Magic uniPaaS V1Plus
トラブルシューティング ツール

初版　2010 年 11 月 26 日

第 2 版　2011 年 2 月 8 日

マジックソフトウェア・ジャパン株式会社

目次

第1章　はじめに

Magic uniPaaS のアプリケーションが思った通りに実行しない場合に、原因を特定するために各種のトラブルシューティング用のツールを活用すると、効率よく調査することができます。

本書では、トラブルシューティングに役立つツールおよびプログラムの小技について説明します。

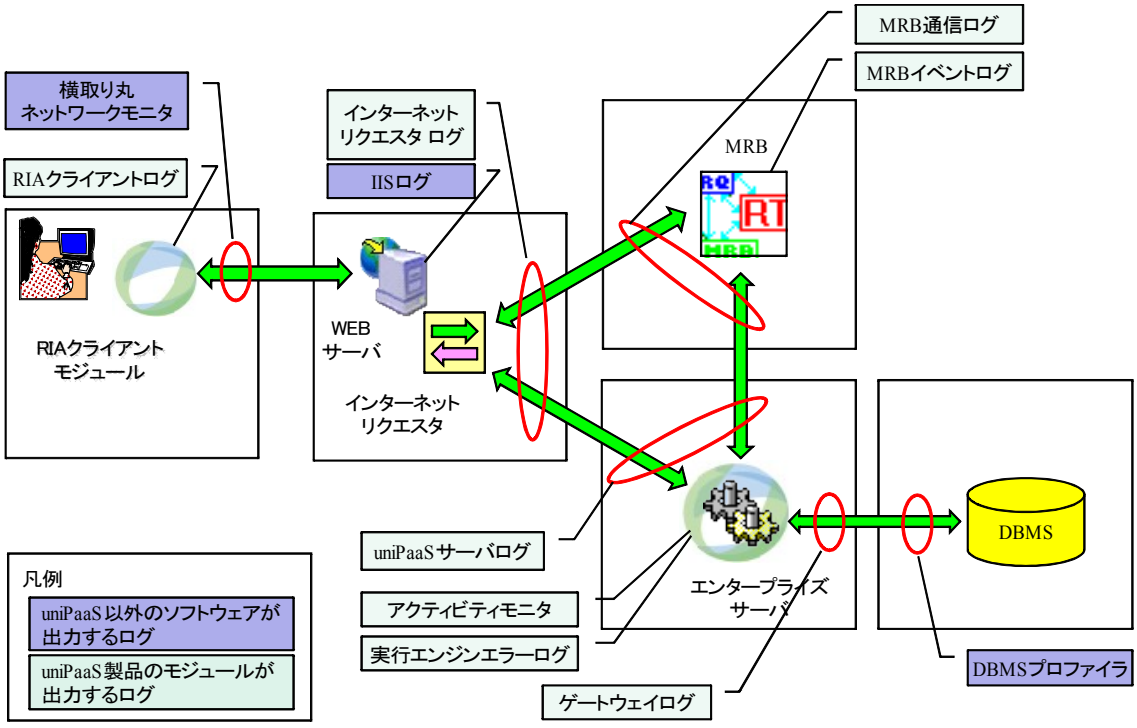
本書の読者は、uniPaaS の基本についてすでに十分に理解していることを前提としています。また、Windows、DBMS、ネットワーク等について基本的な技術知識があることを前提としています。

第2章　uniPaaS 実行エンジンの出力ログ

本章では、uniPaaS 実行エンジンが出力するログについて説明します。

種類	出力内容	設定	GUI に表示	ファイルに出力
アクティビティモニタ	uniPaaS 実行エンジンの実行をトレース	MAGIC.INI（ロギング設定）	○	○
ゲートウェイログ	uniPaaS 実行エンジンが DBMS に発行する SQL 文などをトレース	MAGIC.INI（ロギング設定）	○	○
実行エンジンエラーログ	uniPaaS 実行エンジンのエラーを記録	（自動）		○
RIA クライアントログ	RIA クライアントモジュールの実行状況をトレース	MAGIC.INI (Studio)/ .publish.html		○
サーバ通信ログ	uniPaaS サーバと、MRB/リクエスタとの通信状況をトレース	MGREQ.INI		○
MRB 通信ログ	MRB と他モジュールとの通信状況をトレース	MGRB.INI		○
MRB イベントログ	MRB の主要イベントを記録	（自動）		○
インターネットリクエスタログ	インターネットリクエスタの TCP/IP レベルの通信状況を記録	MGREQ.INI		○

全体像を、RichClient Server システムについて図示したものが、下図です。ここには、uniPaaS 製品モジュール以外のソフトウェアが出力するログも参考のために記載されています。これらのログについては第 8 章「DBMS のユーティリティ」および第 9 章「uniPaaS 製品以外のツール」で説明します。

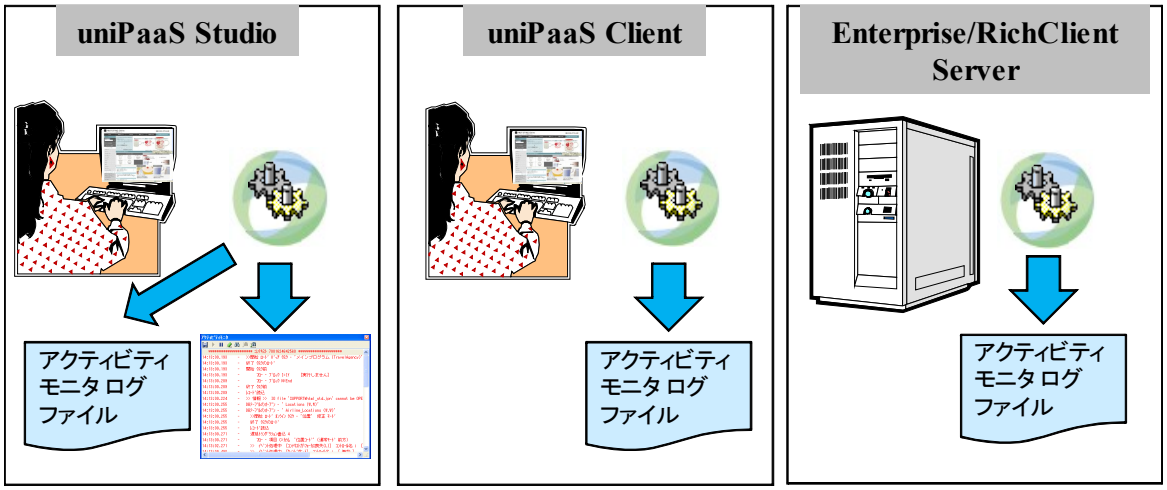


2.1. アクティビティ モニタとは何ですか？

アクティビティモニタは、uniPaaS 実行エンジンのデバッグ機能であり、Magic アプリケーションの実行の様子をトレースします。これには、Magic のコマンド(セレクト、コール、項目更新等々)の実行が 1 ステップごとにトレースされるので、Magic アプリケーションが意図したように動作しない場合に、プログラムの実行の流れをチェックすることができます。

アクティビティモニタの出力は、次のいずれかの方法で記録されます。

方法	説明	参照
アクティビティ モニタ	Studio で開発時にデバッグしている場合にだけ利用可能。Studio の「アクティビティモニタ」画面を開いて表示させます。	2.2 Studio でアクティビティ モニタ画面を開くには？
ログファイルに出力	すべての uniPaaS 製品で利用可能。アクティビティモニタの内容を指定したファイルに書き出します。	2.6 アクティビティ モニタをファイル出力させるには？
リモート デバッガ	リモート デバッガ上の「アクティビティ モニタ」画面に表示します。すべての uniPaaS 製品で利用可能です。	第 7 章「リモートデバッガ」で説明します。



