Delphi/400

業務アプリケーションと メール・チャットサービスの連携

株式会社ミガロ.
システム事業部 2課
前坂　誠二

略　歴

生年月日:1989年3月21日
最終学歴:2011年　関西大学　文学部卒業
入社年月:2011年04月　株式会社ミガロ. 入社
社内経歴:2011年04月　システム事業部配属

現在の仕事内容:

Delphi/400を利用したシステム開発や保守作業を担当。Delphi、Delphi/400の開発経験を積みながら、日々スキルを磨いている。

1.はじめに

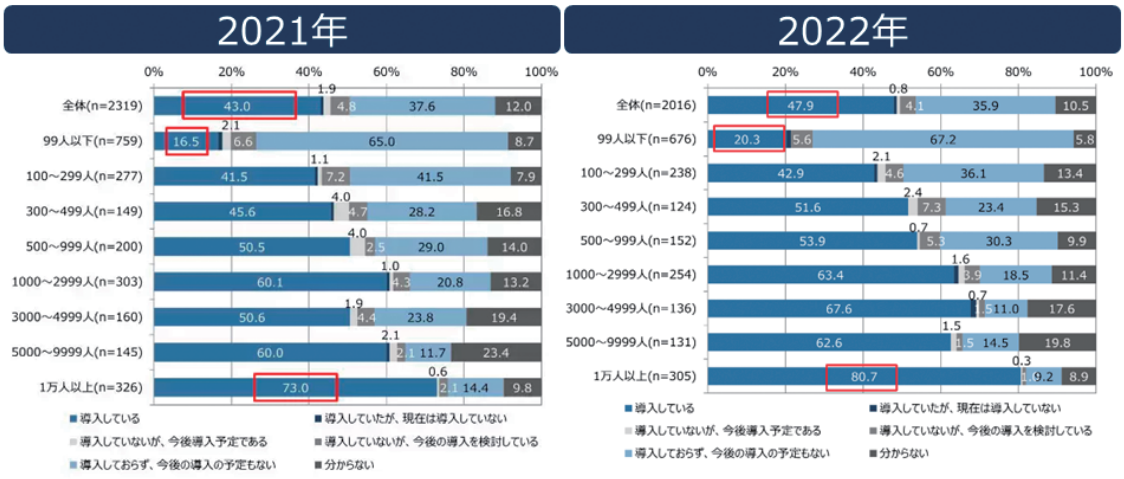
コミュニケーションツールの代表的なものとしてメールがある。使用しているサービスは企業により様々であるが、恐らくほとんどの企業が導入・活用しているであろう。

近年では新たなコミュニケーションツールとしてチャットを導入する企業が増えている。きっかけは、テレワークの実施が大きな要因であったかもしれない。

しかし近年、テレワークは実施する企業が徐々に減少している。一方で企業のチャットサービスの導入率は【図1】のように増加している。つまり、チャットサービスはテレワークの有無に関わらず、コミュニケーションツールのひとつとして認識され、活用されていることが分かる。

そこで本稿では、業務アプリケーションとメール・チャットサービスとの連携方法について紹介する。連携によるメリットは次章にて記載する。

図 1　ビジネスチャット企業導入率



出典：Biz Clip（https://www.bizclip.ntt-west.co.jp/articles/bcl00014-027.html）

2.コミュニケーションツールの特徴

メール・チャットにはそれぞれ異なる特徴がある。チャットはリアルタイムでメッセージの同期が可能である。簡単な連絡やグループでの共有、即座に確認すべきことがある場合に使用が適している。一方でメールは、重要な連絡や長文でメッセージを送る場合、記録として残す必要がある場合に適している。

上記の特徴により、社外への連絡はメールを使用し、社内への連絡はチャットを利用するなど、状況によってコミュニケーションツールを使い分けている企業も多いのではないだろうか。

しかし、日常の業務において使用するツールが増えることは、複数サービス(アプリ)を切り替えて使い分けなければいけないというデメリットも生まれる。

例えば、業務アプリケーションでデータを照会し、照会結果を基にメール内容を入力するシーンでは、業務アプリケーションの起動及びメールサービスの起動が必要となる。さらに社内への連絡が追加で必要となれば、チャットサービスの起動も必要となる。PCの限られた画面スペースの中で複数サービスを立ち上げることは業務効率の低下にも繋がりかねない。

本稿で紹介する業務アプリケーションとの連携を行うと、メールやチャットサービスの立ち上げが不要となる。よって、円滑に業務を進めることが可能となる。

3.Indyの利用

業務アプリケーションと各サービスの連携にはDelphiに標準付属しているIndyを利用する。Indyとは、オープンソースのネットワーク関連コンポーネントである。

本稿の例では業務アプリケーションで見積書を出力し、先方にメール送付するシーンを想定する。また、添付する見積書は圧縮+パスワード付与を行い、メール送信後は社内にチャットで送信完了を報告する業務フローとする【図2】。

Indyを利用し、業務アプリケーションとの連携を実現すると【図3】のようなフローとなる。では、次章より具体的な実装方法について紹介していく。

本サンプルでは、Delphi/400 11 Alexandriaを使用し、メールサービスはOffice365(Outlook)、チャットサービスはChatworkを使用する。

