnoreInvalidBase64Chars ― Base-64 デコード中に検出された無効な文字を無視するかどうかを指定します。既定値は false です (無効な文字ではエラーが発生します)。RFC 2045 では、Base-64 デコード中に予期しない文字を無視するか、エラーを発生させるかは曖昧です。
3. サーバへの SSL/TLS 接続を使用するには
   a. SSLConfiguration プロパティを使用する有効化した SSL/TLS 構成の名前に設定します。

      SSL/TLS 構成の作成と管理の詳細は、インターシステムズの "TLS ガイド" を参照してください。SSL/TLS 構成には、[構成名] と呼ばれるオプションが含まれています。これは、この設定で使用する文字列です。

b. UseSTARTTLS プロパティを 0 か 1 に設定します。

ほとんどの場合、0 の値を使用します。通常の TCP ソケットでサーバのやり取りが開始した後、通常のソケットとして同じポートの TLS に切り替わる場合、1 の値を使用します。詳細は、"RFC 2595" を参照してください。

UseSTATRTLS を 1 に設定している場合は、既定のポート 995 ではなく、ポート 110 にサーバを接続します。

c. 必要に応じて、SSLCheckServerIdentity プロパティを 1 に設定します。これは、証明書のホスト・サーバ名を確認する場合に設定します。

4. インスタンスの Connect() メソッドを呼び出します。このメソッドは、3 つの引数を以下の順番で取ります。

a. POP3 サーバの名前

b. ユーザ名

c. パスワード

5. インスタンスのメソッドを使用して、メールボックスを調べ、メッセージを取得し、メッセージを削除します。以下のセクションで詳細を説明します。

6. オプションで、接続がタイムアウトするのを防ぐため、%Net.POP3 インスタンスの Ping() メソッドを呼び出します。

7. オプションで、削除するようメッセージにマーク付けしたものの、やはり削除しないことにした場合は、%Net.POP3 インスタンスの RollbackDeletes() メソッドを呼び出します。

8. メールボックスへの変更が完了したら、以下のメソッドのいずれかを呼び出します。

- QuitAndCommit() ー 変更をコミットし、メール・サーバからログアウトします。
- QuitAndRollback() ー 変更をロールバックし、メール・サーバからログアウトします。

これらの各メソッドはステータスを返すので、続行する前にそれを確認する必要があります。完全なメソッドのシグニチャについては、%Net.POP3 のクラス・リファレンスも参照してください。

6.2 XOAUTH2 認証の有効化

%Net.POP3 では、認証方法として XOAUTH2 をサポートしています。

%Net.POP3 で XOAUTH2 を使用するには、アクセス・トークンをパラメータとして %Net.POP3.Connect() に渡します。このメソッドは、アクセス・トークンを渡されると、パスワードが提供されたかどうかに関係なく、XOAUTH2 を使用しようとします。XOAUTH2 の使用を望まない場合は、アクセス・トークン・パラメータを渡さないように注意する必要があります。

6.3 接続例

以下の例は、このコンテンツ執筆時点で使用可能な 2 つの異なる無料 POP3 サービスを使用しています。これらのサービスの選択は、特に推奨を意味しているわけではありません。また、例で示しているのは、実際のパスワードではありません。

6.3.1 例 1 : HotPOPAsPOP3()

以下のメソッドがこの目的で既に設定されているアカウントを使用して、HotPOP POP3 サーバにログインします。

Class Member

```
ClassMethod HotPOPAsPOP3() As %Net.POP3
{
  Set server=##class(%Net.POP3).%New()

  //HotPOP POP3 server uses the default port
  //but let's set it anyway
  Set server.port=110

  //just in case we plan to fetch any messages
  //that have attachments
  Set server.StoreAttachToFile=1
  Set server.StoreInlineToFile=1
  Set server.AttachDir="c:\DOWNLOADS\"

  Set servername="pop.hotpop.com"
  Set user="isctest@hotpop.com"
  Set pass="123pass"

  Set status=server.Connect(servername,user,pass)
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Quit $$$NULLOREF
  }
  Quit server
}
```

このメソッドは、%Net.POP3 サーバ・インスタンスを返します。このトピックで後述する例の多くは、%Net.POP3 インスタンスを引数として受け入れます。

6.3.2 例 2 : YPOPsAsPOP3()

以下のメソッドも、%Net.POP3 サーバ・インスタンスを返します。この場合、Yahoo 電子メール・アカウントへの SMTP および POP3 アクセスを提供するクライアント・ソフトウェアである YPOP を使用しています。これは、この目的で既に設定されているテスト・アカウントを使用します。

Class Member

```
ClassMethod YPOPsAsPOP3() As %Net.POP3
{
  Set server=##class(%Net.POP3).%New()

  //YPOPs uses the default port
  //but let's set it anyway
  Set server.port=110

  //just in case we plan to fetch any messages
  //that have attachments
  Set server.StoreAttachToFile=1
  Set server.StoreInlineToFile=1
  Set server.AttachDir="c:\DOWNLOADS\"

  //local host acts as the server
  Set servername="127.0.0.1"
  //YPOPs works with a Yahoo email account
  Set user="isc.test@yahoo.com"
  Set pass="123pass"

  Set status=server.Connect(servername,user,pass)
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Quit $$$NULLOREF
  }
  Quit server
}
```

6.4 メールボックスに関する情報の取得

POP3 サーバに接続している間、ユーザ・アカウントにログインし、そのユーザ・アカウントのメールボックスにアクセスできます。以下のメソッドを使用して、メールボックスに何が含まれているのかを検索します。

GetMailBoxStatus()

メールボックス内のメッセージ数およびメールボックスが使用するバイト数を、参照によって返します。

GetMessageUIDArray()

最初の引数として空の文字列が指定されている場合、このメソッドは、メールボックス内のメッセージについての情報の配列を、参照によって返します (現在削除するようマーク付けされているものを除く)。この配列の各要素には、1 つのメッセージについての以下の情報が含まれています。

配列キー	配列項目
現在の状態でのメールボックス内のメッセージの番号。最初のメッセージが 1 番で、以降順次番号が割り当てられます。 指定したメッセージのメッセージ番号がすべてのセッションで同じであることは保証されません。	すべてのセッションで使用できる、このメッセージの永久の識別子である、一意のメッセージ識別子 (UID)。 UID は、各メールボックスに対して一意です。

GetSizeOfMessages()

最初の引数として空の文字列が指定されている場合、このメソッドは、メールボックス内のメッセージについての情報の配列を、参照によって返します (現在削除するようマーク付けされているものを除く)。この配列の各要素には、1 つのメッセージについての以下の情報が含まれています。

配列キー	配列項目
現在の状態でのメールボックス内のメッセージの番号。	このメッセージのサイズ (単位はバイト)。

これらの各メソッドはステータスを返すので、続行する前にそれを確認する必要があります。これらのメソッドの詳細は、"その他のメッセージ取得メソッド" も参照してください。

6.4.1 例 : ShowMailbox()

例えば、以下のメソッドは、現在アクセスしているメールボックスについての情報を記述します。

Class Member

```
ClassMethod ShowMailbox(server as %Net.POP3)
{
    Set status=server.GetMailBoxStatus(.count,.size)
    If $$$ISERR(status) {
       Do $System.Status.DisplayError(status)
       Quit
    }
    Write "Mailbox information *****",!
    Write "Number of messages in mailbox: ",count,!
    Write "Size of messages: ",size,!

    Set status=server.GetMessageUIDArray(,.uids)
    Set status=server.GetSizeOfMessages(,.sizes)

    //iterate through messages, get info, and write it
    For i=1:1:count {
```

```
        Set uid=uids.GetAt(i)
        Set size=sizes.GetAt(i)
        Write "Msg number:", i,"   UID:",uid, "   size:",size,!
    }

}
```

このメソッドは、以下のような出力を生成します。

```
Mailbox information *****
Number of messages in mailbox: 4
Size of messages: 18634
Msg number:1   UID:6ef78df6fd660391   size:7245
Msg number:2   UID:7410041a6faf4a87   size:5409
Msg number:3   UID:5555af7fa489e406   size:5121
Msg number:4   UID:299ad2b54c01a6be   size:859
```

6.5 メッセージの取得

単にメッセージを取得するには、%Net.POP3 クラスの以下のメソッドのいずれかを使用します。

Fetch()

最初の引数としてメッセージ番号が指定されている場合、このメソッドは (2 番目の引数として、参照によって) そのメッセージを含む %Net.MailMessage のインスタンスを返します。

FetchMessage()

最初の引数としてメッセージ番号が指定されている場合、このメソッドは、From や To およびその他の共通ヘッダなどの情報、すべてのヘッダ (共通ヘッダを含む) を含む配列、およびメッセージ・コンテンツ自体を (参照によって) 返します。

これらの各メソッドはステータスを返すので、続行する前にそれを確認する必要があります。メッセージが現在削除するようマーク付けされている場合、これらのメソッドは、エラー・ステータスを返します。

Fetch() および FetchMessage() のメソッド・シグニチャー一覧については、"その他のメッセージ取得メソッド" も参照してください。

6.5.1 例 : FetchMailbox()

以下の例は、"メールボックスに関する情報の取得" で説明されている ShowMailbox の例のバリエーションです。このメソッドは、Fetch() メソッドを使用し、各メッセージを調べ、各メッセージの件名行を記述します。