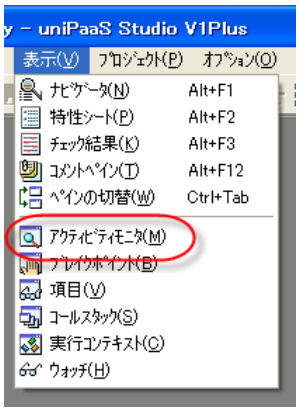
Studio では、モニタ画面、ログファイルの両方に出力できる。

Client および Enterprise Server/RichClient Server では、ログファイルだけに出力できる。

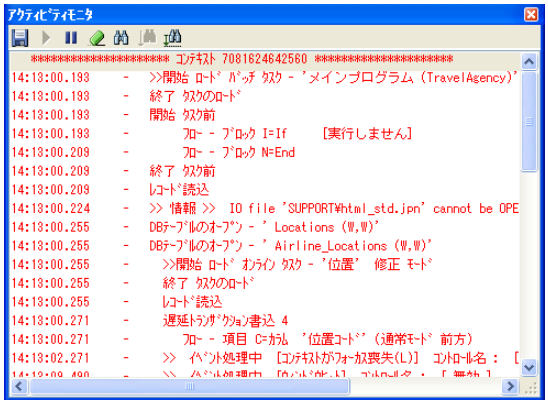
2.2. Studio でアクティビティ モニタ画面を開くには？

開発の段階で、Studio 製品を使って単体テストを行っているときには、Studio の「アクティビティ モニタ」画面を表示させることにより、プログラムの実行の流れを見ることができます。

1. プロジェクトを開いた状態で、メニュー
   「**表示(V) → アクティビティモニタ(M)**」を選択します。



2. 右図のような画面が表示されます。
   初期状態では内容は空白です。
   また、Studio の画面の状態によっては、Studio のウィンドウにドッキングした状態で表示されることもあります。

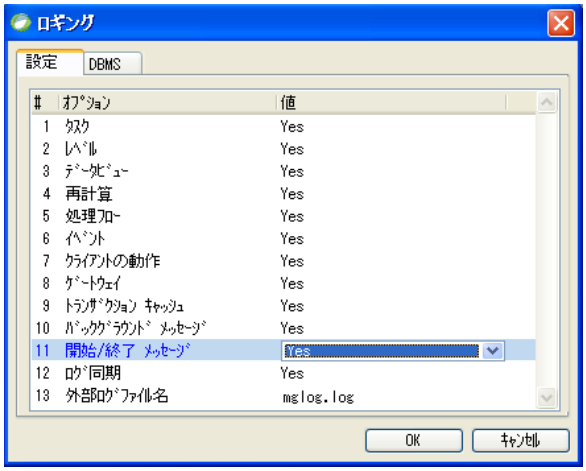


アクティビティモニタにはツールバー(右図)があり、次ページの表のような機能を持っています。

これらの機能は、アクティビティモニタのコンテキストメニュー(右図)からも実行できます。

アイコン	コンテキストメニュー	機能
	保存(S)	ログの内容を、テキストファイルに書き出します。
	出力開始(O)	ログ出力を開始します。
	出力停止(U)	ログ出力を中断します。
	クリア(C)	ログ内容を消去します。
	検索(F)	ログの内容から文字列検索を行います。
	再検索(N)	次の文字列検索を続けます。
	開始/終了チェック(M)	対応する **開始** あるいは **終了** に移動します。

最後の **開始/終了チェック(M)** は、ロギングダイアログ（2.5「ロギングダイアログでフィルタを設定するには？」参照）で、**開始/終了メッセージ** が **Yes** になっている場合にだけ有効です。
例えば、アクティビティモニタ上で、**レコード後開始** の行にカーソルがあった場合、**開始/終了チェック(M)** ボタンを押すと、対応する **レコード後終了** の行にカーソルが移動します。



2.3. アクティビティモニタの「フィルタ」とは何ですか？

アクティビティ モニタの出力は、一番詳細に設定すると、各コマンドの実行、再計算などまで記録されるので、簡単なプログラムの実行でも膨大な量になることがあります。あまり細かすぎると、重要な部分を見つけ出すために却って不便なことがあるので、必要な情報だけに取捨選択したいことがあります。このように出力情報を取捨選択するために、ログメッセージの種類を指定することを「フィルタ」と呼びます。

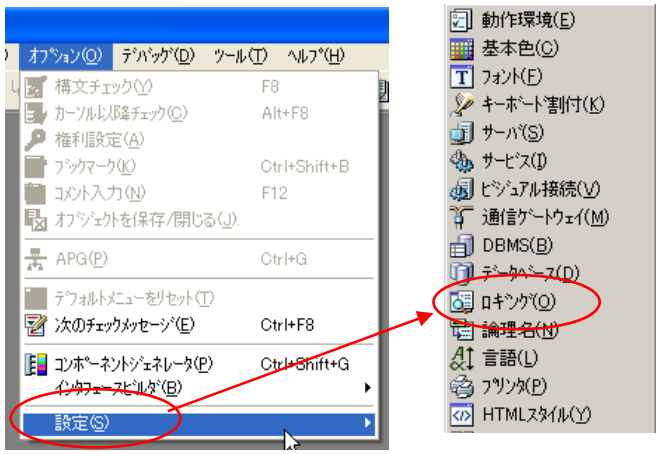
アクティビティモニタの出力内容をフィルタするには、次の三つの方法があります。

方法	説明	参照
ロギングダイアログ	ロギングダイアログを開いて、フィルタ項目を設定します。Studio でテスト実行する場合にだけ有効です。	2.4 Studio でロギングダイアログを開くには？
MAGIC.INI	MAGIC.INI の [MAGIC_LOGGING] セクションでフィルタ項目をパラメータで指定します。すべての uniPaaS 製品で有効です。	2.6 アクティビティ モニタをファイル出力させるには？
Logging () 関数	uniPaaS プログラムの実行中に、フィルタする内容を指定して Logging() 関数を実行します。アプリケーション実行中に、プログラムで動的にログレベルを変更することができます。	10.1 「Logging 関数」で説明します。

2.4. Studio でロギングダイアログを開くには？

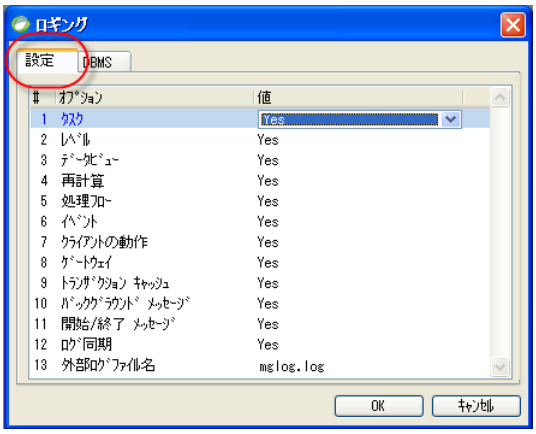
「ロギング ダイアログ」は、Studio 製品でだけ利用できます。ロギングダイアログを使うと、アクティビティログに出力するメッセージの種別を指定することができます。デバッグの目的に応じて、出力の詳細さを変更することができます。

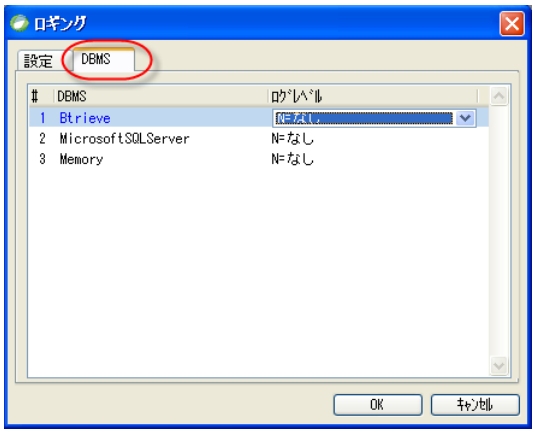
ロギング ダイアログ は、メニュー **オプション(O)** から **設定(S)** → **ロギング(O)** と選択します。



ロギング ダイアログには、設定 と DBMS という二つのタブがあります。

- **設定** タブを開くと、フローモニタに表示するトレースのレベルを設定できます。
- **DBMS** タブには、現在ゲートウェイがロードされている DBMS の一覧が表示され、それぞれにログレベルを設定できるようになっています。