図 2　メール・チャットサービス連携前業務フロー

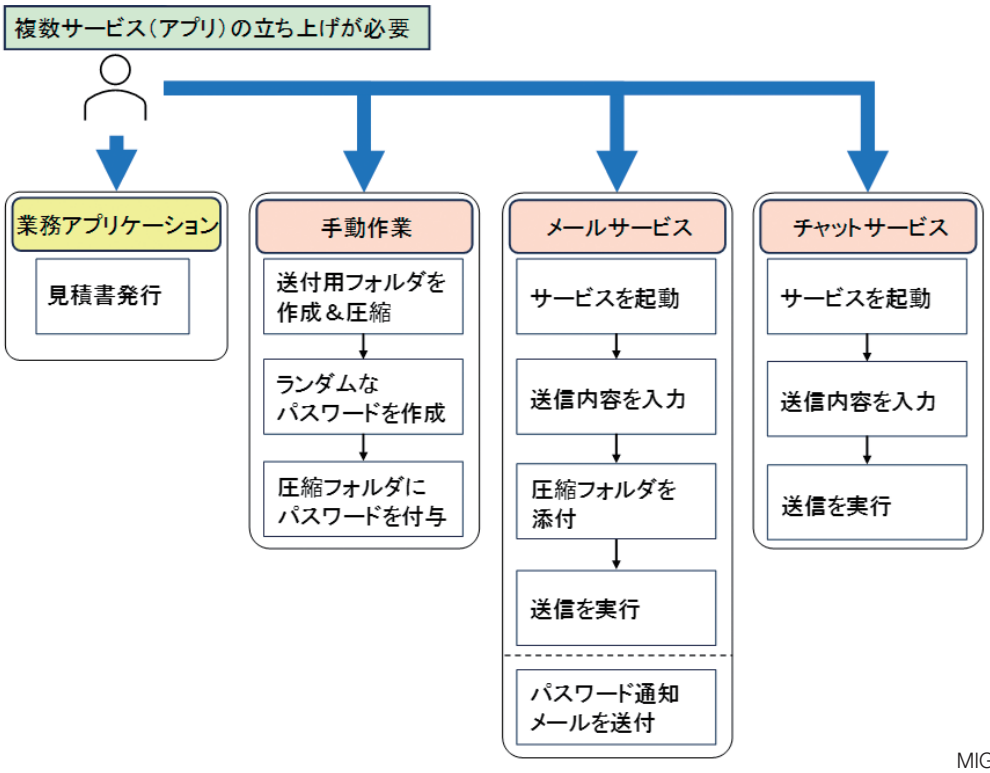
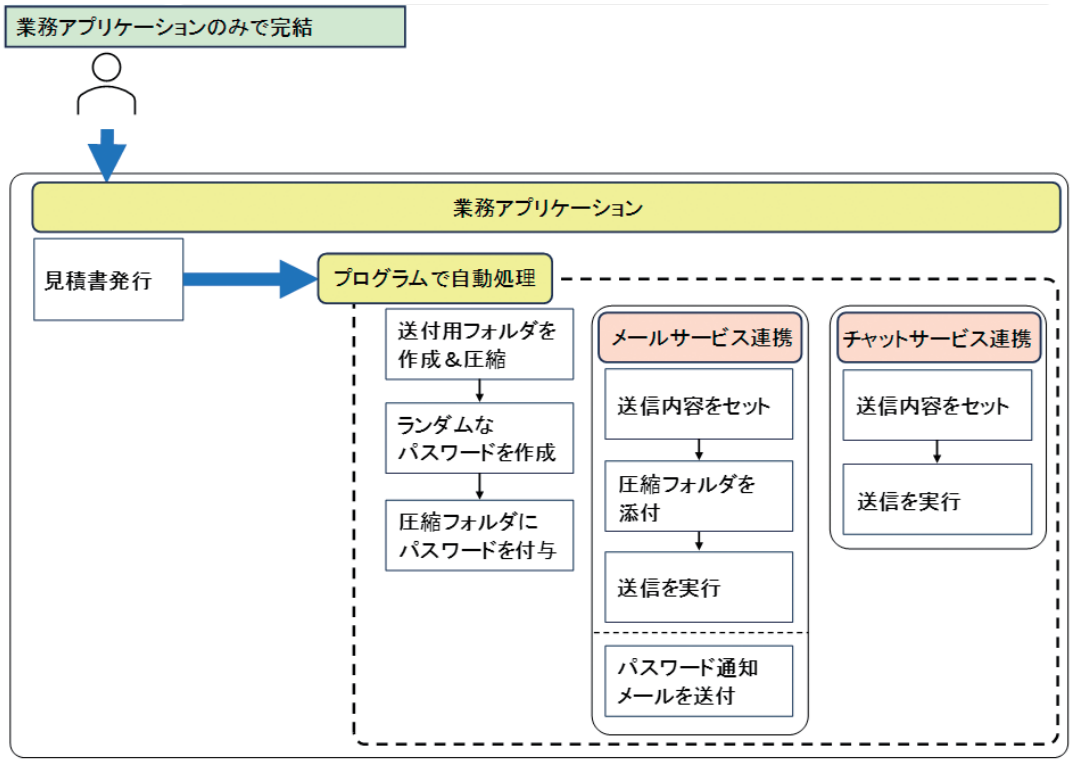


図 3　メール・チャットサービス連携後業務フロー

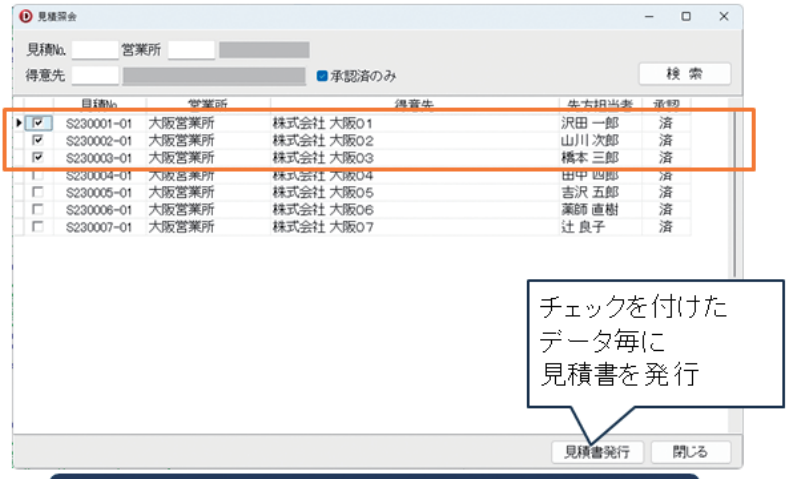


4.メール送信機能の実装

本稿では、業務アプリケーションの仕様を以下と仮定する。

見積照会で作成済みの見積データを検索し、チェックを付けたデータ毎に見積書を発行可能とする。見積照会では、見積書の送付先である先方担当者の名前を表示しており、メールアドレスなどの紐づく担当者情報をマスタにて保管している【図4】。

図 4　業務アプリケーション（改修前）



マスタにメールアドレス等を保管

```
*****************************************************
   FILE-ID    :    MTKTNF
   FUNCTION   :    得意先担当者マスタ
*****************************************************
                              UNIQUE
     R MTKTNR                 TEXT(' 得意先担当者マスタ ')
       TTTKCD        5A       COLHDG(' 得意先コード ')
       TTTNCD        3A       COLHDG(' 得意先担当者コード ')
       TTTNNM      220        COLHDG(' 担当者名 ')
       TTTNML     1000        COLHDG(' メールアドレス ')
       TTTNMF       1A        COLHDG(' メール送信対象 FLG ')
     K TTTKCD
     K TTTNCD
```

4-1.コンポーネントの貼り付け

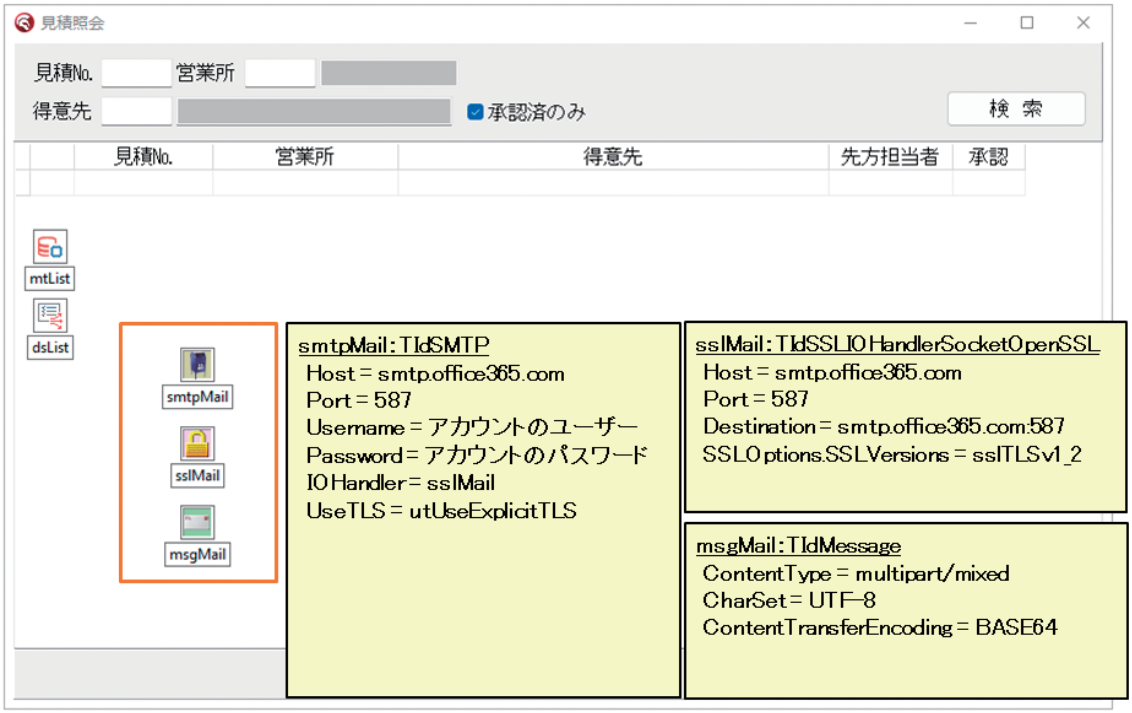
まずは、メール送信処理を実装するために必要なコンポーネントの貼り付けを行う。メール送信機能を実装したい画面に、TIdSMTPとTIdSSLIOHandlerSocketOpenSSL、TIdMessageを配置し、各プロパティ設定を行う【図5】。配置した各コンポーネントの役割は以下の通りである。