Class Member

```
ClassMethod FetchMailbox(server As %Net.POP3)
{
  Set status=server.GetMailBoxStatus(.count,.size)
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Quit $$$NULLOREF
  }
  Write "Mailbox information *****",!
  Write "Number of messages in mailbox: ",count,!
  Write "Size of messages: ",size,!

  Set status=server.GetMessageUIDArray(,.uids)
  Set status=server.GetSizeOfMessages(,.sizes)

  //iterate through messages, get info, and write it
  For i=1:1:count {
    Set uid=uids.GetAt(i)
    Set size=sizes.GetAt(i)
    Set status=server.Fetch(i,.msg)
    If $$$ISERR(status) {
```

```
      Set subj="***error***"
    } else{
      Set subj=msg.Subject
    }
    Write "Msg number:", i,"  UID:",uid, "  Size:",size
    Write "  Subject: ",subj,!
  }
}
```

6.6 エンコードと文字変換

電子メールのメッセージ部分には、使用されている文字セットおよび使用されている Content-Transfer-Encoding (存在する場合) の両方についての情報が含まれています。参照のため、このセクションでは、この情報の使用方法について説明します。

文字セットおよび変換テーブルの詳細は、"変換テーブル" を参照してください。

%Net.POP3 は、各メッセージ部分の `Content-Transfer-Encoding` ヘッダを確認し、必要に応じて本文をデコードします。

その後、%Net.POP3 は、各メッセージ部分の `Content-Type` ヘッダを確認します。これは、メッセージ部分の Charset プロパティに影響を与え、また、メッセージ部分が InterSystems IRIS 内で作成されるときに使用される変換テーブルの制御も行います。

6.7 添付の保存

`Content-Disposition` ヘッダにより、ファイル名を付加、あるいは付加せずに `attachment` を指定する場合があります。以下はその例です。

```
Content-Disposition: attachment; filename=genome.jpeg;
```

`Content-Disposition` ヘッダで `attachment` を指定した場合、ユーザの %Net.POP3 インスタンスでメッセージ内のすべての添付をファイルに保存できます。この機能を有効にするには、以下を行います。

1. %Net.POP3 のインスタンスの以下のプロパティを指定します。
   - StoreAttachToFile を 1 に設定します。
   - StoreInlineToFile を 1 に設定します。
   - AttachDir に有効なディレクトリを指定します。ディレクトリの名前は、オペレーティング・システムに応じて適切に、スラッシュ (/) またはバックスラッシュ (\) で終わるようにします。また、これが既に存在するディレクトリで、ユーザがこれに対するアクセス権を持っていることを確認してください。
2. ユーザの %Net.POP3 インスタンスの Fetch() または FetchMessage() を呼び出します。

各ファイル名は、以下のとおり決定されます。

1. `Content-Disposition` ヘッダでファイル名が指定されている場合、そのファイル名が使用されます。
2. それ以外の場合、`Content-Type` ヘッダでファイル名が指定されていれば、そのファイル名が使用されます。
3. それ以外の場合、システムは、ATTxxxxxx.dat の形式の名前を作成します。

以下の点に注意してください。

- ファイルが既に存在する場合、添付はダウンロードされません。
- AttachDir には、既定値はありません。
- 添付のサイズは、InterSystems IRIS によっては制限されませんが、ファイル・システムによって制限される場合があります。
- ここでは、Dir および FileName プロパティは使用されていません。これらのプロパティは、"メッセージへの添付の追加" で説明したとおり、添付をメール・メッセージにアップロードする場合にのみ関係します。

6.7.1 例 : GetMsg()

以下のメソッド例は、%Net.POP3 のインスタンスおよびメッセージ番号を指定すると、メッセージ全体を取得します。

Class Member

```
ClassMethod GetMsg(server as %Net.POP3,msgno as %Integer) as %Net.MailMessage
{
    Set status=server.Fetch(msgno,.msg)
    If $$$ISERR(status) {
       Do $System.Status.DisplayError(status)
       Quit $$$NULLOREF
    }
    Quit msg
}
```

メッセージに添付があり、%Net.POP3 サーバの StoreAttachToFile、StoreInlineToFile、および AttachDir の各プロパティを指定した場合、それらの添付は、このメソッドを呼び出したときに指定のディレクトリに保存されます。

6.8 添付されているメッセージの取得

メールボックスに接続している間は、受信トレイ内の電子メール・メッセージに添付されている電子メール・メッセージをダウンロードできます。そのためには、%Net.POP3 インスタンスの GetAttachedEmail() メソッドを使用して、添付された電子メールのコンテンツを取得します。

%Net.MailMessagePart のインスタンスを指定した場合、このメソッドは、そのメッセージ部分のコンテンツを持つシングルパートのメッセージを返します。具体的には、添付された電子メール・メッセージから取られたデータで初期化された %Net.MailMessage のインスタンスを (出力パラメータとして) 返します。

6.9 調べて取得するメソッドの概要

このセクションでは、メッセージを調べ、取得する際に使用できる %Net.POP3 のメソッドをすべてリストします。

Fetch()

```
method Fetch(MessageNumber As %Integer,
             ByRef MailMsg As %Net.MailMessage,
             Delete As %Boolean = 0,
             messageStream As %BinaryStream) as %Status
```

MessageNumber によって指定されたメッセージを (参照によって) 返し、オプションでそのメッセージに削除するようマーク付けします。メッセージが既に削除するようマーク付けされている場合、このメソッドは、エラー・ステータスを返します。

messageStream が指定された場合、元のメッセージはこのバイナリ・ストリームに書き込まれます。

FetchFromStream()

```
method FetchFromStream(messageStream As %BinaryStream, ByRef Msg As %Net.MailMessage) as %Status
```

このメソッドは、Fetch() に messageStream 引数を指定した場合に使用します。

単一の電子メール・メッセージを指定されたバイナリ・ストリームから取得します。messageStream は、メッセージを含んだバイナリ・ストリームである必要があります。このメッセージは Msg の参照によって返されます。これはマルチパート・メッセージとなる可能性があります。

FetchMessage()

```
method FetchMessage(MessageNumber As %Integer,
                    ByRef From As %String,
                    ByRef To As %String,
                    ByRef Date As %String,
                    ByRef Subject As %String,
                    ByRef MessageSize As %Integer,
                    ByRef MsgHeaders As %ArrayOfDataTypes,
                    ByRef MailMsg As %Net.MailMessage,
                    Delete As %Boolean = 0) as %Status
```

特定のメッセージ・ヘッダ、メッセージ・サイズ、メッセージ・ヘッダ配列、およびメッセージ自体を（参照によって）返し、オプションでメッセージに削除するようマーク付けします。メッセージが既に削除するようマーク付けされている場合、このメソッドは、エラー・ステータスを返します。

FetchMessageHeaders()

```
method FetchMessageHeaders(MessageNumber As %Integer,
                           ByRef MsgHeadersArray As %String) as %Status
```

メッセージ番号を指定すると、このメソッドは、そのメッセージのすべてのヘッダを含む配列を（参照によって）返します。メッセージが現在削除するようマーク付けされている場合、このメソッドは、エラー・ステータスを返します。

FetchMessageInfo()

```
method FetchMessageInfo(MessageNumber As %Integer,
                        Lines As %Integer,
                        ByRef From As %String,
                        ByRef To As %String,
                        ByRef Date As %String,
                        ByRef Subject As %String,
                        ByRef MessageSize As %Integer,
                        ByRef MsgHeaders As %ArrayOfDataTypes,
                        ByRef MessageText As %String) as %Status
```

メッセージ番号を指定すると、このメソッドは、特定のメッセージ・ヘッダ、メッセージ・サイズ、メッセージ・ヘッダの配列、およびこのメッセージのテキストの指定の行数を（参照によって）返します。メッセージが現在削除するようマーク付けされている場合、このメソッドは、エラー・ステータスを返します。

GetAttachedEmail()

```
method GetAttachedEmail(msgpart As %Net.MailMessagePart,
       Output mailmsg As %Net.MailMessage) as %Status
```

メッセージ部分を指定すると、このメソッドは、メッセージ部分からのデータで初期化されたシングルパートの電子メール・メッセージを（出力パラメータとして）返します。

GetMessageUID()

```
method GetMessageUID(MessageNumber As %Integer,
                     ByRef UniqueID As %String) as %Status
```

メッセージ番号を指定すると、メッセージの UID を（参照によって）返します。メッセージ番号およびUID についての詳細は、前のセクションを参照してください。メッセージが現在削除するようマーク付けされている場合、このメソッドは、エラー・ステータスを返します。

GetMessageUIDArray()

```
method GetMessageUIDArray(MessageNumber As %String = "",
                          ByRef ListOfUniqueIDs As %ArrayOfDataTypes) as %Status
```

最初の引数として空の文字列が指定されている場合、このメソッドは、メールボックス内のメッセージについての情報の配列を、参照によって返します（現在削除するようマーク付けされているものを除く）。この配列の各要素には、1 つのメッセージについての以下の情報が含まれています。

配列キー	配列項目
現在の状態でのメールボックス内のメッセージの番号。最初のメッセージが 1 番で、以降順次番号が割り当てられます。 指定したメッセージのメッセージ番号がすべてのセッションで同じであることは保証されません。	すべてのセッションで使用できる、このメッセージの永久の識別子である、一意のメッセージ識別子（UID）。 UID は、各メールボックスに対して一意です。