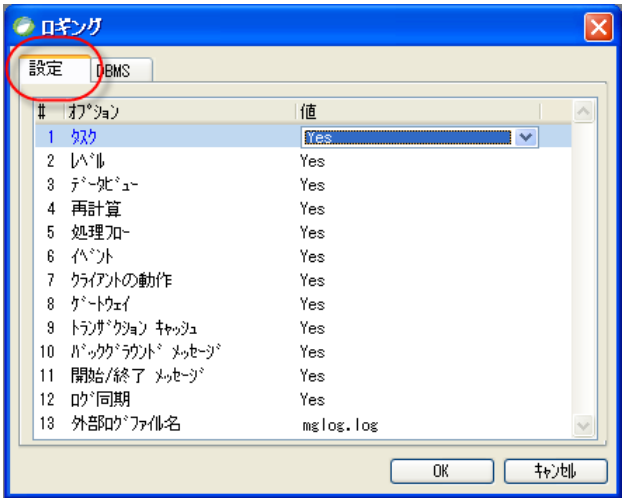


DBMS タブの設定は、ゲートウェイログ を出力する場合に使います。ゲートウェイログについては、2.7「ゲートウェイ ログとは？」で説明します。

2.5. ロギングダイアログでフィルタを設定するには？

アクティビティモニタのフィルタを設定するには、ロギングダイアログの「設定」タブで、アクティビティモニタに出力する情報の種類を設定します。次のようなオプションがあります。

ロギング ダイアログで設定した内容は、MAGIC.INI の [MAGIC_LOGGING] セクションに保存されます（次節参照）。このため、いったん Studio を終了しても、次回 Studio を起動したときには、前回の設定が記憶されています。



オプション	出力される内容
タスク	タスクの開始、あるいは終了時の、タスク名やタスクモード
レベル	タスク/レコード/コントロールの前処理/後処理などのハンドラが実行されたときの、レベル名やイベント名
データビュー	データ定義、範囲/位置付のような、データビューに関する情報
再計算	再計算処理時の変数名
処理フロー	処理コマンド名
イベント	トリガされたイベント名
クライアントの動作	リッチクライアントタスクにおける、クライアント側の動作のログ。 初期接続時にこのオプションが「No」に設定されている場合、あとから「Yes」に変更してもクライアントにログが作成されません。 「Yes」に設定された場合、サーバとの更新回数が増えるので、パフォーマンスが低下することがあります。
ゲートウェイ	ゲートウェイ ログの出力。ゲートウェイログを出力する場合、Yes にします。No にすると、「DBMS」タブでログレベルを設定してもアクティビティモニタには出力されません。
トランザクションキャッシュ	遅延トランザクション利用時の、トランザクションキャッシュへの操作。
バックグラウンドメッセージ	バックグラウンドメッセージがフィルタされるかどうかを指定します
開始/終了 メッセージ	ロギングモニタの処理の開始と終了の整合処理を有効にするかどうかを指定します
ログ同期	「外部ログファイル名」を指定することによりログをファイルに出力する際に、ログの同期を行うかどうかを指定します。ログの同期については、2.13 「ログの「同期」とは？」を参照してください。 なお、「ロギングダイアログ」では、「フラッシュ」の指定はできません。
外部ログファイル名	ログ情報をファイルに書き込みたい場合、ファイル名を指定します。 フルパスを指定しない場合、アプリケーションの作業フォルダ上に作成されます。

2.6. アクティビティ モニタをファイル出力させるには？

Studio 以外の製品では、アクティビティ モニタ 画面が表示されないので、アクティビティモニタはファイル出力だけができます。また、ロギング ダイアログが表示されないので、出力内容のフィルタの設定は MAGIC.INI の [MAGIC_LOGGING] セクションをテキストエディッタなどで直接編集して設定します。
アクティビティモニタをログファイルに出力するには、ログファイル名を MAGIC.INI の[MAGIC_LOGGING] で指定します。同時に、ここにはフィルタの設定も行ないます。

以下の表は、フィルタのオプションと、MAGIC.INI で指定するパラメータ名、および各パラメータに指定可能な値を示したものです。

オプション	MAGIC.INI パラメータ名	設定可能な値
タスク	Task	Y/N
レベル	Levels	Y/N
データビュー	DataView	Y/N
再計算	Recompute	Y/N
処理フロー	Flow	Y/N
イベント	Events	Y/N
クライアントの動作	LogClient	Y/N
ゲートウェイ	Gateway	Y/N
トランザクションキャッシュ	TransCache	Y/N
バックグラウンドメッセージ	BackgroundMsg	Y/N
開始/終了 メッセージ	BeginEndMsg	Y/N
ログ同期	LogSynch	Y/N/F
外部ログファイル名	ExternalLogFileName	(ログファイル名)

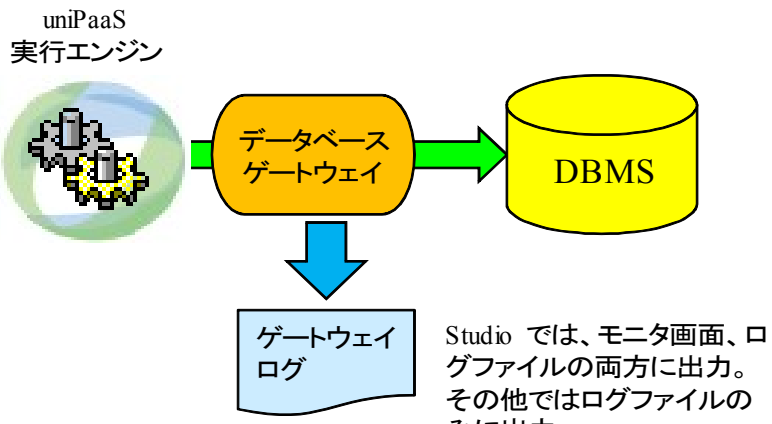
- 外部ログファイル名 (ExternalLogFileName) の指定は必須です。
- その他のパラメータは、指定されていなければデフォルト N とみなされます。
- ログ同期 (LogSynch) には、「Y」(同期あり)、「N」(同期なし)の他に、「F」(フラッシュ)の指定をすることができます。(2.13 「ログの「同期」とは？」参照)

設定例: 次の設定例では、タスクレベルの出力のみを行い、ログ同期を行ないます。また、ログファイル名は mgmonitor.log であり、エンジンの作業ディレクトリ (アプリケーションの ECF ファイルの存在するディレクトリ) に作成されます。

```
[MAGIC_LOGGING]
Task = Y
Levels = N
DataView = N
Recompute = N
Flow = N
Events = N
LogClient = N
TransCache = N
LogSynch = Y
BeginEndMsg = N
Gateway = N
BackgroundMsg = N
ExternalLogFileName = mgmonitor.log
```

2.7. ゲートウェイ ログとは？

Magic が DBMS にデータアクセスを行うときには、その DBMS 用のゲートウェイを通してアクセスします。従って、Magic と DBMS との間のデータ・制御のやりとりはすべて DBMS ゲートウェイを通して行うことになります。



Magic 実行時に、予期しないレコードロック、テーブルロック、ファイルマネージャの異常、パフォーマンスの問題などが起こった場合には、Magic が DBMS をどのように利用しているのか、あるいは、低レベルの DBMS とのインターフェースにおいて、どのようなデータやステータスコードが現れているのかなどを確認することが、問題解決につながることがあります。

Magic の DBMS ゲートウェイは、この目的のためのログ機能を備えています。

ゲートウェイログは、アクティビティ モニタの機能の一環として実装されていて、アクティビティモニタの画面、あるいはログファイルに出力されます。

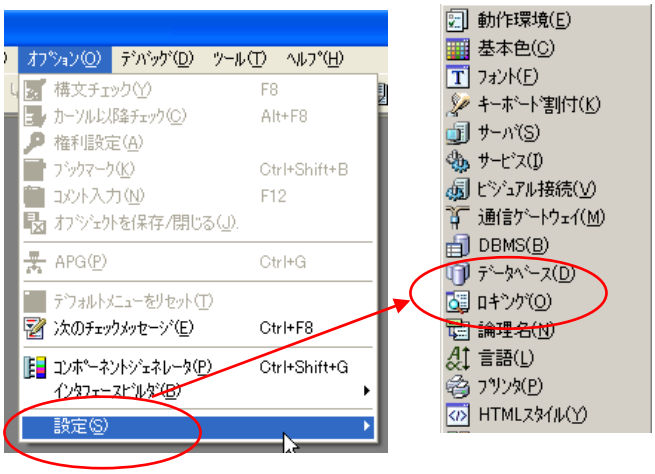
ゲートウェイログの設定は、アクティビティ モニタの設定と同様に、次の三つの方法により行うことができます。