smtpMail:TIdSMTP
メールサーバーとの接続及び送信処理の実施

sslMail:TIdSSLIOHandlerSocketOpenSSL
TLS/SSLを使用した暗号化通信を可能にする

msgMail:TIdMessage
送信先アドレスやメッセージ内容の設定

図5 メール送信用コンポーネントの配置



各コンポーネントに対するプロパティ設定について詳しく解説していく。

smtpMailでは、メールサーバーへの接続情報について設定している。本稿ではoffice365を使用するため、Hostは「smtp.office365.com」、Portは「587」にて設定する。設定値については使用するメールサービスによって異なるため、各メールサービスのヘルプなどでご確認いただきたい。また、使用するメールサービスによっては2段階認証の設定をメールサービス側で行い、外部アプリケーションで使用するためのパスワードを生成する必要がある。その際は、生成パスワードをPasswordプロパティに設定しなければならない。

sslMailでは、暗号化通信を行うための設定を行う。本稿ではSSL/TLS(STARTTLS)を使用して通信を行うためsmtpMailと同様にsslMailにHostやPortの設定を行う。SSLVersionsプロパティにはTIdSSLIOHandlerSocketOpenSSLで指定可能な最新バージョンである「sslTLSv1_2」を設定する。

smtpMailのIOHandlerプロパティには「sslMail」を指定し、UseTLSプロパティには「utUseExplicitTLS」を指定する。しかし、UseTLSプロパティは暗号化方式などにより、設定値が異なる為、各メールサービスに合わせて設定いただきたい。

msgMailでは、送信するメッセージの文字タイプや文字コードについて設定している。ContentTypeプロパティには主に以下のような形式を指定できるが、本稿では「multipart/mixed」を指定する。

text/plain：文字テキストのみのメール形式
text/html：htmlと同様に文字に装飾が可能な形式
multipart/mixed：複数のメール形式が混在可能な形式

CharSetプロパティでは文字コードを指定する。Unicode形式に対応するため「UTF-8」を設定する。ContentTransferEncodingプロパティではメールの変換方式について指定できる。本稿ではメールにファイル添付を行う想定であるため、エンコード方式は「BASE64」を設定する。

4-2.メール送信ロジックの実装

コンポーネントの貼り付け完了後、次は送信ロジックの実装を行う。実装は「見積書出力ボタン」押下時のタイミングで行う。**【ソース１】**が送信ロジック実装前の処理である。本稿では、自動送信とするため、件名及び本文を予めAS/400にデータ登録しておく**【図６】**。追加のロジックとしては、**【ソース２】【ソース３】**の通り

①メール内容のデータを取得
②SMTPサーバーへの接続
③メールの送信元と送信先のセット
④メールの件名と本文のセット
⑤送信の実行
⑥SMTPサーバーの接続解除

である。また②の実施後は⑥を必ず実行する必要があるためtry～finally～endにて処理している。また、メール件名の文字化けを防ぐために、msgMailのOnInitializeISOイベントに処理を追加する**【ソース４】**。メール送信のロジックは以上で完了である。しかし、実行すると**【図７】**のようなエラーが発生する。

ソース１

メール送信ロジック実装前

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
begin
  mtList.DisableControls;
  mtList.First;
  try
    while not mtList.Eof do
    begin
      if mtList.FieldByName('CHEK').AsInteger = 1 then
      begin
        // 見積書出力(見積No.のフォルダ)
        OutPutMTMR;
      end;

      mtList.Next;
    end;
  finally
    mtList.EnableControls;
  end;
end;
```

図 6 登録メッセージ

ファイルレイアウト

```
A*       FILE-ID    :    MMESGF
A*       FUNCTION   :    メッセージ登録マスタ
A                                UNIQUE
A         R MMESGR               TEXT(' メッセージ登録マスタ ')
A           MSSKEY       10A     COLHDG(' メッセージ識別キー ')
A           MSMSRW        3S 0   COLHDG(' メッセージ行 ')
A           MSMESG       100     COLHDG(' メッセージ ')
A         K MSSKEY
A         K MSMSRW
```

メール件名

	メッセージ識別キー	メッセージ行	メッセージ
000001	MTMRMAILK	1	御見積書の送付

メール本文

データ毎にセットする値を変更するものは仮の値とし、ロジックにて変換する

000001	MTMRMAILH	1	[CANPANY]
000002	MTMRMAILH	2	[TANTNM] 様
000003	MTMRMAILH	3	いつもお世話になっております。株式会社ミガロ．の [MYNAME] です。
000004	MTMRMAILH	4	
000005	MTMRMAILH	5	この度は見積もりのご依頼をいただき、誠にありがとうございます。
000006	MTMRMAILH	6	御見積書を添付ファイルにてお送りします。
000007	MTMRMAILH	7	内容にご不明な点、ご要望がございましたらお申し付けください。
000008	MTMRMAILH	8	
000009	MTMRMAILH	9	═══════════
000010	MTMRMAILH	10	株式会社ミガロ．
000011	MTMRMAILH	11	〒 556-0017
000012	MTMRMAILH	12	大阪府大阪市浪速区湊町 2-1-57
000013	MTMRMAILH	13	難波サンケイビル 13F
000014	MTMRMAILH	14	TEL:06-6631-8601　FAX:06-6631-8603
000015	MTMRMAILH	15	═══════════

ソース 2

メール送信処理の実装(メイン処理)

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
var
  sSubject, sGetBody, sBody: String;
  sConpany: String;
  sTANTNM: String;
  sMYNAME: String;
begin
  // メール内容取得
  sSubject := GetMSG('MTMRMAILK'); // 件名
  sGetBody := GetMSG('MTMRMAILH'); // 本文

  mtList.DisableControls;
  mtList.First;
  try
    // SMTPサーバーへの接続
    smtpMail.Connect;
    try
      while not mtList.Eof do
      begin
        if mtList.FieldByName('CHEK').AsInteger = 1 then
        begin
          // 見積書出力(見積No.のフォルダ)
          OutPutMTMR;
```