あるいは、メッセージ番号を指定すると、このメソッドは、そのメッセージの UID を含む 1 要素の配列を返します。この場合、メッセージが現在削除するようマーク付けされている場合、このメソッドは、エラー・ステータスを返します。

GetSizeOfMessages()

```
method GetSizeOfMessages(MessageNumber As %String = "",
                         ByRef ListOfSizes As %ArrayOfDataTypes) as %Status
```

最初の引数として空の文字列が指定されている場合、このメソッドは、メールボックス内のメッセージについての情報の配列を、参照によって返します（現在削除するようマーク付けされているものを除く）。この配列の各要素には、1 つのメッセージについての以下の情報が含まれています。

配列キー	配列項目
現在の状態でのメールボックス内のメッセージの番号。	このメッセージのサイズ（単位はバイト）。

あるいは、メッセージ番号を指定すると、このメソッドは、そのメッセージのサイズ（バイト単位）を含む 1 要素の配列を返します。この場合、メッセージが現在削除するようマーク付けされている場合、このメソッドは、エラー・ステータスを返します。

6.10 メッセージの削除

メールボックスに接続している間は、そのメールボックス内で削除するようメッセージにマーク付けすることができます。これは、以下の 2 つの方法で行うことができます。

- これには DeleteMessage() メソッドを使用できます。このメソッドは、削除するメッセージ番号である 1 つの引数を取ります。
- Fetch() メソッドまたは FetchMessage() メソッドによってメッセージを取得する際、メッセージ取得後に削除するようメッセージにマーク付けするように POP3 サーバに指示するオプションの引数を指定できます。

以下の点に留意してください。

- これらのメソッドは、メッセージを削除しません。削除するようこれにマーク付けします。メッセージは、QuitAndCommit() によって POP3 トランザクションを完了するまで削除されません。単にサーバから切断すると、変更は破棄されます。
- RollbackDeletes() メソッドを呼び出してメッセージを変更し、削除のマーク付けを解除することができます。
- これらの各メッセージはステータスを返すので、それを確認する必要があります。

6.10.1 例 : GetMsgAndDelete() および CommitChanges()

以下の例は、メッセージを取得し、削除するようマーク付けします。

Class Member

```
ClassMethod GetMsgAndDelete(ByRef server As %Net.POP3, msgno As %Integer) As %Net.MailMessage
{
  //third argument to Fetch says whether to
  //mark for deletion
  Set status=server.Fetch(msgno,.msg,1)
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Quit $$$NULLOREF
  }

  Quit msg
}
```

このメッセージは、%Net.POP3 の変更バージョンを (参照によって) 返します。変更バージョンには、どのメッセージが削除するようマーク付けされているかについての情報が含まれています。

前述のメソッドを以下のようなメソッドと共に使用します。

Class Member

```
ClassMethod CommitChanges(server As %Net.POP3) As %Status
{
  //commit all changes and log out
  Set status=server.QuitAndCommit()
  If $$$ISERR(status) {
    Do $System.Status.DisplayError(status)
    Quit $$$ERROR($$$GeneralError,"Failed to commit changes")
  }
  Quit $$$OK
}
```

または、RollbackDeletes() や QuitAndRollback() によって変更をロールバックします。

6.11 関連項目

- InterSystems IRIS における電子メールのサポート
- %Net.FetchMailProtocol (例や広範なコメント)
- 受信した電子メールの操作

7

受信した電子メールの操作

ここでは、受信した電子メール・メッセージ (%Net.MailMessage) の操作方法について説明します。

7.1 メッセージの基本

電子メール・メッセージ (%Net.MailMessage) を取得後、一般には、そのメッセージの種類とその読み取り方法 (それがマルチパート・メッセージであるかどうか、およびその部分がバイナリであるかどうか) をまず判断します。この手順では、ContentType プロパティを使用できます。あるいは、間接的に ContentType プロパティと同じ情報を提供する IsBinary プロパティ、IsHTML プロパティ、および IsMultiPart プロパティを使用することもできます。

メッセージがマルチパート・メッセージの場合、各部分は、%Net.MailMessagePart のインスタンスです。

7.2 メッセージ・ヘッダ

メッセージ自体およびメッセージの各部分にはどちらも、1 組みのヘッダがあります。

%Net.MailMessage クラスおよび %Net.MailMessagePart クラスは、最も一般的に使用されるヘッダに簡単にアクセスできるプロパティを提供します。例えば、%Net.MailMessage は、To、From、Subject、Date などのプロパティを提供します。Headers 配列プロパティによって、カスタム・ヘッダにアクセスできます。"電子メール・メッセージ・ヘッダの指定" を参照してください。

また、%Net.POP3 によってメッセージを取得した場合、GetAttribute() メソッドを使用することができます。ヘッダ名と属性を指定すると、このメソッドは、その属性の値を返します。

7.3 メッセージ・コンテンツ

全般的なメッセージ構造がわかったら、以下の方法でコンテンツを取得します。

- マルチパート・メッセージの場合、部分の配列である Parts プロパティを使用します。`Parts.Count()` によって、部分の数が得られます。各部分のキーは、1 で始まる整数です。GetAt() メソッドを使用して、指定部分を取得します。メッセージ部分は、%Net.MailMessagePart のインスタンスです。

  %Net.MailMessage と %Net.MailMessagePart の関係については、"InterSystems IRIS で MIME 電子メール・メッセージを表す方法" を参照してください。

- バイナリ・メッセージ (またはメッセージ部分) の場合、BinaryData プロパティを使用します。
- テキスト・メッセージ (またはメッセージ部分) の場合、TextData プロパティを使用します。
  - IsHTML が 0 の場合、TextData プロパティは通常のテキスト文字列です。
  - IsHTML が 1 の場合、TextData プロパティは HTML テキスト文字列です。

メッセージを送信する電子メール・クライアントがメッセージ内のラップを決定します。メール・サーバも InterSystems IRIS も、これを制御できません。

7.4 その他のメッセージ情報

MessageSize プロパティは、添付された電子メール・メッセージ以外に、メッセージ長の合計を示します。

以下のメソッドは、メッセージについての追加情報を提供します。

GetLocalDateTime()

メッセージが取得された日付と時刻を、$HOROLOG 形式のローカル時間に変換して返します。

GetUTCDateTime()

メッセージが取得された日付と時刻を、$HOROLOG 形式の UTC に変換して返します。

GetUTCSeconds()

メッセージが取得された日付と時刻を 12/31/1840 からの秒数で返します。

日付/時刻変換では、次のクラス・メソッドも使用できます。

HToSeconds()

$HOROLOG 形式の日付および時刻を 12/31/1840 からの秒数に変換するクラス・メソッド。

SecondsToH()

12/31/1840 からの秒数を $HOROLOG 形式の日付および時刻に変換するクラス・メソッド。

7.5 例 1 : ShowMsgInfo()

%Net.MailMessage のインスタンスを指定すると、以下のメソッドは、メッセージについての情報を現在のデバイスに書き込みます。

Class Member

```
ClassMethod ShowMsgInfo(msg as %Net.MailMessage)
{
    Write "Message details *****",!
    Write "To (count): ", msg.To.Count(),!
    Write "From: ", msg.From,!
    Write "Cc (count): ", msg.Cc.Count(),!
    Write "Bcc (count): ", msg.Bcc.Count(),!
    Write "Date: ", msg.Date,!
    Write "Subject: ", msg.Subject,!
    Write "Sender: ", msg.Sender,!
    Write "IsMultipart: ", msg.IsMultiPart,!
    Write "Number of parts: ", msg.Parts.Count(),!
    Write "Number of headers: ", msg.Headers.Count(),!
    Write "IsBinary: ", msg.IsBinary,!
    Write "IsHTML: ", msg.IsHTML,!
    Write "TextData: ", msg.TextData.Read(),!
    Write "BinaryData: ", msg.BinaryData.Read(),!
}
```

このメソッドは、以下のような出力を生成します。

```
Message details *****
To (count): 1
From: "XXX XXX" <XXX@XXX.com>
Cc (count): 0
Bcc (count): 0
Date: Fri, 16 Nov 2023 11:57:46 -0500
Subject: test 5
Sender:
IsMultipart: 0
Number of parts: 0
Number of headers: 16
IsBinary: 0
IsHTML: 0
TextData: This is test number 5, which is plain text.
BinaryData:
```

7.6 例 2 : ShowMsgPartInfo()

以下のメソッドは、メッセージの部分についての情報を記述します。

Class Member

```
ClassMethod ShowMsgPartInfo(msg as %Net.MailMessage, partno as %Integer)
{
    Set part=msg.Parts.GetAt(partno)
    Write "Message part details *****",!
    Write "Message part: ", partno,!
    Write "IsMultipart: ", part.IsMultiPart,!
    Write "Number of parts: ", part.Parts.Count(),!
    Write "Number of headers: ", part.Headers.Count(),!
    Write "IsBinary: ", part.IsBinary,!
    Write "IsHTML: ", part.IsHTML,!
    Write "TextData: ", part.TextData.Read(),!
    Write "BinaryData: ", part.BinaryData.Read(),!
}
```

これは、以下のような出力を生成します (前に示したメッセージとは異なるメッセージを指定した場合)。

```
Message part details *****
Message part: 1
IsMultipart: 0
Number of parts: 0
Number of headers: 2
IsBinary: 0
IsHTML: 0
TextData: 1 test string


BinaryData:
```