方法	説明	参照
ロギングダイアログ	ロギングダイアログを開いて、フィルタ項目を設定します。Studio でテスト実行する場合にだけ有効です。	2.8 ロギングダイアログでゲートウェイログを設定するには？
MAGIC.INI	MAGIC.INI の [MAGIC_DBMS] セクションでログレベルをパラメータで指定します。すべての uniPaaS 製品で有効です。	2.9 MAGIC.INI でゲートウェイログを設定するには？
Logging () 関数	uniPaaS プログラムの実行中に、フィルタする内容を指定して Logging() 関数を実行します。アプリケーション実行中に、プログラムで動的にログレベルを変更することができます。	Logging 関数は、10.1「Logging 関数」で説明します。

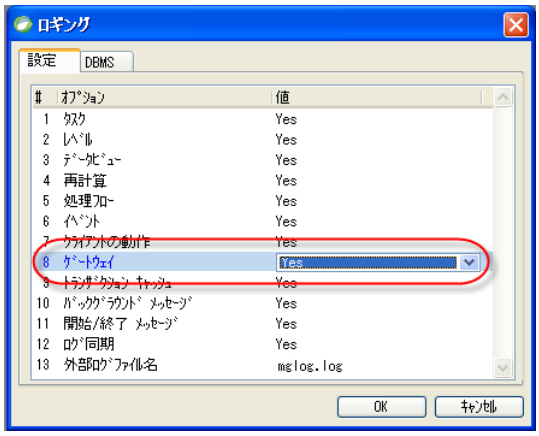
2.8. ロギングダイアログでゲートウェイログを設定するには？

Studio 製品上でテスト実行する場合には、ロギングダイアログを開いて、ゲートウェイログのログレベルを設定することができます。

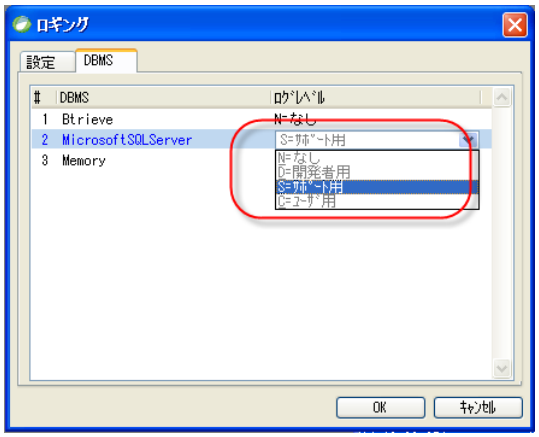
1. メニュー「**オプション(O) → 設定(S) → ロギング(O)**」を選択して、ロギングダイアログを開きます。



2. 「設定」タブで、「ゲートウェイ」を Yes に設定します。
このオプションが No だと、ゲートウェイログは出力されません。



3. 「DBMS」タブで、ログをとりたいデータベースについて、「ログレベル」を設定します。
右図では、MS SQL Server ゲートウェイに対し、「S=サポート用」レベルのログを出力することを指定しています。



「ログレベル」としては、以下の指定ができます。「N=なし」以外では、C=ユーザ用、S=サポート用、D=開発者用の順で出力量が多くなります。(2.12「ログ出力のレベルとは？」を参照)。

ログレベル	出力内容	出力量
N=なし	ログファイルが作成されません。	なし
D=開発者用	MSJ の技術部門向けのログファイルが作成されます。	大
S=サポート用	アプリケーション開発者向けのログファイルが作成されます。	中
C=ユーザ用	SQL コマンドが出力するログのみ出力されます。	小

2.9. MAGIC.INI でゲートウェイログを設定するには？

Studio 以外の製品では、ロギングダイアログが使えないので、MAGIC.INI の指定により、ゲートウェイログの出力を指定することになります。
ゲートウェイログは、[MAGIC_DBMS] セクションで、DBMS の設定の一部（第 5 番目のパラメータ）として指定します。
例えば、MS SQL Server で「S=サポート用」レベルのゲートウェイ ログを出力させるためには、次のように指定します。

```
[MAGIC_DBMS]
MicrosoftSQLServer = 21, NotAllowNull, 10.3, MicrosoftSQL Parameters, S, +
, NotLogSync, 0, 0, NotCheckExist,
```

ここで、「S」と指定されているものがゲートウェイログの指定で、「S=サポート用」レベルを意味しています。
「S」以外には、N（なし）、D（開発者用）、C（ユーザ用）が指定できます。

2.10. ログ出力の例

DBMS ゲートウェイは、DBMS ごとに内部処理が大きく異なりますので、ログも DBMS によりかなり異なるものとなりますが、特に、ISAM 系の DBMS ゲートウェイ (Pervasive, メモリなど) と、SQL 系の DBMS ゲートウェイ (Oracle、MS-SQL など)とは大きく異なります。
ここでは、詳しい解説はしませんが、以下に簡単なレコードフェッチのみを行った場合の例を挙げます。