① GetMSG関数はソース3で記述

②

```
          try
            // メールFrom - To
            msgMail.From.Address := 'mgrtechnical@outlook.jp'; // メールFrom
            msgMail.Recipients.Clear;                          // クリア
            msgMail.Recipients.EMailAddresses :=
              mtList.FieldByName('TTTNML').AsString;           // メールTo

            msgMail.Subject := sSubject; // 件名セット

            // メール本文
            // 動的な値をStringReplaceで変換
            sConpany  := mtList.FieldByName('TKNM').AsString; // 得意先名
            sTANTNM   := mtList.FieldByName('STNM').AsString; // 先方担当者
            sMYNAME   := '前坂'; // 送信者名

            msgMail.Body.Clear;
            sBody := StringReplace(sGetBody, '[CANPANY]', sConpany, [rfReplaceAll]);
            sBody := StringReplace(sBody, '[TANTNM]', sTANTNM, [rfReplaceAll]);
            sBody := StringReplace(sBody, '[MYNAME]', sMYNAME, [rfReplaceAll]);
            msgMail.Body.Text := sBody; // 本文セット

            // 送信処理
            smtpMail.Send(msgMail);
          except
            on E: Exception do
            begin
              raise;
            end;
          end;
        end;

        mtList.Next;
      end;
    finally
      // SMTPサーバー接続解除
      smtpMail.Disconnect;
    end;
  finally
    mtList.EnableControls;
  end;
end;
```

③ ④ ⑤ ⑥

ソース 3

メール送信処理の実装(メール内容の取得)

```
{*******************************************************************************
 目的: メッセージマスタ取得処理
 引数:
 戻値:
*******************************************************************************}
function TfrmReview.GetMSG(AGetKEY: String): String;
begin
  qryMSG.Close;
  qryMSG.SQL.Text :=
    ' SELECT MSMESG FROM MMESGF WHERE MSSKEY = :MSSKEY ORDER BY MSMSRW ';
  qryMSG.ParamByName('MSSKEY').AsString := AGetKEY;
  qryMSG.Open;
  try
    while not qryMSG.Eof do
    begin
```

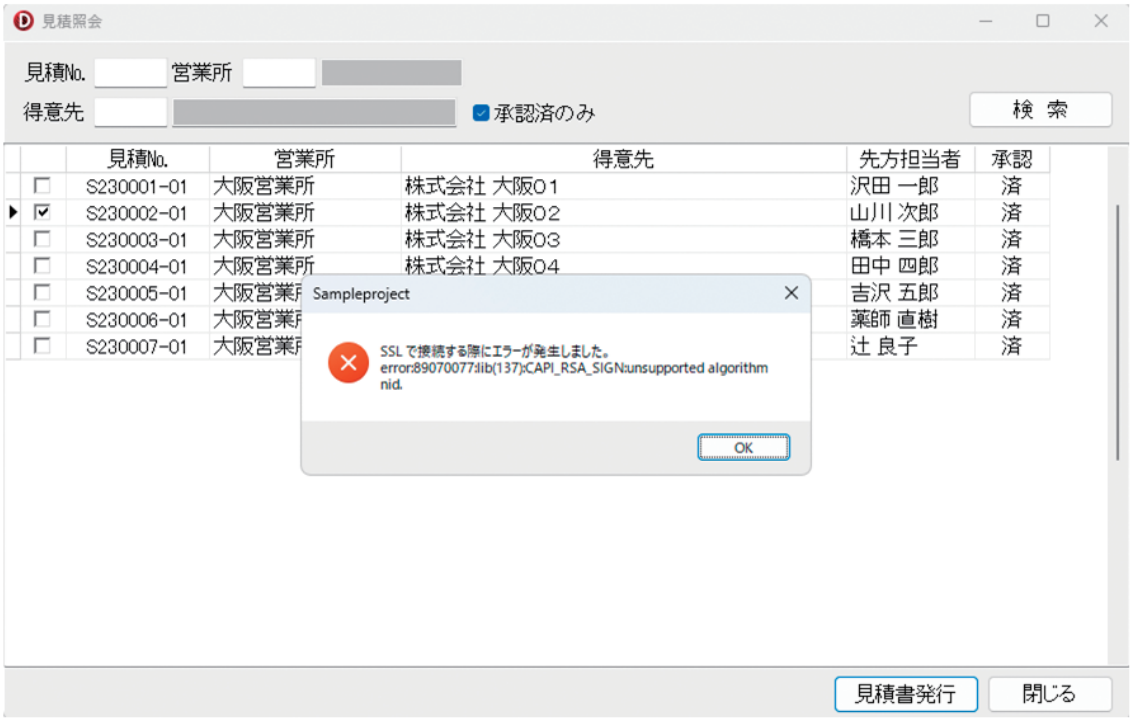
```
        if Result <> '' then
        begin
          // 改行
          Result := Result + #13#10;
        end;
        Result := Result + qryMSG.FieldByName('MSMESG').AsString;
        qryMSG.Next;
      end;
    finally
      qryMSG.Close;
    end;
  end;
```

ソース 4

メール件名の文字化け防止

```
{*******************************************************************************
 目的: TIdMessage Initialize処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.msgMailInitializeISO(var VHeaderEncoding: Char;
  var VCharSet: string);
begin
  VHeaderEncoding := 'B';
  // メールヘッダーのCharSetはここで設定する(メール件名に影響)
  VCharSet := 'UTF-8';
end;
```

図 7 メール送信エラー



【図7】のエラーを解決するためには https://indy.fulgan.com/SSL/ より「libeay32.dll」「ssleay32.dll」を取得し、実行モジュールと同階層に配置する必要がある【図8】。各dllを配置した後、再度モジュールを実行するとメール送信がエラーなく実行される【図9】。