7.7 例 3 : ShowMsgHeaders()

以下のメソッドは、メッセージのヘッダについての情報を記述します。メッセージ部分に対して同じことを行う同様のメソッドを記述することができます。

Class Member

```
ClassMethod ShowMsgHeaders(msg as %Net.MailMessage)
{
    Set headers=msg.Headers
    Write "Number of headers: ", headers.Count(),!

    //iterate through the headers
    Set key=""
    For {
        Set value=headers.GetNext(.key)
        Quit:key=""
        Write "Header:",key,!
        Write "Value: ",value,!!
    }
}
```

これは、以下のような出力を生成します。

```
Number of headers: 16
Header: content-class
Value: urn:content-classes:message

Header: content-type
Value: multipart/alternative; boundary="----_=_NextPart_001_01C8286D.D9A7F3B1"

Header: date
Value: Fri, 16 Nov 2023 11:29:24 -0500

Header: from
Value: "XXX XXX" <XXX@XXX.com>

Header: message-id
Value: <895A9EF10DBA1F46A2DDB3AAF061ECD501801E86@Exchange1_backup>

Header: mime-version
Value: 1.0

...
```

7.8 関連項目

- InterSystems IRIS における電子メールのサポート
- POP3 を介した電子メールの取得
- MIME メッセージの作成、記述および読み取り

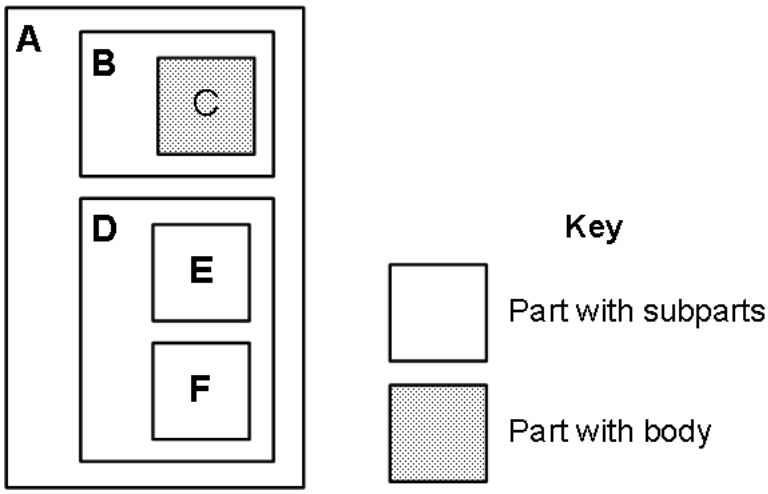
8

MIME メッセージの作成、記述および読み取り

InterSystems IRIS® データ・プラットフォームでは、マルチパート MIME メッセージの作成に使用できるクラス (%Net.MIMEPart) を提供します。SOAP メッセージに追加する添付を作成するには、このクラスを使用します。"Web サービスおよび Web クライアントの作成" を参照してください。電子メールの処理や HTTP マルチパート POST など、他にも多くのアプリケーションが考えられます。

8.1 概要

MIME 形式のドキュメントは、MIME パートと呼ばれます。各 MIME パートにはヘッダがあり、メッセージ本文 (テキストまたはバイナリのいずれか)、または追加 MIME パートが含まれます。`MIME-Version` ヘッダを持つ MIME パートは、トップレベル・ドキュメントとして使用でき、MIME メッセージと呼ばれます。次の図に例を示します。



この例で、E と F には、表示されない追加サブパートがあります。

MIME パートを表示するには、ヘッダおよびこのパートのコンテンツの設定に使用するプロパティを提供する %Net.MIMEPart クラスを使用します。

8.2 MIME パートの作成

MIME パートを作成する手順は以下のとおりです。

1. %Net.MIMEPart のインスタンスを作成します。
2. 以下のいずれかを行います。
   - テキストまたはバイナリの本文を追加します。そのためには、ストリーム (テキストまたはバイナリのいずれか) のインスタンスを作成し、MIME パートの Body プロパティをこのストリームと等しくなるように設定します。データをこのストリームに書き込むには、標準のストリーム・インタフェースを使用します。Parts プロパティの値は指定しないでください。
   - MIME パートのリストを追加します。そのためには、ここで説明したとおりに MIME パートを作成し、Parts プロパティをこれらのパートのリストと等しくなるように設定します。Body プロパティの値は指定しないでください。
3. オプションで、"MIME パート・ヘッダの設定と取得" の説明に従って、ヘッダを設定します。

8.2.1 MIME パート・ヘッダの設定と取得

HTTP ヘッダの値を設定および取得できます。%Net.MIMEPart の以下のプロパティが MIME ヘッダに影響します。

- ContentType — `Content-Type` ヘッダのインターネット・メディア・タイプ (MIME タイプ)。これは、Body データのインターネット・メディア・タイプを指定します。例えば、`"text/plain"`、`"text/html"`、`"image/jpeg"`、`"multipart/mixed"` です。
- ContentCharset — `Content-Type` ヘッダの `charset` パート。これを設定する場合、まず、ContentType プロパティを設定する必要があります。テキストの本文を含む各 MIME パートに対し、本文で使用する文字セットを指定するよう、ContentCharset プロパティを正しく設定しください。%Net.MIMEPart では、変換は行われないため、このプロパティでは、既に使用されている文字セットを宣言する必要があります。
- ContentId — 山括弧 (<>) および先頭と末尾の空白のない、正規化された `Content-ID` ヘッダ。
- ContentLocation — 先頭と末尾の空白のない、正規化された `Content-Location` ヘッダ。
- ContentTransferEncoding — `Content-Transfer-Encoding` ヘッダ。このプロパティは、`"base64"`、`"quoted-printable"`、`"7bit"`、`"8bit"` のいずれかです。

  既定値はありません。

  重要　コンテンツを `"base64"` でエンコードする場合、Unicode 文字を含めることはできません。送信するコンテンツに Unicode 文字が含まれている場合は必ず、$ZCONVERT を使用してコンテンツを UTF-8 に変換してから、base-64 でエンコードしてください。以下に例を示します。

```
set BinaryText=$ZCONVERT(UnicodeText,"O","UTF8")
set Base64Encoded=$system.Encryption.Base64Encode(BinaryText)
```

  受信者は、逆のプロセスを使用してテキストをデコードする必要があります。

```
set BinaryText=$system.Encryption.Base64Decode(Base64Encoded)
set UnicodeText=$ZCONVERT(BinaryText,"I","UTF8")
```

%Net.MIMEPart クラスは、MIME ヘッダの管理に使用できる一般的なメソッドを提供します。

- GetHeader() は、ヘッダの値を返します。
- NextHeader() は、次のヘッダを取得します。
- SetHeader() は、ヘッダの値を設定します。通常、これを使用して、標準外ヘッダを設定します。
- RemoveHeader() は、ヘッダを削除します。

完全なメソッド・シグニチャおよびその他の詳細は、%Net.MIMEPart のクラス・ドキュメントを参照してください。

8.2.2 オプションのメッセージ範囲の値を指定

既定では、メッセージ範囲は、自動的に生成されます。必要に応じて、メッセージの範囲を指定できます。そのためには、Boundary プロパティの値を指定します。どのメッセージ・パートでも使用される可能性が極めて低い文字列を使用するようにしてください。

8.3 MIME メッセージの記述

MIME メッセージを記述するには、以下のように %Net.MIMEWriter を使用します。

1. %Net.MIMEWriter クラスのインスタンスを作成します。
2. オプションで、出力先を指定します。出力先を指定するには、ライター・インスタンスの OutputToDevice() (既定)、OutputToFile()、または OutputToStream() のメソッドのいずれかを使用します。
3. 必要に応じて、ライターのメソッドを呼び出して、出力を記述します。
   - ヘッダ名および値を指定すると、WriteHeader() がそのヘッダを記述します。
   - %Net.MIMEPart のインスタンスを指定すると、WriteMIMEBody() が、メッセージの本文を記述します。これは、複数のパートを持つことが可能です。

     メッセージがマルチパートの場合、このメソッドは、ヘッダを記述しません。ユーザの責任でヘッダを記述してください。ただし、メッセージがマルチパートではない場合、メソッドはヘッダを記述します。
   - %Net.MIMEPart のインスタンスを指定すると、WriteMIMEMessage() は、すべてのヘッダを含む、MIME メッセージを記述します。

   シングルパートのメッセージの場合、WriteMIMEBody() および WriteMIMEMessage() は、同じ出力を生成します。

完全なメソッド・シグニチャおよびその他の詳細は、%Net.MIMEPart のクラス・ドキュメントを参照してください。

8.3.1 例 : WriteMIMEMessage()

以下の例は、WriteMIMEMessage() の使用法を示しています。

Class Member

```
ClassMethod WriteMIMEMessage(text As %String,header as %String) as %Status
{
 Set msg=##class(%Net.MIMEPart).%New()
 Set msg.Body=##class(%GlobalCharacterStream).%New()
 Do msg.Body.Write(text)

 //specify some headers
 Set msg.ContentType="text/html"
 Set msg.ContentCharset="us-ascii"
 Do msg.SetHeader("Custom-header",header)

 //create MIME writer; write MIME message
 Set writer=##class(%Net.MIMEWriter).%New()
 Set status=writer.WriteMIMEMessage(msg)

 If $$$ISERR(status) do $system.Status.DisplayError(status)
 Quit $$$OK
}
```