2.10.1. ISAM 系 DBMS ゲートウェイの場合 (Pervasive)

Pervasive ゲートウェイのように、ISAM ファイルをアクセスするゲートウェイの場合には、DBMS が提供する API へのオペレーション、データ、戻り値などが記録されます。
赤色で示したところが、Pervasive の提供する API (BTRV()と記されている)を呼び出している部分です。ファイルのオープン (OPEN)、STEP 命令や GET 命令などが発行されている様子がわかります。

```
17:49:03.546    - 3,91359  LogStart() : >>>>> ***** Btrieve *****
17:49:03.546    -  ,91359  Version uniPaaS 1.8 SP1a, Log = mglog.log,  Level = Support/QA,  Sync = None,  Date = 30/08/2010
17:49:03.546    -  ,91359  LogStart() : <<<<<
17:49:10.000    -   0,97812  ==> _fil_open()   open Counters (W,W,N) : normal (C:¥Program Files¥uniPaaS¥Studio
V1Plus¥Projects¥TravelAgency¥Data¥)
17:49:10.000    -   ),97812  ==> BTRV() #1   OPEN (0 )
17:49:10.000    -   ,97812   >>
17:49:10.000    -   ,97812   rec buf  =  00
17:49:10.000    -   ,97812  <== OK (0)
17:49:10.000    -   ,97812  ==> BTRV() #2   STAT (15)
17:49:10.015    -   ,97828   <<          1. 2. 3. 4. 5. 6. 7. 8. 9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.
17:49:10.015    -   ,97828   rec buf  =  28 00 00 0a 01 00 01 00 00 00 00 12 00 00 00 00 01 00 02 00 02 01 01 00 00 00 01 00 00 00 00 00
17:49:10.015    -   ,97828   .....  ( . . . . . .
17:49:10.015    -   ,97828  <== OK (0)
17:49:10.015    -   ,97828  ==> BTRV() #3   STAT (15)
17:49:10.015    -   ,97828   <<          1. 2. 3. 4. 5. 6. 7. 8. 9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.
17:49:10.015    -   ,97828   rec buf  =  28 00 00 0a 01 60 01 00 00 00 00 12 00 00 00 00 01 00 02 00 02 01 01 00 00 00 01 00 00 00 00 00
17:49:10.015    -   ,97828   .....  ( . . . . ` . . . . . .
17:49:10.015    -   ,97828  <== OK (0)
17:49:10.015    -   ,97828 <== _fil_open() OK
17:49:10.015    -    ,97828 ==> _crsr_get() GET_GTEQ : crsr #0 (key #1, Counters)
17:49:10.015    -    ,97828  key buf =   00 80
17:49:10.015    -    ,97828  .....  . .
17:49:10.015    -    ,97828  ==> BTRV() #4   GET_GE (9 )
17:49:10.015    -    ,97828   <<          1. 2. 3. 4. 5. 6. 7. 8.
9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.33.34.35.36.37.38.39.40.
17:49:10.015    -    ,97828   rec buf  =  01 00 4f 72 64 65 72 73 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00 00 00 00 00 00 44 40
17:49:10.015    -    ,97828   .....  . . O r d e r s                                                  . . . . . D @
17:49:10.015    -    ,97828  <== OK (0)
17:49:10.015    -    ,97828  ==> BTRV() #5   GET_POSITION (22)
17:49:10.015    -    ,97828   <<
17:49:10.015    -    ,97828   rec buf  =  00 00 06 0a
17:49:10.015    -    ,97828  <== OK (0)
17:49:10.015    -    ,97828 <== _crsr_get() OK
17:49:10.015    -    ,97828 ==> _crsr_set_curr() crsr #0 (key #1, Counters)
17:49:10.015    -    ,97828  ==> BTRV() #6   GET_DIRECT/CHUNK (23)
17:49:10.015    -    ,97828   >>
17:49:10.015    -    ,97828   rec buf  =  00 00 06 0a
17:49:10.031    -    ,97843   <<          1. 2. 3. 4. 5. 6. 7. 8.
9.10.11.12.13.14.15.16.17.18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.33.34.35.36.37.38.39.40.
17:49:10.031    -    ,97843   rec buf  =  01 00 4f 72 64 65 72 73 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
00 00 00 00 00 00 44 40
17:49:10.031    -    ,97843   .....  . . O r d e r s                                                  . . . . . D @
17:49:10.031    -    ,97843  <== OK (0)
```

```
17:49:10.031    -     ,97843  <== _crsr_set_curr() OK
17:49:10.031    -     ,97843  ==> _crsr_get() GET_NEXT : crsr #0 (key #1, Counters)
17:49:10.031    -     ,97843   key buf =    01 00
17:49:10.031    -     ,97843   .....    . .
17:49:10.031    -     ,97843   ==> BTRV() #7   GET_NEXT_EXTENDED (36)
17:49:10.031    -     ,97843    >>           1. 2. 3. 4. 5. 6. 7. 8. 9.10.11.12.13.14.15.16.
17:49:10.031    -     ,97843   rec buf   =   10 00 45 47 00 00 00 00 1e 00 01 00 28 00 00 00
17:49:10.031    -     ,97843   ...........   . . E  G . . . . . . . . ( . . .
17:49:10.031    -     ,97843    <<
17:49:10.031    -     ,97843   rec buf   =   00 00
17:49:10.031    -     ,97843   <== KNW_END_OF_FILE (9)
17:49:10.031    -     ,97843  <== _crsr_get() DB_ERR_NOREC
17:49:11.500    -    1,99312  ==> _fil_close()   close Counters
17:49:11.500    -     ,99312   ==> BTRV() #8   CLOSE (1 )
17:49:11.500    -     ,99312   <== OK (0)
17:49:11.500    -     ,99312  <== _fil_close() OK
```

2.10.2. SQL 系 DBMS ゲートウェイの場合 (MS-SQL)

SQL 系の DBMS ゲートウェイが出力するログでは、DBMS の提供する低レベルの API レベルでの操作よりも、発行される SQL 文やその結果の方が重要です。
以下の例では、赤色で示した部分で、テーブルの存在確認、トランザクションの制御、レコード検索のための SELECT 文、データのフェッチなどが行われている様子が見て取れます。

```
17:54:11.093    -   1,98906  ms7_fil_open(): >>>>> ctxID = -1.000000, database = MAGIC, table = Counters, dbd->opened=0, share = W
17:54:11.093    -   ),98906  ms7_connect_extra_session(): >>>>> database = MAGIC, pConnection->sess_extra = 0
17:54:11.093    -    ,98906  ms7_connect_extra_session(): >>>>> database = MAGIC, pConnection->sess_extra = 0
17:54:11.093    -    ,98906  ms7_esqlc_fil_exist() : db - MAGIC, owner - dbo, table - Counters
17:54:11.093    -    ,98906  ms7_esqlc_fil_exist():   select id from MAGIC..sysobjects where name = 'Counters' AND uid = user_id('dbo') AND
(type = 'U' OR type = 'V' OR type = 'S')
17:54:11.093    -    ,98906  ms7_fil_open(): <<<<< ctxID = -1.000000, retcode = 0
17:54:11.093    -    ,98906  ms7_crsr_prepare(): >>>>> ctxID = -1.000000, database = MAGIC, key_idx = 0
17:54:11.093    -    ,98906  ms7_crsr_prepare(): table = Counters
17:54:11.093    -    ,98906  ms7_crsr_prepare(): number of blobs = 0
17:54:11.093    -    ,98906  ms7_connect_extra_session(): >>>>> database = MAGIC, pConnection->sess_extra = 0
17:54:11.093    -    ,98906  ms7_crsr_prepare(): <<<<< ctxID = -1.000000, retcode = 0
17:54:11.093    -     ,98906  ms7_crsr_begin(): >>>>> ctxID = -1.000000
17:54:11.093    -     ,98906  ms7_crsr_begin(): <<<<< ctxID = -1.000000, retcode = RET_OK
17:54:11.093    -     ,98906  ms7_trans(): >>>>> ctxID = -1.000000, transmode = 1, db = 20
17:54:11.093    -     ,98906  Hold thru ms7_trans(): >>>>> ctxID = -1.000000, serverID = 3, dbconn_hdl = 1
17:54:11.093    -     ,98906  Hold thru ms7_trans(): <<<<< ctxID = -1.000000, errcode = 0, RC = RC_OK
17:54:11.093    -     ,98906  session - 0, in use - 1, write - 1, reusable - 1, results pending - 0,not only for cursors - 1
17:54:11.093    -     ,98906  ms7_trans(): OPEN_READ
17:54:11.093    -     ,98906  ms7_trans(): <<<<< ctxID = -1.000000, retcode = 0
17:54:11.093    -     ,98906  ms7_crsr_open(): >>>>> ctxID = -1.000000, dbd_hdl = 0
17:54:11.093    -     ,98906  ms7_crsr_open(): db = MAGIC, table = Counters, crsr_hdl = 0, dir = A, dir_rev = 0 , rngs = 0, Range:FALSE, Start
Pos:FALSE
17:54:11.093    -     ,98906  ms7_crsr_open(): checking if dummy fetch available
17:54:11.109    -     ,98921  ms7_connect_extra_session(): >>>>> database = MAGIC, pConnection->sess_extra = 0
17:54:11.109    -     ,98921  SET IMPLICIT_TRANSACTIONS ON
17:54:11.109    -     ,98921     STMT: SELECT ｶｳﾝﾀｰｺｰﾄﾞ,ｶｳﾝﾀｰ説明,ｶｳﾝﾀｰ値 FROM MAGIC.dbo.Counters ORDER BY ｶｳﾝﾀｰｺｰﾄﾞ ASC
17:54:11.109    -     ,98921     STMT: SELECT ｶｳﾝﾀｰｺｰﾄﾞ,ｶｳﾝﾀｰ説明,ｶｳﾝﾀｰ値 FROM MAGIC.dbo.Counters ORDER BY ｶｳﾝﾀｰｺｰﾄﾞ ASC
17:54:11.109    -     ,98921  Using CURSOR
17:54:11.109    -     ,98921  ms7_crsr_open(): <<<<< ctxID = -1.000000, retcode = 0
17:54:11.109    -     ,98921  ms7_crsr_fetch(): >>>>> ctxID = -1.000000, dbd_hdl = 0, lock = FALSE
17:54:11.109    -     ,98921   FETCH Read0
17:54:11.109    -     ,98921     RESULT: 1
17:54:11.109    -     ,98921   RESULT:  Orders                          RESULT: 40.000000
17:54:11.109    -     ,98921  ms7_crsr_fetch(): <<<<< ctxID = -1.000000, retcode = 0
17:54:11.109    -     ,98921  ms7_crsr_fetch(): >>>>> ctxID = -1.000000, dbd_hdl = 0, lock = FALSE
17:54:11.109    -     ,98921  ms7_crsr_fetch(): <<<<< ctxID = -1.000000, retcode = 60
17:54:11.109    -     ,98921  ms7_crsr_close(): >>>>> ctxID = -1.000000, crsr_hdl = 0
```

```
17:54:11.109  -     ,98921  ms7_crsr_close(): <<<<< ctxID = -1.000000, retcode = RET_OK
17:54:11.109  -     ,98921  ms7_trans(): >>>>> ctxID = -1.000000, transmode = 8, db = 20
17:54:11.109  -     ,98921  session - 0, in use - 1, write - 1, reusable - 1, results pending - 0,not only for cursors - 1
17:54:11.109  -     ,98921  ms7_trans(): COMMIT
17:54:11.109  -     ,98921  SET IMPLICIT_TRANSACTIONS OFF
17:54:11.109  -     ,98921  ms7_trans(): <<<<< ctxID = -1.000000, retcode = 0
17:54:12.656  -    2,00468  ms7_crsr_end >>>>> ctxID = -1.000000
17:54:12.671  -     ,00484  ms7_crsr_end <<<<< ctxID = -1.000000, retcode = RET_OK
17:54:12.671  -     ,00484  ms7_crsr_release(): >>>>> ctxID = -1.000000
17:54:12.671  -     ,00484  ms7_crsr_release(): database = MAGIC, table name = Counters, crsr_hdl = 0
17:54:12.671  -     ,00484  ms7_crsr_release(): <<<<< ctxID = -1.000000, retcode = 0
17:54:12.671  -     ,00484  ms7_fil_close(): >>>>> ctxID = -1.000000, file : Counters, dbd->opened = 0
17:54:12.671  -     ,00484  ms7_fil_close(): <<<<< ctxID = -1.000000, retcode = 0
```