図8 dllのインストール

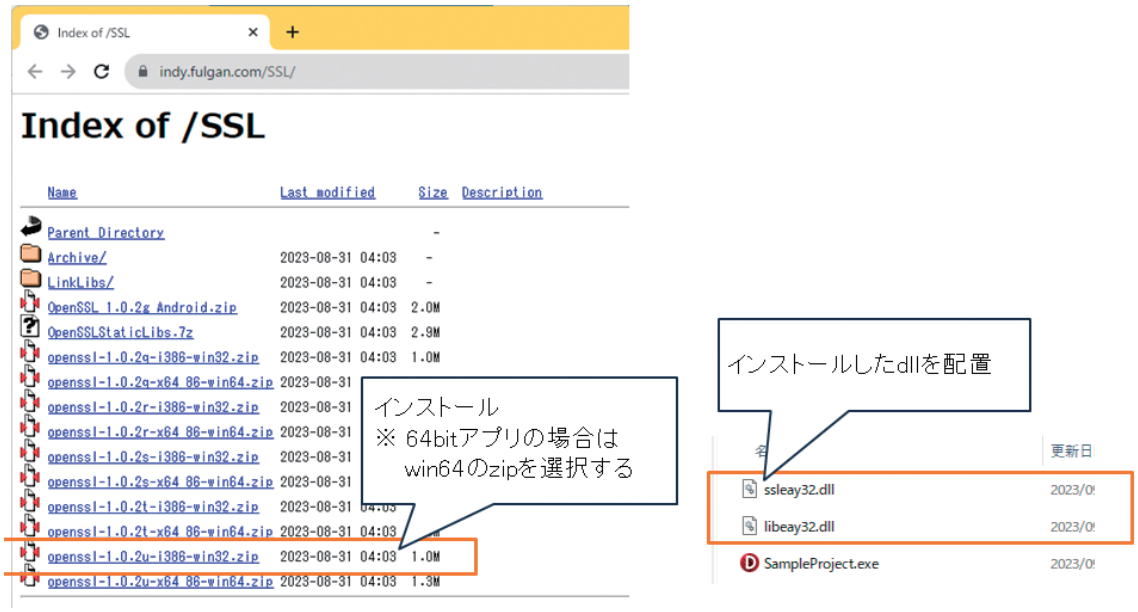
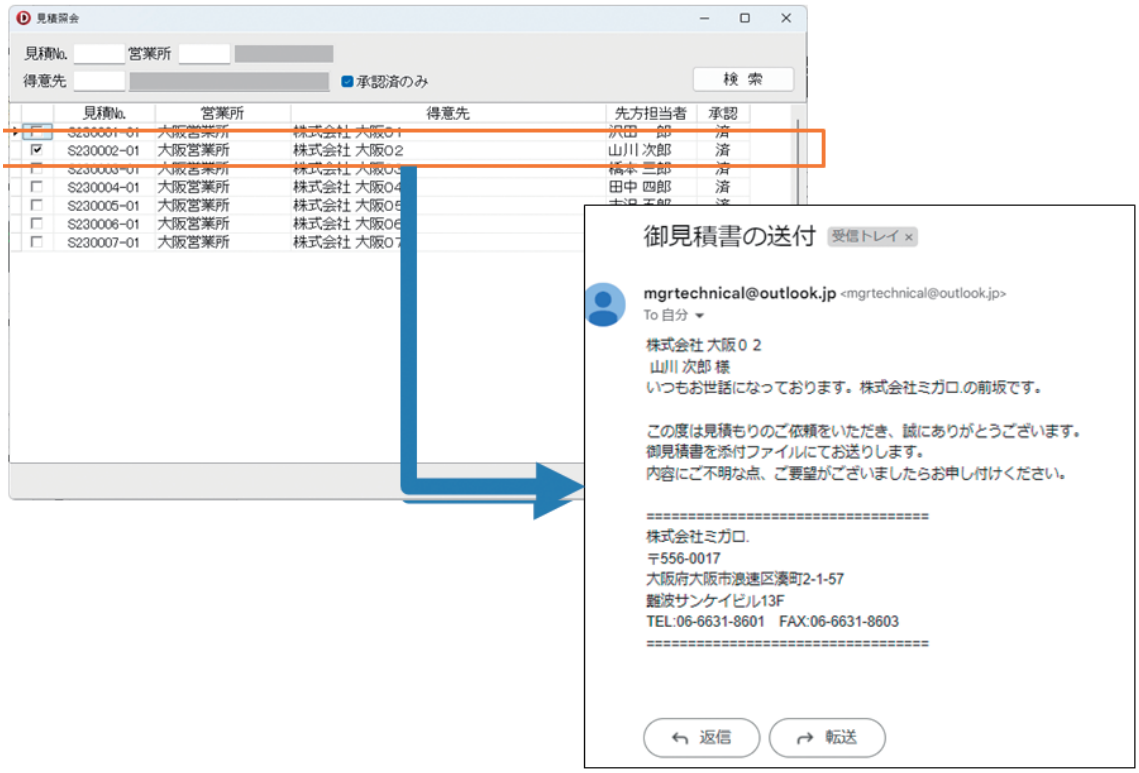


図9 メール送信実行結果



4-3.出力フォルダの圧縮とパスワード付与

本稿では、出力した見積書は特定のフォルダに保管される仕組みとする。メールには該当フォルダを圧縮後、パスワードを付与したものを添付する。

フォルダの圧縮には「TAbZipper」コンポーネントを使用する。「TAbZipper」は「GetItパッケージマネージャー」から「Abbrevia　2021.11」を選択しインストールする。インストール方法は**【図 10】**のとおりである。インストールが完了したら、画面に「TAbZipper」コンポーネントを貼り付けてみよう**【図 11】**。

図 10　コンポーネントのインストール

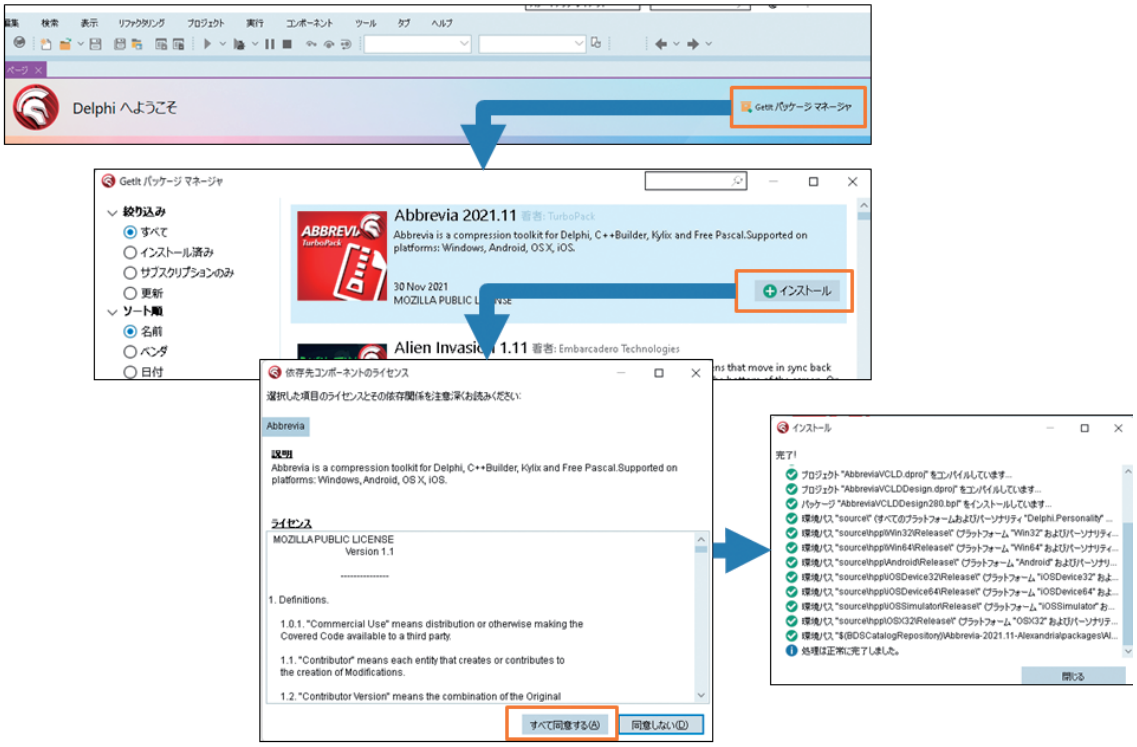
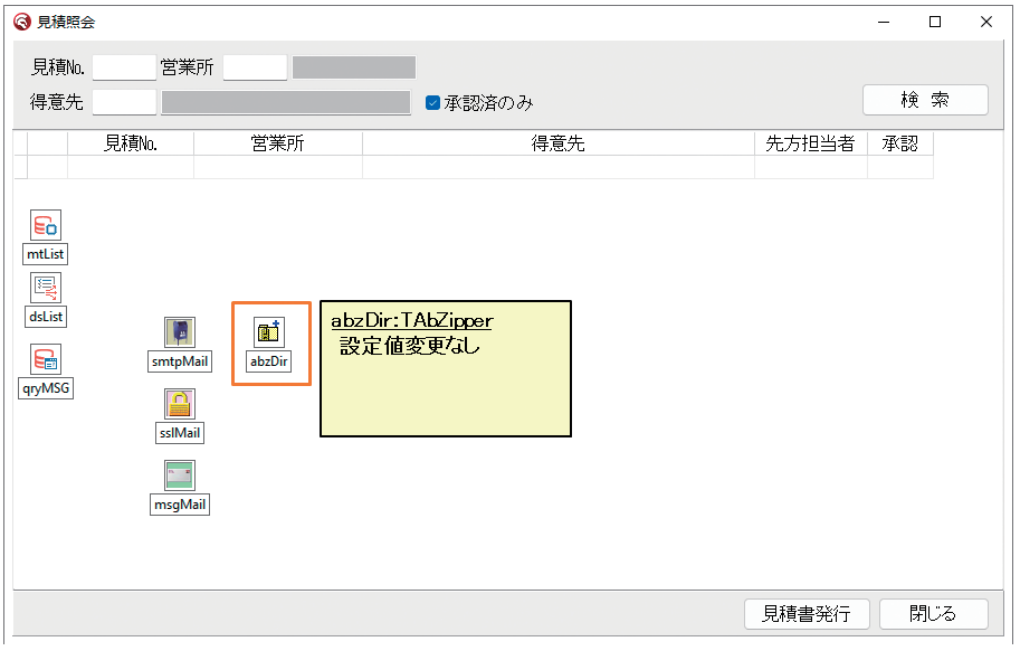