以下のターミナル・セッションでは、このメソッドが使用されています。

```
GNET> Set text = "message text"

GNET> Set header="my header value"

GNET> Do ##class(GNET.MIME).WriteMIMEMessage(text,header)
CONTENT-TYPE: text/html
Custom-header: my header value

message text

GNET>
```

8.4 MIME メッセージの読み取り

MIME メッセージを読み取るには、以下のように %Net.MIMEReader を使用します。

1. %Net.MIMEReader クラスのインスタンスを作成します。
2. 入力のソースを指定します。入力のソースを指定するには、リーダ・インスタンスの OpenFile() メソッドまたは OpenStream() メソッドを使用します。
3. リーダ・インスタンスの ReadMIMEMessage() メソッドを呼び出します。このメソッドは、最初の引数として %Net.MIMEPart のインスタンスを参照によって返します。これはステータスを返すので、それを確認する必要があります。

完全なメソッド・シグニチャおよびその他の詳細は、%Net.MIMEPart のクラス・ドキュメントを参照してください。

9

FTP の使用法

InterSystems IRIS® データ・プラットフォームは、InterSystems IRIS 内から FTP サーバとのセッションを確立する際に使用できる %Net.FtpSession クラスを提供します。

9.1 FTP セッションの確立

FTP セッションを確立する手順は次のとおりです。

1. %Net.FtpSession のインスタンスを作成します。
2. オプションで、セッションの一般的な動作を制御するために、このインスタンスのプロパティを設定します。
   - Timeout は、FTP サーバからの応答を待機する時間を秒単位で指定します。
   - SSLConfiguration は、接続に使用する有効化された SSL/TLS 構成を指定します (存在する場合)。FTP サーバが https を使用する場合にこれを使用します。

     SSL/TLS 構成の作成と管理の詳細は、インターシステムズの "TLS ガイド" を参照してください。SSL/TLS 構成には、[構成名] と呼ばれるオプションが含まれています。これは、この設定で使用する文字列です。
   - TranslateTable は、ファイルの内容の読み取りまたは書き込みの際に使用する変換テーブルを指定します。

     指定した文字セットのテーブル名を検索するには %Net.Charset クラスを使用します。
   - UsePASV は、PASV モードを有効化します。
   - SSLCheckServerIdentity は、FTP サーバが https を使用する場合に適用されます。既定では、%Net.FtpSession のインスタンスは、SSL/TLS サーバに接続するときに、証明書サーバ名と、そのサーバへの接続に使用した DNS 名が一致しているかどうかを確認します。これらの名前が一致していないと、接続は許可されません。この既定の動作は、中間者攻撃の防止を目的としています。RFC 2818 のセクション 3.1 に説明があります。また、RFC 2595 のセクション 2.4 も参照してください。

     この確認を無効にするには、SSLCheckServerIdentity プロパティを 0 に設定します。
   - SSLUseSessionResumption は、true の場合、データ・チャンネルの SSL 接続を作成するときに、セッション・パラメータをコマンド・チャンネルから再利用するよう指定します。この機能には OpenSSL v1.1.x 以降が必要です。OpenSSL v1.0.x を使用しているインスタンスにこのフラグを設定した場合、無視されます。
3. 特定の FTP サーバに接続するには、Connect() メソッドを呼び出します。
4. 転送モードを ASCII モードまたはバイナリ・モードに設定するには、それぞれ、Ascii() メソッドまたは Binary() メソッドを呼び出します。現在の転送モードを参照するには、インスタンスの Type プロパティの値を確認します。

注釈　%Net.FtpSession の各メソッドはステータスを返すので、それを確認する必要があります。これらのメソッドは、セッションの状態についての有益な情報を提供するプロパティの値も設定します。

- Connected は、現在接続されている場合は True、接続されていない場合は False です。
- ReturnCode には、前回の FTP サーバとの通信からの返りコードが含まれています。
- ReturnMessage には、前回の FTP サーバとの通信からの返りメッセージが含まれています。

Status() メソッドは、FTP サーバのステータスを (参照によって) 返します。

詳細は、%Net.FtpSession のクラス・ドキュメントを参照してください。

9.1.1 コマンド用のテーブルの変換

%Net.FtpSession は、FTP サーバでファイル名とパス名を検索するときに、RFC 2640 で説明されている手法を使用して、文字セットの変換を自動的に処理します。%Net.FtpSession のインスタンスは FTP サーバに接続するときに、FEAT メッセージを使用して、サーバが UTF-8 文字であるかどうかを判断します。UTF-8 の場合は、コマンド・チャネルの通信を UTF-8 に切り替えて、すべてのファイル名およびパス名の UTF-8 への変換と UTF-8 からの変換が適切に行われるようにします。

サーバが FEAT コマンドをサポートしていない場合や、UTF-8 のサポートを報告しない場合、%Net.FtpSession インスタンスは RAW モードを使用して未加工のバイトを読み取りまたは書き込みします。

特殊な状況では、使用する変換テーブルの指定が必要になることがあります。その場合は、%Net.FtpSession インスタンスの CommandTranslateTable プロパティを設定します。通常は、このプロパティを使用する必要はありません。

9.2 FTP ファイルおよびシステム・メソッド

FTP セッションを確立したら、セッション・インスタンスのメソッドを呼び出して、FTP タスクを実行します。%Net.FtpSession は、ファイルの読み取りおよび書き込みのための以下のメソッドを提供します。

Delete()

ファイルを削除します。

Retrieve()

ファイルを FTP サーバから InterSystems IRIS ストリームにコピーし、参照によってストリームを返します。このストリームを使用するには、標準ストリーム・メソッドである Write()、WriteLine()、Read()、ReadLine()、Rewind()、MoveToEnd()、および Clear() を使用します。また、ストリームの Size プロパティも使用できます。

RetryRetrieve()

以前 Retrieve() を使用して作成されたストリームが指定されている場合には、続けてファイルを取得できます。

Store()

InterSystems IRIS ストリームのコンテンツを FTP サーバのファイルに書き込みます。

StoreFiles()

ローカル・ディレクトリとワイルドカード・マスクを指定すると、このメソッドによってそのディレクトリに複数のファイルが書き込まれます。このメソッドではディレクトリが無視され、現在の転送モード (バイナリまたは ASCII) が使用されます。つまり、バイナリ・ファイルと ASCII ファイルを 1 回の呼び出しで同時にアップロードすることはできません。

Append()

ストリームの内容を指定したファイルの最後に追加します。

Rename()

ファイルの名前を変更します。

さらに、%Net.FtpSession は、FTP サーバ上でファイル・システムをナビゲートおよび変更するためのメソッド、GetDirectory()、SetDirectory()、SetToParentDirectory()、および MakeDirectory() を提供します。

ファイル・システムのコンテンツを調べるには、List() メソッドまたは NameList() メソッドを使用します。

- List() は、名前が指定されたパターンと一致するすべてのファイルのリストを含むストリームを作成し、このストリームを参照によって返します。
- NameList() は、ファイル名の配列を作成し、この配列を参照によって返します。

ChangeUser() メソッドを使用して、別のユーザに変更することもできます。この方が、ログアウトして、再びログインするよりも早いです。ログアウトするには、Logout() メソッドを使用します。

System() メソッドは、(参照によって) FTP サーバをホストしているコンピュータのタイプについての情報を返します。

Size() および MDTM() メソッドは、ファイルのサイズとファイルの変更日時をそれぞれ返します。

FTP サーバにコマンドを送信して、応答を読み取るには、汎用メソッド sendCommand() を使用します。このメソッドは、%Net.FtpSession で明示的にはサポートされていないコマンドの送信に使用される場合があります。

詳細は、%Net.FtpSession のクラス・ドキュメントを参照してください。

9.3 リンクされたストリームの使用による大きなファイルのアップロード

大きなファイルをアップロードする場合、ストリーム・インタフェースの LinkToFile() メソッドを使用することを検討してください。つまり、ストリームを作成してファイルをその中に読み込む代わりに、ストリームを作成してそれをファイルにリンクします。%Net.FtpSession の Store() メソッドを呼び出すときに、このリンクされたストリームを使用します。

以下はその例です。

Class Member

```
Method SendLargeFile(ftp As %Net.FtpSession, dir As %String, filename As %String)
{
    Set filestream=##class(%FileBinaryStream).%New()
    Set sc=filestream.LinkToFile(dir_filename)
    If $$$ISERR(sc) {do $System.Status.DisplayError(sc) quit }

    //Uploaded file will have same name as the original
    Set newname=filename

    Set sc=ftp.Store(newname,filestream)
    If $$$ISERR(sc) {do $System.Status.DisplayError(sc) quit }
}
```

9.4 FTP サーバが発行するコールバックのカスタマイズ

FTP サーバによって生成されたコールバックをカスタマイズできます。そうすることによって、例えば、ユーザにサーバがまだ大きな転送を行っていることを示したり、ユーザが転送を中止できるようにすることができます。