2.11. 実行エンジンエラーログ

2.11.1. 実行エンジンエラーログとは？

実行エンジンエラーログは、Magic アプリケーションサーバの実行中に重要な問題が起こった場合の、エラー情報を記録するものです。Magic プログラムがフォアグランドで動作している場合には、実行中にエラーが起こった場合に、ステータス行にもエラーメッセージが表示されます。
問題が起こった場合には、このログをチェックすることにより、問題についての情報を得ることができます。

具体的には、次のような情報が日時と共に出力されます。

- フォアグランドモードであればステータス行に表示されるようなエラー・ワーニングメッセージ。これには、レコードやテーブルロック、DBMS のエラー、テンプレートファイルが開けない問題、UDF/UDP が見つからない、その他、Magic の実行の継続ができなくなるようなエラーなどが含まれます。
- 「エラー」コマンドにより出力したメッセージ
- ライセンス関係のエラー。
- MRB やリクエスタとの通信中に起こった問題の記録。

実行エンジンエラーログは、プロジェクトを開く以前の問題(ライセンスの不正など)に対しては、uniPaaS 製品をインストールしたディレクトリに出力されます。プロジェクトを開いて実行中のエラーに関しては、プロジェクトディレクトリに出力されます。

2.11.2. 実行エンジンエラーログを出力させるには？

実行エンジンエラーログは、MAGIC.INI のパラメータ GeneralErrorLog で設定されます。デフォルトでは、mgerror.log となっています。この設定を空白にすると、ログは出力されないようになります。

```
GeneralErrorLog = mgerror.log
```

2.11.3. ログファイル例

次に示すのは、エンジンエラーログの例です。各種のエラーが記録されているのがわかります。

```
 24/06/2010 17:48:09.817  -  >> ｴﾗｰ >>>> 処理に失敗しました.他のユーザか処理にによってレコードが変更されています.データソース:
DUPREC_TBL2, program : deftrn
 24/06/2010 17:57:43.201  -  >> ｴﾗｰ >>>> オープンできません.データソース:DUPREC_TBL1, program : deftrn.create TBL1
 24/06/2010 18:12:26.804  -  >> ｴﾗｰ >>>> インデックスが重複しています.データソース: DUPREC_TBL1, program : deftrn
 24/06/2010 19:48:09.817  -  >> ｴﾗｰ >>>> インデックスが重複しています.テーブル: DENPYO
 24/06/2010 20:57:43.201  -  >> ｴﾗｰ >>>> MRGSendResponse(): : "LOG_ON_1" ("KAISHA_SYS") (msgid 81)
 24/06/2010 21:12:26.804  -  >> ｴﾗｰ >>>> ライセンス上の接続可能数を超えました.
 24/06/2010 22:12:39.590  -  >> ｴﾗｰ >>>> インデックスが重複しています.テーブル: W_DENPYO
 24/06/2010 23:12:41.529  -  >> ｴﾗｰ >>>> MRGSendResponse(): ERR-RUN-TIME-EXCEPTION(-139)<BR> : "MENU_H"
("KAISHA_SYS") (msgid 1236)
 25/06/2010 01:48:09.124  -  >> ｴﾗｰ >>>> ユーザモジュールが見つかりません.
 24/06/2010 02:57:43.567  -  >> ｴﾗｰ >>>> ユーザモジュールが見つかりません.
 25/06/2010 08:12:26.436  -  >> ｴﾗｰ >>>> ユーザモジュールが見つかりません.
 25/06/2010 12:12:39.742  -  >> ｴﾗｰ >>>> ユーザモジュールが見つかりません.
 25/06/2010 16:12:41.589  -  >> ｴﾗｰ >>>> テンプレートファイルのオープン処理の問題です. -
```

2.12. ログ出力のレベルとは？

ゲートウェイログ、あるいは後述のリクエスタログ（第 3 章「リクエスタログ」）では、ログの詳細さにより、
以下の3レベルがあります。（指定文字は、大文字でも小文字でも構いません）

指定文字	レベル	説明
C	簡易	一番出力量が少ない設定です。オーバーヘッドも最小ですが、出力が少なすぎて必要な情報が記録されていないことがあります。
S	サポート	中間的な量のログが出力されます。ログの調査を行う場合には、最初はこのレベルが適当でしょう。
D	開発者	詳細なログが出力されます。通常はこのレベルのログは必要ありませんが、MSJ で詳細な調査を行う必要がある場合に、このレベルのログが必要になることがあります。

必要な情報の種類と、実行速度の許容範囲とから、適当なログレベルを選択してください。

一般的には、次のような選択になると思われます。

- 調査の初期段階で、「どのあたりに問題がありそうか？」について当たりをつける段階では、全体を見渡しやすく、負荷も小さい C レベルを選択。
- 調査が進んでより深い情報が必要になってきたら S レベルを選択。
- MSJ のサポートセンターとのやりとりで必要になってきたら D レベルを選択。

2.13. ログの「同期」とは？

uniPaaS モジュールが出力するログで、「ログの同期」という設定がよく出てきます。これは、ログの各行をファイルに出力する際に、どのタイミングでディスクへの書き込みを行うか、を指定するもので、ここにはログの確実性(異常終了が起こった場合にログがどれだけ確実にファイルに書き込まれるか？)と、オーバーヘッドのトレードオフがあります。

2.13.1. ログの同期の種類

ログの同期には、以下の3種類があり、各ログの指定の箇所において、「設定文字」のような文字により指定されます。（設定文字は、大文字でも小文字でも構いません）

同期	設定文字
同期なし	N
同期あり	Y
フラッシュ	F

デフォルトの設定は「同期なし」であり、ログ出力は、uniPaaS モジュールのプロセスのメモリ内にバッファリングされます。これは C++言語のランタイムルーチンで自動的に行われている処理であり、ディスクへの書き込みのオーバーヘッドを最小にするために、一定量のデータが溜まるまでメモリ内に格納しておき、それを超えたら初めてディスクへ書き込む、という処理を行ないます。
バッファリングを行うと、ログ出力に伴なうオーバーヘッド(処理速度の低下)を最小限に抑えることができますが、問題点として、プロセスが異常終了した場合に、バッファリングされていてまだディスクに書き込まれていないデータは、プロセスと一緒に失われてしまう、という点があります。異常終了の原因を追求するためにログの内容を確認しようとしても、異常終了の直前の最も重要なタイミングでのデータが失われてしまって、原因追求が思うようにできないことになります。従って、「同期あり」の設定は、異常終了しない場合の問題調査の場合に設定するのが適当です。