図 11　ファイル圧縮用コンポーネントの貼り付け



コンポーネントの貼り付けが完了したら、まずはランダムパスワードを作成する関数を実装する。ランダムな文字列を設定するにはRandom関数を使用する。Random関数の呼び出し前には、Randomize手続きにより乱数生成関数を初期化しておく【ソース5】。ランダムパスワードの実装が完了したら、次は圧縮化の処理を実装する。圧縮化の処理はFileNameに圧縮後のファイル名（フルパス）を指定し、BaseDirectoryに圧縮対象のフォルダ（フルパス）を指定する。その後、AddFilesにて圧縮化するファイルを指定し、SaveにてFileNameで指定した場所に保管される。本稿のようにPassWord設定を行いたい場合は、PassWordに文字列をセットする【ソース6】。この時に併せて、設定パスワードとZipファイル名を戻り値にしておく（メールへの添付処理で使用）。

ソース 5

ランダムパスワード作成処理

```
{*************************************************************************
 目的: ランダムパスワード作成処理
 引数:
 戻値: 生成パスワード
*************************************************************************}
function TfrmReview.GetRandomPass: String;
var
  sRandPassStr: String;
begin
  Randomize;
  sRandPassStr := 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890';
  repeat
    Result := Result + sRandPassStr[Random(Length(sRandPassStr)) + 1];
  until (Length(Result) = 10);
end;
```

ソース 6

フォルダ圧縮処理

```
{*************************************************************************
 目的: フォルダ圧縮処理
 引数: AZipPassWord - 設定パスワード　戻り値
 戻値: 圧縮ファイル名（フルパス）
*************************************************************************}
function TfrmReview.ZipFileCreate(var AZipPassWord: String): String;
var
  LFileName: TFileName;
  sZipFileName: String;
begin
  abzDir.FileName:= 'C:¥Document¥見積書¥'+ mtList.FieldByName('MTNO').AsString+ '.zip';
  abzDir.BaseDirectory:= 'C:¥Document¥見積書¥'+ mtList.FieldByName('MTNO').AsString;
```

```
  // abzDir.BaseDirectory直下のファイルを全て圧縮
  for LFileName in TDirectory.GetFiles(
    abzDir.BaseDirectory, '*.*', TSearchOption.soAllDirectories) do
  begin
    sZipFileName :=
      StringReplace(LFileName,
        IncludeTrailingPathDelimiter(abzDir.BaseDirectory), '', [rfIgnoreCase]);
    abzDir.AddFiles(sZipFileName, 0);
  end;

  abzDir.Password := GetRandomPass; // ランダムパスワード設定
  AZipPassWord := abzDir.Password;
  abzDir.Save;

  Result := abzDir.FileName;
  abzDir.CloseArchive;
end;
```

4-4.圧縮ファイルの添付

圧縮ファイル作成の実装が完了したら、メールへの添付処理を実装する。メールへの添付はUses節にIdAttachmentFileを追加し、TIdAttachmentFileを生成する。後は、圧縮ファイル名及び各プロパティを設定するだけで実装完了である【ソース7①】。また、パスワードメールも送付するよう併せて実装する【ソース7②】。ファイル添付処理を実装した結果が【図12】である。

ソース7

ファイル添付処理

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
var
  sSubject, sGetBody, sBody: String;
  sConpany: String;
  sTANTNM: String;
  sMYNAME: String;
  sZipFileName, sZipPassWord: String;
begin
  // メール内容取得
  sSubject := GetMSG('MTMRMAILK'); // 件名
  sGetBody := GetMSG('MTMRMAILH'); // 本文

  mtList.DisableControls;
  mtList.First;
```

```
  try
    // SMTPサーバーへの接続
    smtpMail.Connect;
    try
      while not mtList.Eof do
      begin
        if mtList.FieldByName('CHEK').AsInteger = 1 then
        begin
          // 見積書出力(見積No.＋システム日時のフォルダ)
          OutPutMTMR;

          // Zip化 (Zipファイル名を返却)
          sZipFileName := ZipFileCreate(sZipPassWord);
          try
```

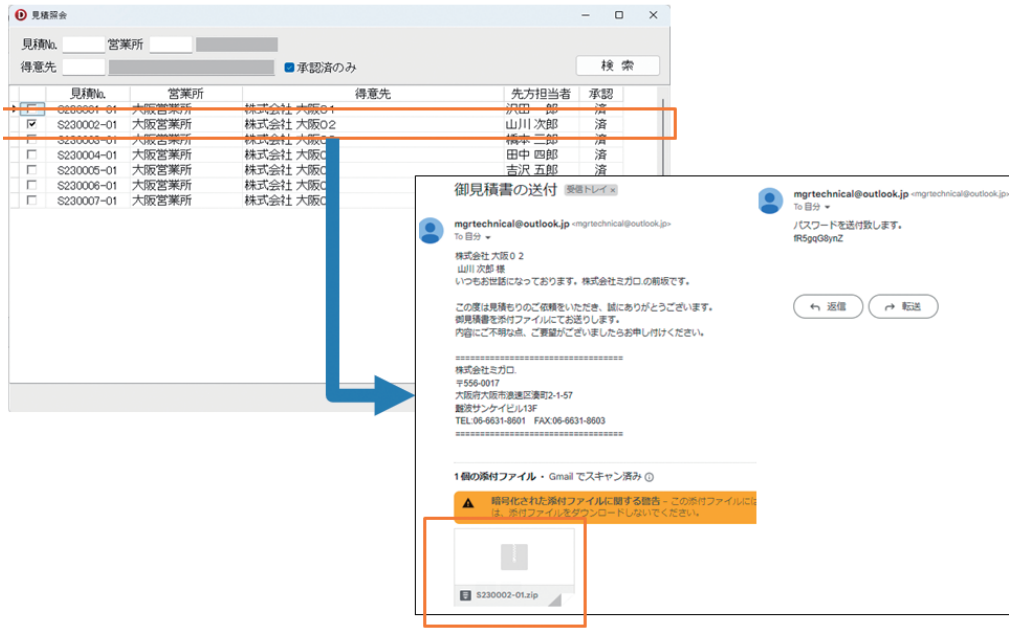
～ メール本文設定処理 省略 ～