FTP コールバックをカスタマイズするには、以下の手順を実行します。

1. %Net.FtpCallback のサブクラスを作成します。
2. このサブクラスで、RetrieveCallback() メソッドを実装します。このメソッドは、FTP サーバからデータを受信するときに定期的に呼び出されます。
3. StoreCallback() メソッドも実装します。このメソッドは、FTP サーバにデータを書き込むときに定期的に呼び出されます。
4. "FTP セッションの確立" で説明されているとおり FTP セッションを作成したら、%Net.FtpCallback のサブクラスと等しくなるように Callback プロパティを設定します。

詳細は、%Net.FtpCallback のクラス・ドキュメントを参照してください。

10

IBM WebSphere MQ メッセージの送受信

InterSystems IRIS® データ・プラットフォームは、IBM WebSphere MQ へのインタフェースを提供しており、これを使用して、InterSystems IRIS と IBM WebSphere MQ のメッセージ・キューとの間でメッセージをやり取りできます。このインスタンスを使用するには、IBM WebSphere MQ サーバにアクセスでき、IBM WebSphere MQ クライアントが InterSystems IRIS と同じマシン上で稼動している必要があります。

このインタフェースは、**%Net.MQSend** クラスおよび **%Net.MQRecv** クラスから構成されています。これらのクラスは共に **%Net.abstractMQ** のサブクラスです。すべての適切なプラットフォームで InterSystems IRIS によって自動的にインストールされるダイナミック・リンク・ライブラリをこれらのクラスは使用します (これは、Windows では、**MQInterface.dll** です。ファイルの拡張子は、その他のプラットフォームでは異なります)。同様に、InterSystems IRIS ダイナミック・リンク・ライブラリは、IBM WebSphere MQ ダイナミック・リンク・ライブラリを必要とします。

インタフェースは、テキスト・データのみの送受信をサポートし、バイナリ・データの送受信はサポートしません。

IBM WebSphere MQ を使用するには、この製品の正式なドキュメントが必要です。また、IBM WebSphere MQ への InterSystems IRIS インタフェースについての追加情報は、**%Net.abstractMQ** のクラス・リファレンスを参照してください。

10.1 概要

通常、IBM WebSphere MQ への InterSystems IRIS インタフェースを使用するには、次の操作を実行します。

1. IBM WebSphere MQ v7.x 以降にアクセスできることを確認します。具体的には以下の手順を実行します。
   - IBM WebSphere MQ クライアントが InterSystems IRIS と同じマシンにインストールされている必要があります。インストーラは、PATH 環境変数を更新し、必要に応じてその他のシステム変数を追加します。
   - InterSystems IRIS がクライアントを認識するように、クライアントをインストールした後、マシンをリブートしたことを確認します。
   - クライアントは、IBM WebSphere MQ サーバにアクセスできる必要があります。
   - サーバにアクセスするユーザ名には、キュー・マネージャおよび使用予定のキューを使用する権限が必要です。
2. メッセージを送信するのか、受信するのかによって、**%Net.MQSend** または **%Net.MQRecv** の新規インスタンスを作成します。
3. IBM WebSphere MQ サーバに接続します。このためには、以下の情報を指定します。
   - キュー・マネージャの名前。
   - 使用するキューの名前。

- そのキューと通信するチャネル。チャネル名、転送メカニズム、および IBM WebSphere MQ サーバの IP アドレスとポートを指定します。

IBM WebSphere MQ の認証機能を使用している場合、名前とパスワードも入力できます。

4. メッセージを送信または受信するには、%Net.MQSend または %Net.MQRecv の適切なメソッドを呼び出します。

注釈　64 ビットの Linux プラットフォームで IBM Websphere MQ を使用するには、MQ ライブラリの場所が含まれるように LD_LIBRARY_PATH を設定する必要があります。このパスは、MQ インタフェースを使用するすべての InterSystems IRIS プロセスに対して設定されていなければならないため、バックグラウンド処理を行う場合は、InterSystems IRIS の起動よりも前に設定する必要があります。また、これは iris terminal の実行前にすべての UNIX® ターミナルで設定します。

10.1.1 エラー・コードの取得

%Net.MQSend および %Net.MQRecv のメソッドは、成功した場合 1 を返し、成功しない場合 0 を返します。エラーの場合は、%GetLastError() メソッドを呼び出します。このメソッドは、IBM WebSphere MQ によって提供された最後の理由コードを返します。理由コードの詳細は、正式な IBM ドキュメントを参照してください。

10.2 Connection オブジェクトの生成

IBM WebSphere MQ によりメッセージを送信または受信する前に、接続オブジェクト (キュー・マネージャへの接続を確立し、チャネルを開き、使用するキューを開くオブジェクト) を作成する必要があります。これを行うには、以下の 2 つの方法があります。

- %Init メソッドを使用できます。このメソッドは、すべての必要な情報を指定する引数を取ります。
- すべての必要な情報を指定するプロパティを設定した後、%Connect メソッドを使用できます。

10.2.1 %Init() メソッドの使用法

%Init() メソッドを使用して、接続オブジェクトを作成する手順は以下のとおりです。

1. %Net.MQSend (メッセージを送信する場合) または %Net.MQRecv (メッセージを受信する場合) のインスタンスを作成します。このトピックでは、このインスタンスを接続オブジェクトと呼びます。

   注釈　<DYNAMIC LIBRARY LOAD> エラーが発生した場合、ダイナミック・リンク・ライブラリはなく、詳細は、(システム・マネージャのディレクトリ内の) messages.log ファイルに記載されます。

2. 接続オブジェクトの %Init() メソッドを呼び出します。このメソッドは、順番に以下の引数を取ります。
   a. (必須) キュー名を指定する文字列。これは、指定したキュー・マネージャに対して有効なキューである必要があります。
   b. キュー・マネージャを指定する文字列。これは、IBM WebSphere MQ サーバ上で有効なキュー・マネージャである必要があります。

      この引数を省略すると、システムは、IBM WebSphere MQ で構成された既定のキュー・マネージャを使用します。あるいは、キュー名によってキュー・マネージャを決定するように IBM WebSphere MQ を構成した場合、システムは、指定したキュー名に適したキュー・マネージャを使用します。

c. 以下の形式の、チャネルの仕様を指定する文字列。

```
"channel_name/transport/host_name(port)"
```

ここで、channel_name は使用するチャネルの名前、transport はチャネルが使用する転送、host_name は IBM WebSphere MQ サーバを実行しているサーバ名 (または IP アドレス)、port はこのチャネルが使用するポートです。

Transport は、**TCP**、**LU62**、**NETBIOS**、**SPX** のいずれかです。

以下はその例です。

```
"CHAN_1/TCP/rodan(1401)"
```

```
"CHAN_1/TCP/127.0.0.1(1401)"
```

この引数を省略すると、システムは、IBM WebSphere MQ で構成された既定のチャネル仕様を使用します。あるいは、キュー名によってチャネルを決定するようにシステムを構成した場合、システムは、指定したキュー名に適したチャネルを使用します。

d. エラー・メッセージを書き込むログ・ファイルを指定するオプションの文字列。既定の設定は、ログ記録しない設定です。

3. %Init() メソッドによって返される値を確認します。メソッドが 1 を返した場合、接続は正常に確立されており、接続オブジェクトを使用して、(使用しているクラスによって) メッセージを送受信することができます。"エラー・コードの取得" を参照してください。

10.2.2 %Connect() メソッドの使用法

場合によっては、接続のすべての詳細を個別に指定したいことがあります。このためには、以下のように %Connect() メソッドを使用します。

1. **%Net.MQSend** (メッセージを送信する場合) または **%Net.MQRecv** (メッセージを受信する場合) のインスタンスを作成します。前述のとおり、このトピックでは、このインスタンスを接続オブジェクトと呼びます。

   注釈　〈DYNAMIC LIBRARY LOAD〉 エラーが発生した場合、ダイナミック・リンク・ライブラリはなく、詳細は、(システム・マネージャのディレクトリ内の) **messages.log** ファイルに記載されます。

2. 接続オブジェクトの以下のプロパティを設定します。
   - **QName** ― (必須) キュー名を指定します。これは、指定したキュー・マネージャに対して有効なキューである必要があります。
   - **QMgr** ― 使用するキュー・マネージャを指定します。これは、IBM WebSphere MQ サーバ上で有効なキュー・マネージャである必要があります。

     この引数を省略すると、システムは、IBM WebSphere MQ で構成された既定のキュー・マネージャを使用します。あるいは、キュー名によってキュー・マネージャを決定するように IBM WebSphere MQ を構成した場合、システムは、指定したキュー名に適したキュー・マネージャを使用します。

3. オプションで、接続オブジェクトの以下のプロパティを設定して、使用するチャネルを指定します。
   - **Connection** ― IBM WebSphere MQ サーバのホストおよびポートを指定します。例えば、**"127.0.0.1:1401"** です。
   - **Channel** ― 使用するチャネル名を指定します。これは、IBM WebSphere MQ サーバ上で有効なチャネルである必要があります。

- Transport – チャネルが使用する転送を指定します。このプロパティは、"TCP"、"LU62"、"NETBIOS"、または "SPX" のいずれかです。

これらの引数を省略すると、システムは、IBM WebSphere MQ で構成された既定のチャネル仕様を使用します。あるいは、キュー名によってチャネルを決定するようにシステムを構成した場合、システムは、指定したキュー名に適したチャネルを使用します。