「同期あり」にすると、バッファリングは行われず、1行出力するごとに、ファイルのオープン→書き込み→ファイルのクローズ という処理が行われます。このような処理を行うことにより、ログの各行が確実にディスク上に書き込まれるようになり、万一異常終了した場合でも、直前のログまでファイルに記録されるようになります。
一方、同期を行うことの問題点としては、1行ごとにファイルのオープン・クローズが行われるために、非常にオーバーヘッドが大きくなり、実行速度が遅くなることが挙げられます。実行速度が遅くなっても、確実にログを記録しておきたい、という場合には同期を行うように設定をしてください。

同期についての第三の選択肢として、「フラッシュ」という設定があります。これは、バッファリングは行わず、ログの行ごとにディスクにデータを書き込むようにするが、行ごとのファイルのオープン・クローズは行わない、というものです。この設定を行うと、異常終了時のログの確実性とオーバヘッドは、「同期あり」の場合と「同期なし」の場合の中間になります。

2.13.2. 同期の設定例

例として、アクティビティモニタのログ出力を MAGIC.INI で設定する場合 (2.6 「アクティビティ モニタをファイル出力させるには？」参照)には、次の行のようにします。これは、「同期あり」を指定した場合の例です。

```
LogSynch = Y
```

2.14. RIA クライアントログとは？

2.14.1. RIA クライアントログとは？

RIA クライアントログは、uniPaaS RichClient サーバシステムにおいて、クライアント側のモジュール uniRC.exe が出力する実行ログです。これには、クライアント側の通信内容、イベント処理、データやタスクの情報などのデバッグ情報が記録されます。
このログは、通常、uniPaaS アプリケーションの開発者が読んで解析する目的のものではありません。内部動作にかかわる記録がほとんどなので、MSJ のサポートセンターが問題の分析を行うためのものです。

2.14.2. RIA クライアントログの設定方法（開発時）

開発時、Studio を使ってデバッグ実行する際には、RIA クライアントログの指定は、MAGIC.INI ファイルの [MAGIC_RIA] セクションで行ないます。

```
[MAGIC_RIA]
ClientModulesPath=RIAModules¥uniRC_1_8_1_440¥
InternalLogLevel=
InternalLogFile=
InternalLogSync=Message
DisplayStatisticInformation=N
```

このうち、以下のパラメータがログ設定に関連します。

パラメータ名	設定可能な値	説明
InternalLogLevel（ログレベル）	SERVER	HTTP リクエストと応答のみ。
	SUPPORT	SERVER 指定の内容に加え、サーバ間のデータの内容。
	GUI	クライアントによって記録された GUI メッセージの一部。
	DEV	クライアントによって記録されるすべてのメッセージ。
	（なし）	出力しない。
InternalLogFile（ログファイル名）	ファイル名	指定したファイルにログが出力されます。 ファイル名を指定しなければ、クライアントのデスクトップに uniRC_YYYY_MM_DD[.Process ID].log というファイル名で保存されます。
InternalLogSync（ログの同期）	None	ログの各行について、ログファイルへの同期が行われます。
	Session	ログファイルへの同期が行われません。このレベルは最も処理が早くなりますが、uniRC.exe の異常終了/ハングアップ時にログがすべて書き込まれない可能性があります。
	Message	ログファイルは、各メッセージ毎にオープン/クローズされます。このレベルは、ログの保全性には優れていますが動作が遅くなります

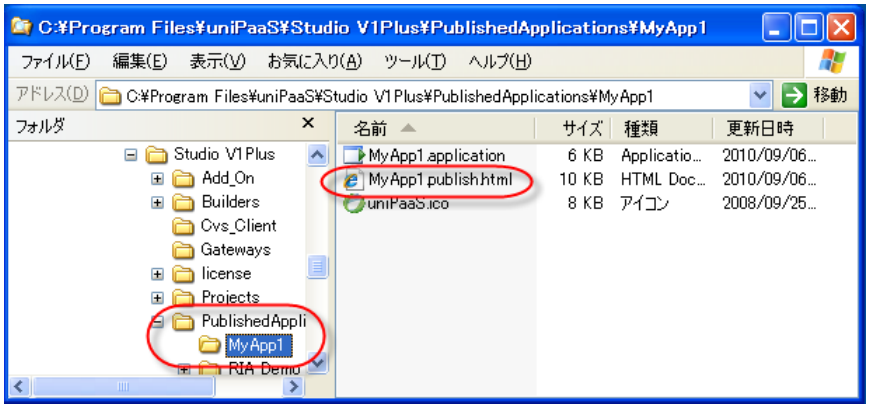
2.14.3. RIA クライアントログの設定方法（実行時）

Studio を使わず、RichiClient Server で RichClient プログラムを実行させる場合には、RIA クライアントログの設定は、起動用の.publish.html ファイル中で行ないます。

.publish.html ファイルというのは、uniPaaS Studio の「リッチクライアントインターフェースビルドダ」が作成するファイルの一つで、デプロイメントマニフェストファイル .application ファイルと一緒に作成されます。作成場所は、Studio をインストールしたディレクトリの下にある PublishedApplications¥（アプリケーション名）¥ サブディレクトリ中に作成されます。
ここで作成されるファイル名は、以下の二つがあります。

- （アプリケーション名）.application … ClickOnce で利用するデプロイメントマニフェストファイル
- （アプリケーション名）.publish.html … 起動用 HTML ページ

下図は、MyApp1 という名前のアプリケーションについて作成されたファイルです。



このうち、（アプリケーション名）.publish.html ファイルをテキストエディタで開いてみると、次のような部分があります。このうち、key="InternalLogLevel" 以下の 3 行で、ログについての指定を行ないます。

```
<xml id="rcExecProps">
    <properties>
      <property key="protocol" val="http"/>
      <property key="server" val="MyRCServer"/>
      <property key="requester" val="/uni18Scripts/Mgrqispi018.dll"/>
      <property key="appname" val="MyApp1"/>
      <property key="prgname" val="MyRCProg1"/>
      <property key="arguments" val=""/>
      <property key="envvars" val=""/>
      <property key="UseWindowsXPThemes" val="Y"/>
       （途中省略）
      <property key="InternalLogLevel" val="server"/>
      <property key="InternalLogFile" val="c:¥tmp¥rc.log"/>
      <property key="InternalLogSync" val="Session"/>
    </properties>
  </xml>
```

この「val="…"」の部分に、それぞれ、ログレベル、ログファイル名、ログ同期の設定を指定します。指定するキーワードは、MAGIC.INI で設定する場合(2.14.2「RIA クライアントログの設定方法（開発時）」)と同じです。

第3章　リクエストログ

Magic で Web 対応アプリケーションや3階層アプリケーションを作成した場合には、クライアント側での Web ブラウザ、Magic クライアント、サーバ側での Web サーバ、Magic リクエストブローカ、Magic アプリケーションサーバなど、多くのソフトウェアコンポーネントが関連して実行しているので、問題の切り分けが複雑なります。根本的な原因解明のためには、表面に現れる現象だけでなく、内部で起こっていることを正確に把握することが必要となります。

Magic リクエストログとは、Magic の Web アプリケーションや3階層アプリケーションに関係するモジュール間の通信のトレースを保存する機能です。このログを見ることによって、Magic の Web アプリケーションにおいて障害が起きた場合などに、原因の追求に役立ちます。ここでは、ログの種類と、その設定方法について説明します。

また、Magic のリクエストログ機能の他に、Windows オペレーティングシステムやサードパーティのソフトウェアにも、プログラムの実行状況を監視するさまざまなユーティリティがあります。このような機能を併せて活用することにより、内部の動作を把握し、問題解決に至るまでの時間を大幅に短縮することができます。