```
            // 添付ファイルの設定
            msgMail.MessageParts.Clear;
            with TIdAttachmentFile.Create(msgMail.MessageParts, sZipFileName) do
            begin
              FileName := sZipFileName;
              ContentType := 'application/octet-stream';
              ContentTransfer := 'base64';
            end;

            // メール本文送信処理
            smtpMail.Send(msgMail);

            // パスワードメールの送付
            msgMail.Subject := sSubject; // 件名セット
            msgMail.Body.Clear;
            msgMail.Body.Text := 'パスワードを送付致します。' + #13#10 + sZipPassWord; // 本文セット
            msgMail.MessageParts.Clear;  // 添付ファイルクリア

            // パスワードメール送信処理
            smtpMail.Send(msgMail);
          except
            on E: Exception do
            begin
              raise;
            end;
          end;
        end;

        mtList.Next
      end;
    finally
      // SMTPサーバー接続解除
      smtpMail.Disconnect;
    end;
  finally
    mtList.EnableControls;
end;
end;
```

①

②

図 12　ファイル添付処理実行結果



5.チャット送信機能の実装

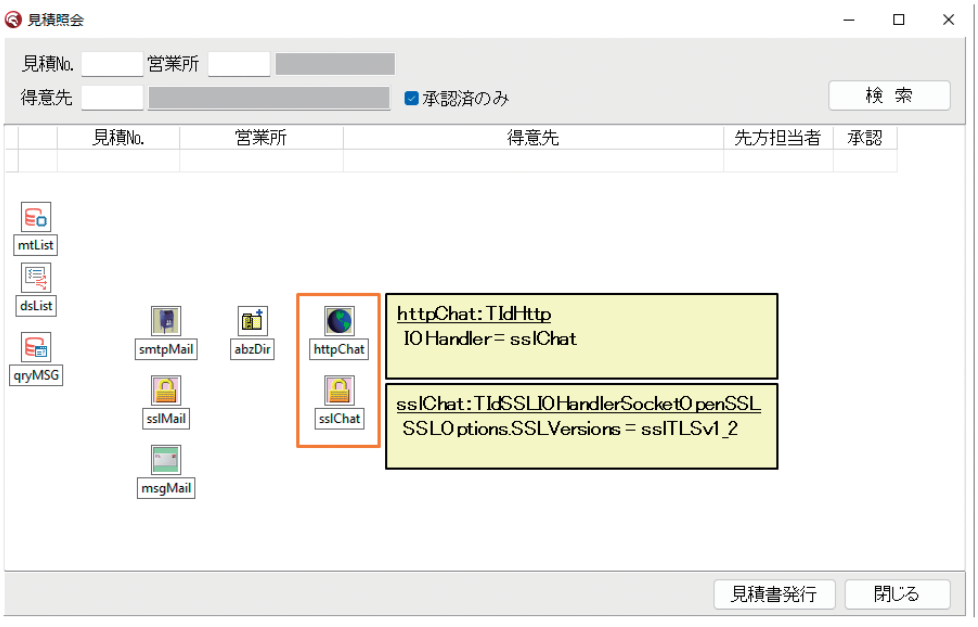
本章では、チャット送信機能の実装方法について紹介していく。

5-1.コンポーネントの貼り付け

まずは、チャット送信処理を実装するために必要なコンポーネントの貼り付けを行う。TIdHTTPとTIdSSLIOHandlerSocketOpenSSLを配置し、各プロパティ設定を行う【図 13】。配置した各コンポーネントの役割は以下の通りである。

httpChat:TIdHTTP
Postメソッドを使用し、指定URLにデータ送信を実施
sslChat:TIdSSLIOHandlerSocketOpenSSL
TLS/SSLを使用した暗号化通信を可能にする

図 13　チャット送信用コンポーネントの配置



5-2.APIエンドポイントの確認

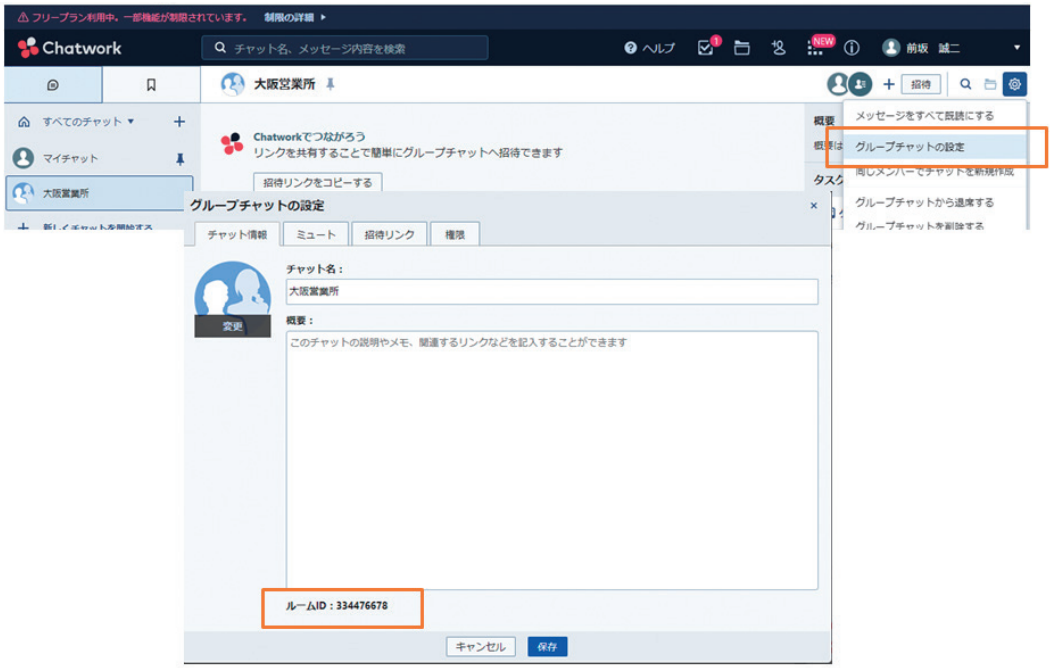
APIエンドポイントとは、Webサービスやアプリケーションに外部クライアントが通信するためのURLである。つまり、本稿で使用するChatworkにおいても、業務アプリケーションが通信するためのURLを確認しなければならない。Chatworkでは、https://developer.chatwork.com/docs/endpointsのページにて、設定するURLやパラメータに関する記載がある。また、チャット送信を行うためには併せてトークン、チャットのルームIDを確認する必要がある。

トークンとはAPIの認証を提供するセキュリティの仕組みであり、APIを使用するためのパスワードのようなものである。トークンの取得はサービスによって異なるが、Chatworkの場合は【図 14】のように、サービス連携のリンクから確認することで取得可能である。

ルームIDはチャットを行うグループの識別IDであり、ルームIDの指定により、指定グループへのチャット送信が可能となる、ルームIDの確認方法は【図 15】である。

図 14　トークンの確認方法

図 15　ルームIDの確認方法



5-3.チャット送信ロジックの実装

チャット送信ロジックの実装は先ほどのAPIエンドポイントを確認したページを参考にする。リクエストに必要なContentTypeプロパティとCustomHeadersプロパティに値を設定する。後は、送信先URLと送信するメッセージ内容を引数にPostメソッドを実行すると、実装は完了である**【ソース8】【ソース9】**。実行結果は**【図16】**となる。

ソース 8

チャット送信処理(呼び出し元)