4. 接続オブジェクトの %ErrLog() メソッドを呼び出します。このメソッドは、この接続オブジェクトに使用するログ・ファイルの名前である、1 つの引数を取ります。
5. %ErrLog() メソッドによって返される値を確認します。"エラー・コードの取得" を参照してください。
6. 接続オブジェクトの %Connect() メソッドを呼び出します。
7. %Connect() メソッドによって返される値を確認します。メソッドが 1 を返した場合、接続は正常に確立されており、接続オブジェクトを使用して、(使用しているクラスによって) メッセージを送受信することができます。"エラー・コードの取得" を参照してください。

10.3 文字セット (CCSID) の指定

メッセージ変換に使用される文字セットを設定するには、接続オブジェクトの %SetCharSet() メソッドを呼び出します。IBM WebSphere MQ で使用される整数のコード化文字セット識別コード (CCSID) を指定します。

- メッセージを送信する場合、これは、それらのメッセージの文字セットである必要があります。文字セットを指定しない場合、MQ システムでは、メッセージは MQ クライアントに対して指定された既定の文字セットを使用するものと見なします。
- メッセージを取得する場合、これは、これらのメッセージ変換後の文字セットです。

現在使用されている CCSID を取得するには、%CharSet() メソッドを呼び出します。このメソッドは、参照によって CCSID を返し、成功したかどうかを示す 1 または 0 を返します。"エラー・コードの取得" を参照してください。

指定した文字セットに該当する CCSID の詳細は、正式の IBM ドキュメントを参照してください。

10.4 その他のメッセージ・オプションの指定

メッセージ記述子オプションを指定するには、オプションで、接続オブジェクトの以下のプロパティを設定します。

- ApplIdentityData は、アプリケーション ID メッセージ記述子オプションを指定します。
- PutApplType は、メッセージを入れたアプリケーションのタイプ・メッセージ記述子オプションを指定します。

10.5 メッセージの送信

メッセージを送信するには、以下の手順を実行します。

1. "Connection オブジェクトの作成" の説明に従って、接続オブジェクトを作成します。この場合、%Net.MQSend のインスタンスを作成します。接続オブジェクトには、メッセージを送信できるメッセージ・キューがあります。
2. 必要に応じて、以下のメソッドを呼び出します。

- %Put() – 文字列を指定すると、このメソッドは、その文字列をメッセージ・キューに書き込みます。
- %PutStream() – 初期化済みファイルの文字ストリームを指定すると、このメソッドは、その文字列をメッセージ・キューに書き込みます。これを初期化するには、ストリームの Filename プロパティを設定する必要があります。バイナリ・ストリームはサポートされません。
- %SetMsgId() – 文字列を指定すると、このメソッドは、その文字列を次に送信されるメッセージのメッセージ ID として使用します。

3. 呼び出したメソッドによって返される値を確認します。"エラー・コードの取得" を参照してください。
4. メッセージの取得が完了したら、接続オブジェクトの %Close() メソッドを呼び出して、ダイナミック・リンク・ライブラリへのハンドルを解放します。

10.5.1 例 1 : SendString()

以下のクラス・メソッドは、QM_antigua キュー・マネージャ、および S_antigua という名前のキュー・チャネルを使用して、単純な文字列のメッセージを mqtest キューに送信します。チャネルは TCP 転送を使用し、IBM WebSphere MQ サーバは antigua と呼ばれるマシンで稼動し、ポート 1401 で待ち受け状態になっています。

Class Member

```
///Method returns reason code from IBM WebSphere MQ
ClassMethod SendString() As %Integer
{
 Set send=##class(%Net.MQSend).%New()
 Set queue="mqtest"
 Set qm="QM_antigua"
 Set chan="S_antigua/TCP/antigua(1414)"
 Set logfile="c:\mq-send-log.txt"

 Set check=send.%Init(queue,qm,chan,logfile)
 If 'check  Quit send.%GetLastError()

 //send a unique message
 Set check=send.%Put("This is a test message "_$h)

 If 'check  Quit send.%GetLastError()
 Quit check
}
```

10.5.2 例 2 : SendCharacterStream()

以下のクラス・メソッドは、ファイル文字ストリームのコンテンツを送信します。前の例で使用したキューと同じキューを使用します。

Class Member

```
///Method returns reason code from IBM WebSphere MQ
ClassMethod SendCharacterStream() As %Integer
{
 Set send=##class(%Net.MQSend).%New()
 Set queue="mqtest"
 Set qm="QM_antigua"
 Set chan="S_antigua/TCP/antigua(1414)"
 Set logfile="c:\mq-send-log.txt"

 Set check=send.%Init(queue,qm,chan,logfile)
 If 'check  Quit send.%GetLastError()

 //initialize the stream and tell it what file to use
 Set longmsg=##class(%FileCharacterStream).%New()
 Set longmsg.Filename="c:\input-sample.txt"

 Set check=send.%PutStream(longmsg)
```

```
 If 'check  Quit send.%GetLastError()
 Quit check
}
```

10.5.3 例 3 : ターミナルからのメッセージの送信

以下に、IBM WebSphere MQ キューにメッセージを送信するターミナル・セッションの例を示します。これは、IBM WebSphere MQ クライアントで構成されているマシンでのみ機能します。

```
Set MySendQ = ##class(%Net.MQSend).%New()

Do MySendQ.%Init("Q_1", "QM_1","QC_1/TCP/127.0.0.1(1401)","C:\mq.log")

Do MySendQ.%Put("Hello from tester")

Set MyRecvQ =##class(%Net.MQRecv).%New()

Do MyRecvQ.%Init("Q_1", "QM_1","QC_1","C:\mq.log")

Do MyRecvQ.%Get(.msg, 10000)

Write msg,!
```

その他の例については、前のセクションも参照してください。

10.6 メッセージの取得

メッセージを取得するには、以下の手順を実行します。

1. ″Connection オブジェクトの作成″ の説明に従って、接続オブジェクトを作成します。この場合、%Net.MQRecv のインスタンスを作成します。接続オブジェクトには、メッセージを取得できるメッセージ・キューがあります。
2. 必要に応じて、以下のメソッドを呼び出します。
   - %Get() ー 最初の引数として文字列メッセージを参照によって返します。
   - %GetStream() ー 初期化済みのファイルの文字ストリームを指定すると、このメソッドは、メッセージをキューから取得し、それをそのストリームに関連付けられているファイルに配置します。これを初期化するには、ストリームの Filename プロパティを設定する必要があります。バイナリ・ストリームはサポートされません。

   両方のメソッドにおいて、2 番目の引数は、ミリ秒単位でのタイムアウトです。これは、サーバへのアクセスに使用される時間を制御します。既定のタイムアウトは 0 です。
3. 呼び出したメソッドによって返される値を確認します。″エラー・コードの取得″ を参照してください。IBM WebSphere MQ は、キューが空の場合、2033 を返すことに留意してください。
4. メッセージの取得が完了したら、接続オブジェクトの %Close() メソッドを呼び出して、ダイナミック・リンク・ライブラリへのハンドルを解放します。

10.6.1 例 1 : ReceiveString()

以下のクラス・メソッドは、mqtest キューからメッセージを取得します。

Class Member

```
///Method returns string or null or error message
ClassMethod ReceiveString() As %String
{
 Set recv=##class(%Net.MQRecv).%New()
 Set queue="mqtest"
 Set qm="QM_antigua"
```

```
 Set chan="S_antigua/TCP/antigua(1414)"
 Set logfile="c:\mq-recv-log.txt"

 Set check=recv.%Init(queue,qm,chan,logfile)
 If 'check  Quit recv.%GetLastError()

 Set check=recv.%Get(.msg)
 If 'check {
     Set reasoncode=recv.%GetLastError()
     If reasoncode=2033 Quit ""
     Quit "ERROR: "_reasoncode
     }

 Quit msg
}
```

10.6.2 例 2 : ReceiveCharacterStream()

以下のメソッドは、%GetStream() を使用しているため、より長いメッセージを取得できます。

Class Member

```
/// Method returns reason code from IBM WebSphere MQ
ClassMethod ReceiveCharacterStream() As %Integer
{
 Set recv=##class(%Net.MQRecv).%New()
 Set queue="mqtest"
 Set qm="QM_antigua"
 Set chan="S_antigua/TCP/antigua(1414)"
 Set logfile="c:\mq-recv-log.txt"

 Set check=recv.%Init(queue,qm,chan,logfile)
 If 'check  Quit recv.%GetLastError()

 //initialize the stream and tell it what file to use
 //make sure filename is unique we can tell what we received
 Set longmsg=##class(%FileCharacterStream).%New()
 Set longmsg.Filename="c:\mq-received"_$h_".txt"

 Set check=recv.%GetStream(longmsg)

 If 'check  Quit recv.%GetLastError()
 Quit check
}
```

10.7 メッセージ情報の更新

%Net.MQSend クラスおよび %Net.MQRecv クラスも以下のメソッドを提供します。

%CorId()

最後に読み取ったメッセージの相関 ID を (参照によって) 更新します。

%ReplyQMgrName()

最後に読み取ったメッセージの応答キュー・マネージャ名を (参照によって) 更新します。

%ReplyQName()

最後に読み取ったメッセージの応答キュー名を (参照によって) 更新します。

10.8 トラブルシューティング

IBM WebSphere MQ への InterSystems IRIS インタフェースを使用しているときに問題が発生した場合、まず、クライアントが正しくインストールされているか、およびサーバと通信可能であるかを判断する必要があります。そのようなテストを行うには、IBM WebSphere MQ が提供するサンプル・プログラムを使用できます。実行可能プログラムは、IBM WebSphere MQ クライアントの bin ディレクトリにあります。