```
{*******************************************************************************
 目的: 見積書発行ボタン押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TfrmReview.btnMTROutputClick(Sender: TObject);
var
  sSubject, sGetBody, sBody: String;
  sConpany: String;
  sTANTNM: String;
  sMYNAME: String;
  sZipFileName, sZipPassWord: String;
  sSendMTNO: String;
begin
  // メール内容取得
  sSubject := GetMSG('MTMRMAILK'); // 件名
  sGetBody := GetMSG('MTMRMAILH'); // 本文

  mtList.DisableControls;
  mtList.First;
  try
    // SMTPサーバーへの接続
    smtpMail.Connect;
    try
      while not mtList.Eof do
      begin
        if mtList.FieldByName('CHEK').AsInteger = 1 then
        begin
          // 見積書出力(見積No.+システム日時のフォルダ)
          OutPutMTMR;

          ～ メール送信処理 省略 ～

          // 送付見積No.
          if sSendMTNO <> '' then
          begin
            sSendMTNO := sSendMTNO + #13#10;
          end;
          sSendMTNO := sSendMTNO + mtList.FieldByName('MTNO').AsString

        end;

        mtList.Next;
      end;
    finally
      // SMTPサーバー接続解除
      smtpMail.Disconnect;
    end;
  finally
    mtList.EnableControls;
  end;

  // チャット送信処理
  SendChat(sSendMTNO);
end;
```

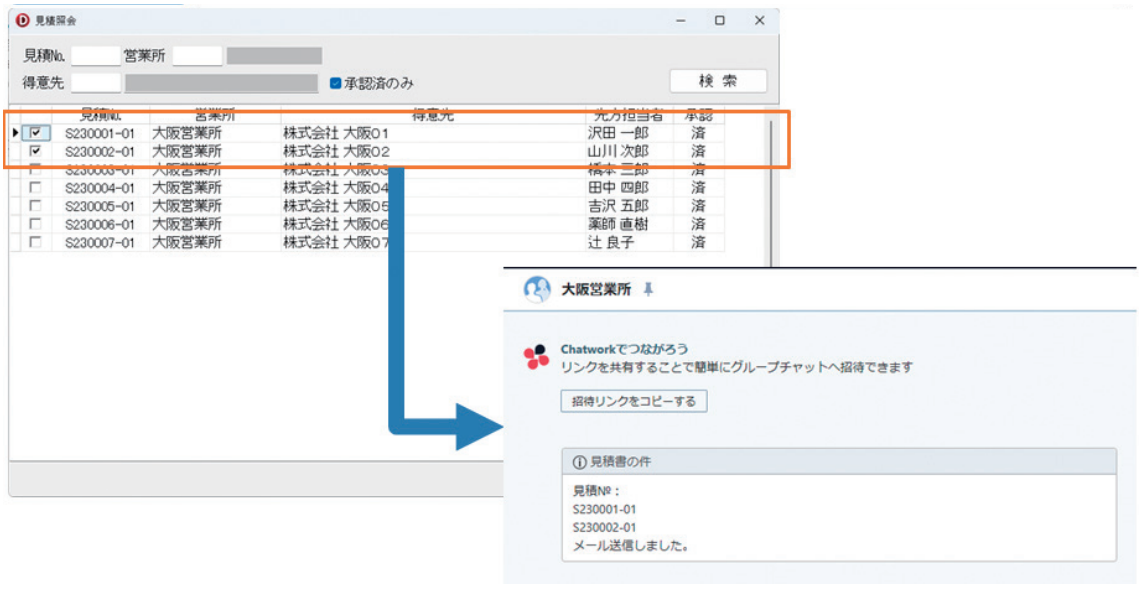
SendChat処理は
ソース9で記述

ソース 9

チャット送信処理(メイン処理)

```
{*******************************************************************************
 目的: チャット送信処理
 引数: AMTNO : 送付見積No. ※複数可能性あり
 戻値:
*******************************************************************************}
procedure TfrmReview.SendChat(AMTNO: String);
var
  slBody: TStringList;
  sSetInfo: String;
  sURL, sTOKN, sRMID: String;
begin

  sURL  := 'https://api.chatwork.com/v2/rooms/';  // URL
  sTOKN := '7fe7a578371e92cdaa609e34b0ae43a2';     // トークン
  sRMID := '334476678';                           // ルームID

  slBody := TStringList.Create;
  try
    sSetInfo := 'body=[info]';
    sSetInfo := sSetInfo + '[title]見積書の件[/title]';

    sSetInfo := sSetInfo + '見積No.:' + #13#10 + AMTNO + #13#10 + 'メール送信しました。';
    sSetInfo := sSetInfo + '[/info]';
    slBody.Add(sSetInfo);

    httpChat.Request.ContentType := 'application/x-www-form-urlencoded';
    httpChat.Request.CustomHeaders.AddValue('X-ChatWorkToken', sTOKN);
    try
      httpChat.Post(sURL + sRMID + '/messages', slBody);
    except
      raise;
    end;
  finally
    FreeAndNil(slBody);
  end;

end;
```

図 16 チャット送信処理実行結果



6.おわりに

各サービスのメッセージ送信画面を立ち上げることなく、業務アプリケーション内で処理が完結することをおわかりいただけただろうか。また、各サービスとの連携方法についても、Indyを利用すると簡単に実装が可能であることを実感いただけたと思う。

本稿は、送信の自動化による例でご紹介したが、もちろん自身でメッセージ送信用の入力画面を作成いただくことも可能である。また、コンポーネントを変更すれば受信機能についても容易に実装可能である。

現在は、メールサービス、チャットサービスの種類が豊富で、どれを導入すべきか判断が難しい。そこで、現在使用している業務アプリケーションと連携可能かといった観点をひとつの判断基準としてもよいのかもしれない。

本稿でご紹介した実装方法であれば、プロパティの設定値を変更するだけで他のサービスへの連携も可能である。もし、現在ご検討されているメールサービスやチャットサービスがあれば、本稿を参考に連携可能かをお試しいただけると幸いである。