以下の手順は、これらのサンプル・プログラムを Windows で使用する方法を示します。その他のオペレーティング・システムでは、詳細は異なる場合があります。IBM ドキュメントを参照し、使用しているクライアントにあるファイルの名前を確認してください。

1. MQSERVER という環境変数を作成します。その値は、channel_name/transport/server の形式である必要があります。ここで、channel_name は使用するチャネルの名前、transport は使用する転送を示す文字列、server は、サーバの名前です。例えば、`S_antigua/TCP/antigua` です。
2. コマンド行で、以下のコマンドを入力します。

   ```
   amqsputc queue_name queue_manager_name
   ```

   queue_name は使用するキューの名前、queue_manager_name はキュー・マネージャの名前です。以下はその例です。

   ```
   amqsputc mqtest QM_antigua
   ```

   `amqsputc` コマンドが認識されない場合、PATH 環境変数が IBM WebSphere MQ クライアントの bin ディレクトリを含めるように更新されていることを確認してください。

   その他のエラーの場合は、IBM ドキュメントを参照してください。
3. 以下のような 2 行が表示されます。

   ```
   Sample AMQSPUT0 start
   target queue is mqtest
   ```

4. これでメッセージを送信できます。各メッセージを入力し、各メッセージの後で Enter キーを押すだけです。以下はその例です。

   ```
   sample message 1
   sample message 2
   ```

5. メッセージの送信が完了したら、Enter キーを 2 回押します。以下のような行が表示されます。

   ```
   Sample AMQSPUT0 end
   ```

6. このテストを完了するには、キューに送信したメッセージを取得します。コマンド行で以下のコマンドを入力します。

   ```
   amqsgetc queue_name queue_manager_name
   ```

   queue_name は使用するキューの名前、queue_manager_name はキュー・マネージャの名前です。以下はその例です。
7. その後、開始行が表示されます。それに続いて、以下のような、以前送信したメッセージが表示されます。

   ```
   Sample AMQSGET0 start
   message <sample message 1>
   message <sample message 2>
   ```

8. このサンプル・プログラムは、その他のメッセージを受信するために短時間待機し、以下を表示します。

```
no more messages
Sample AMQSGET0 end
```

テストに失敗した場合は、IBM ドキュメントを参照してください。問題の考えられる原因には、以下があります。

- セキュリティの問題
- キューが正しく定義されていない
- キュー・マネージャが開始されていない

11

SSH の使用法

%Net.SSH パッケージは、SSH (Secure Shell) 通信のサポートを提供します。このトピックでは、このパッケージに含まれているクラスについて簡単に説明します。

注意

OpenSSL 3.0 では、プロバイダの新しい構想として、アルゴリズム実装のグループをパッケージ化する手法が導入されています。プロバイダの 1 つにレガシー・プロバイダがあります。OpenSSL でも InterSystems IRIS® データ・プラットフォームでも、既定ではレガシー・プロバイダはロードされません。

レガシー・プロバイダのアルゴリズムを使用しないことをお勧めします。このようなすべてのアルゴリズムは、OpenSSL のドキュメントを参照してください。インターシステムズでは、InterSystems IRIS とレガシー・アルゴリズムとの互換性を保証できません。

11.1 SSH セッションの作成

%Net.SSH.Session は SSH セッションを表します。このクラスを使用するには、以下の手順を実行します。

1. クラスのインスタンスを作成します。
2. Connect() インスタンスのメソッドを使用して、サーバに接続します。
3. AuthenticateWithKeyPair()、AuthenticateWithUsername()、または AuthenticateWithKeyboardInteractive() を使用して自身をサーバに対して認証します。詳細は、%Net.SSH.Session のクラス・リファレンスを参照してください。
4. %Net.SSH.Session のその他のメソッドを使用して、リモート・システムとの間での単一ファイルの SCP (Secure Copy) 操作の実行、リモート・コマンドの実行、TCP トラフィックのトンネル、または SFTP 操作の実行を行います。クラス・リファレンスの %Net.SSH.Session を参照してください。

   例えば、OpenSFTP を使用して、SFTP 操作のセッションを使用します。このメソッドは、SFTP 操作に使用できる %Net.SSH.SFTP のインスタンスを参照で返します。次のセクションに示した用例を参照してください。

重要

- これらのクラスを使用できるサポート・プラットフォームの詳細は、%Net.SSH.Session および %Net.SSH.SFTP のクラス・リファレンスを参照してください。
- OpenSSL 3.0 を使用する SSH 接続では、暗号化アルゴリズムとして Blowfish も CAST もサポートしていません。そのような接続を確立しようとすると接続に失敗します。

11.2 OpenSSH と PEM でエンコードされたキー

認証を行うには、公開鍵と秘密鍵のペアと (秘密鍵の) パスフレーズを使用します。公開鍵は OpenSSH 形式でなければならず、秘密鍵は PEM でエンコードする必要があります。ここでは、正しい鍵を持っているかどうかの判断に役立つ、基本情報を提供します。

OpenSSH 形式は、以下の例のようになります (ただし、この形式はここに示す意図的な改行は含みません)。

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCfi2Vq+u0rtt2OC84pyrkq1k7WkrS+s76u3a
+2gdD43KQ2Z3vSUUfksymJjp11JBZEpOtBVIAy221UKdc7j7Qk6sUjZaK8LIy+bzDVwMyFWgVv
Qge7EjdWjrJLBRCDXYML6y1Y25XexThkTWSGyXzGNdr+wfIHYn/mIt0hfvrusauvT/9Wz8K2MG
Aj4BL7UQZpFJrlXzGmewe6++6cZDQQYi0aztwLK798oc9j0LsccdMpqWrjqoU1uANFhYIuUu/T4
7TEhT+e6M+KFYK5TR998eJTO25IjdN2Tgw0feXhQFF/nngbol0bA4auSPaZQsgokKK+E+Q/8UtBd
etEofuV user@hostname
```

PEM でエンコードした秘密鍵には、ファイルの先頭に以下のようなヘッダがあります。

```
--{}{}BEGIN RSA PRIVATE KEY{}{}--
```

ファイルは、同様のフッタで終了します。

```
--{}{}END RSA PRIVATE KEY{}{}--
```

11.3 例 : SFTP 経由のファイルのリスト作成

次のメソッドは、SFTP を経由してサーバ上のファイルのリストを作成する方法を示しています。

Class Member

```
Method SFTPDir(ftpserver, username, password) As %Status
{
    set ssh = ##class(%Net.SSH.Session).%New()
    set status = ssh.Connect(ftpserver)
    set status = ssh.AuthenticateWithUsername(username,password)
    //open an SFTP session and get that returned by reference
    set status = ssh.OpenSFTP(.sftp)
    //get a list of files
    set status = sftp.Dir(".",.files)
    set i=$ORDER(files(""))
    while i'="" {
        write $listget(files(i),1),!
        set i=$ORDER(files(i))
    }
    quit $$$OK
}
```

11.4 例 : キーボード・インタラクティブによる認証

以下のターミナル・セッションは AuthenticateWithKeyboardInteractive() の使用方法を示しています。

Terminal

```
%SYS>set host="192.168.2.100"
%SYS>set lambda="(u,i,p,f,c)  quit $listbuild(c(""password""))"
%SYS>set context("password")="fountain"
%SYS>set sess=##class(%Net.SSH.Session).%New()
%SYS>s status=sess.Connect(host)
%SYS>set status=sess.AuthenticateWithKeyboardInteractive("root",lambda,.context)
```

11.5 例 : リモート・コマンドの実行

以下の例は Execute() の使用方法を示しています。

ObjectScript

```
// %Net.SSH.Session.Execute
// Specify the initial SSH connection information
Set host="192.168.2.37"
Set username = "SSHUser"
Set password = "SSHPassword"
// Set up the initial SSH connection
// localhost <--SSH--> 192.168.2.37
Set sshSession = ##class(%Net.SSH.Session).%New()
Set statusConnection = sshSession.Connect(host)
Set statusConnection = sshSession.AuthenticateWithUsername(username,password)
// Specify the command to execute on the remote host
Set command = "uname -a"
// Execute the command over the SSH connection
// tDevice is a pass-by-reference value to store the raw data passing through the connection
Set statusConnection = sshSession.Execute(command,.tDevice)
```

11.6 例 : ポートの転送

以下の例は ForwardPort() の使用方法を示しています。

ObjectScript

```
// %Net.SSH.Session.ForwardPort
// Specify the initial SSH connection information
Set host="192.168.2.37"
Set username = "SSHUser"
Set password = "SSHPassword"
// Set up the initial SSH connection
// localhost <--SSH--> 192.168.2.37
Set sshSession = ##class(%Net.SSH.Session).%New()
Set statusConnection = sshSession.Connect(host)
Set statusConnection = sshSession.AuthenticateWithUsername(username,password)
// Specify the remote port forward information
Set remotehost = "192.168.2.100"
Set remoteport = 80
// Forward traffic via the SSH connection to a remote host:port
// localhost <--SSH--> 192.168.2.37 --SSHPortFwd--> 192.168.2.100:80
// tDevice is a pass-by-reference value to store the raw data passing through the connection
Set statusConnection = sshSession.ForwardPort(remotehost,remoteport,.tDevice)
```