3.1. リクエスタログにはどんな種類がありますか？

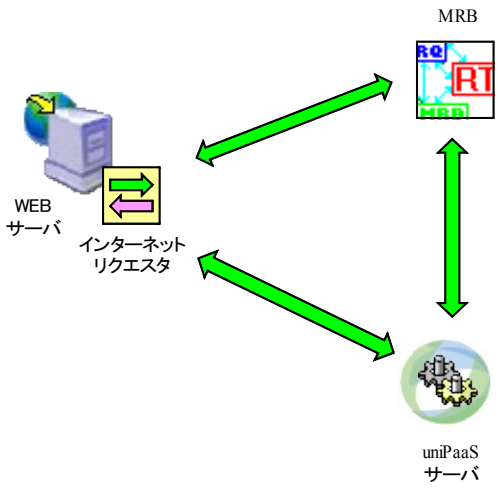
リクエスタログには、ログを出力するモジュールによって、数種類があります。最初に、リクエスタログを出力する uniPaaS モジュールについて説明します。

3.1.1. uniPaaS モジュールの種類

リクエスタログを出力する uniPaaS モジュールとしては、以下のようなものがあります。

モジュール名	ファイル名	ディレクトリ
uniPaaS サーバ実行エンジン	uniRTE.exe	uniPaaS ディレクトリ
リクエストブローカ (MRB)	uniRQBroker.exe	uniPaaS ディレクトリ
インターネットリクエスタ	MGrqispi018.dll MGrqcgi018.exe	Scripts ディレクトリ

ここで、インターネットリクエスタのファイル名にある「018」は、Magic のバージョン番号で、バージョンが変わると番号も変わります。
これらのモジュールは、右図のようにお互いに通信を行っており、それぞれの通信についてログが記録されることになります。



3.1.2. ログの種類

ログ出力が設定されていると、上記のそれぞれのモジュールが、他のモジュールとの通信を行うたびに、ログを書き込みます。次の表は、ログの種類と、それを出力するモジュール名、およびログ出力を設定する INI ファイルとを示します。

ログファイル	出力するモジュール名	読み込む設定ファイル
MRB 通信ログ	リクエストブローカ (MRB)	uniPaaS ディレクトリの MGRB.INI
uniPaaS サーバログ	uniPaaS サーバ実行エンジン	uniPaaS ディレクトリの MGREQ.INI
インターネットリクエスタログ	インターネットリクエスタ	Scripts ディレクトリの MGREQ.INI

このうち、実際のデバッグで一番役に立つログは、MRB 通信ログです。MRB はすべてのリクエストの交通整理を行い、また、uniPaaS サーバの実行状態を監視しているので、MRB のログを見ることにより、受け取ったリクエストの内容、リクエストを処理したエンジンの実行過程、エンジンの状態などを読み取ることができますので、

リクエストがどのように uniPaaS サーバで処理されたかを追跡するためには最適です。
uniPaaS サーバログおよびインターネットリクエスタログに記録されている内容は、TCP/IP に近い低レベルの通信状況なので、ネットワークレベルでの接続の問題がある場合には原因究明のために使います。

ログをとるとそれなりのオーバーヘッドがありますので、目的に応じて選択してください。

3.2. リクエスタログ設定時の注意点は？

リクエスタログを出力するにあたっては、
- 出力ディレクトリに関する注意点
- パフォーマンスに関する注意点

があります。

3.2.1. ログ出力ディレクトリに関する注意点

ログの出力にあたっては、ログ出力先となるディレクトリを適当に決めます。ログ出力先のディレクトリについては、次の点に注意してください。

- 長い間ログ出力を行っていると、ログファイルの合計サイズが非常に大きくなるので、ログ出力先のディレクトリは、十分な空き容量のあるドライブ上に指定する必要があります。ログのレベルやリクエストの量にもよりますが、数時間で数百 MB くらいになることもあります。また、適時、不要なログは削除してください。
- ディレクトリのセキュリティ権限の設定に気をつけてください。MRB や uniPaaS サーバをサービスとして実行する場合には、実行ユーザ（デフォルトではローカルシステムアカウント）での作成・書き込み権限が必要です。
- 同様に、インターネットリクエスタは、Web サーバと同じ Windows 資格情報で実行されますので、ログファイルを出力するディレクトリには、Web サーバを実行するユーザ での作成・書き込み権限が必要です。
- ログファイルは、決まった大きさ(数 MB くらい)になったら、自動的に分割され、日付と時刻などがファイル名に追加され、保存されます。(例えば、MRB.LOG が MRB_0210_1906.LOG などのようなファイル名に変更されます)。
- アクティブなログファイルにはなるべく触れないようにしてください。アクティブなファイルをテキストエディタで開いたり、別ディレクトリへコピーなどをしたりすると、一時的ですが排他ロックがかかるので、それ以降のログ出力がされないようになってしまう場合があります。アクティブなログファイルとは、MRB.LOG などのように、「Log =」パラメータで指定されたファイル名のものです。一方、「MRB_0210_1906.LOG 」などのファイル名を持つファイルは、サイズが大きくなったために分割・退避されたログファイルで、これはアクティブではありませんので、テキストエディタなどで開いても構いません。

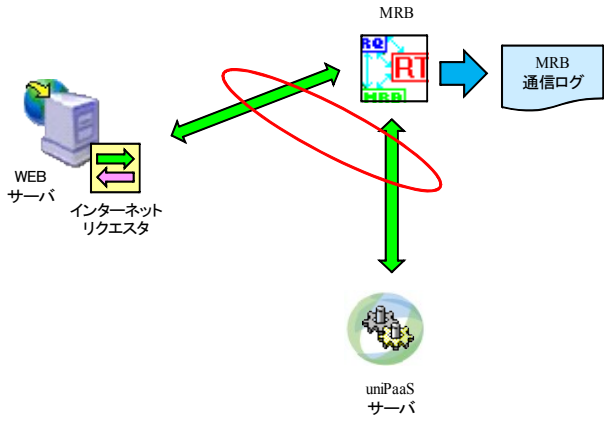
3.2.2. パフォーマンスに関する注意点

リクエスタログを出力すると、パフォーマンスに影響が出ます。一般に、リクエストの量が多いシステムほど、また、パフォーマンスとのバランスについて、次の項目を実測しながら調節してください。

- ログ出力レベル: ログのレベルが高い（詳細なログ出力を行う）ほど、実行速度が遅くなります。2.12「ログ出力のレベルとは？」を参照して、必要最小限なログレベルを設定してください。
- ログ同期: 異常終了はハングアップなど、実行モジュールの動作継続が不可能になってしまう場合のデバッグを目的とする場合には、同期＝Y の設定が必要になりますが、オーバーヘッドが非常に大きいので、同期＝フラッシュ（F）でもできないか、確認してください。実行モジュールの動作が継続する状況でのログ採取では、一般には同期＝N とするのが適当です。
- ログの種類: リクエスタログのうち、通常必要となるのは、「MRB 通信ログ」（3.3「MRB 通信ログの目的と設定は？」）です。TCP/IP レベルでのエラーの確認が必要な場合にだけ、UniPaaS サーバログ（3.5 uniPaaS サーバログの目的と設定は？）やインターネットリクエスタログ（3.6 インターネット リクエスタ ログの目的と設定は？）をとってください。

3.3. MRB 通信ログの目的と設定は？

MRB 通信ログは、インターネットリクエスタからのリクエスト、uniPaaS サーバとの管理上の通信内容など、MRB が行う通信内容を記録します。
MRB は uniPaaS サーバシステムの制御の要であるので、MRB 通信ログを取得することにより、リクエストの処理の流れを追っていくことができます。



MRB 通信ログを出力させるには？

MRB 通信ログを出力させるには、MRB モジュール（uniRQBroker.exe）が存在するディレクトリと同じディレクトリにある MGRB.INI ファイルで設定します。

1. Magic ディレクトリ中に、MGRB.INI というファイルがありますので、テキストエディタで開きます。
2. 「Log = ・・・」パラメータの行を探します。
3. デフォルトでは、行の最初にセミコロン「;」が入っていて、コメントアウトされていますので、セミコロンをはずします。
4. 「Log =」に続けて、ログファイル名、同期の有無、ログレベルを指定します。全体を括弧「(・・・)」でくくってください。
5. ファイルを保存します。
6. 設定を有効にするためには、MRB を再起動する必要があります。

例:

```
[MRB_ENV]
Log = (C:¥TMP¥LOG¥REQ.LOG Y S)
```

「Log =」に続くパラメータとしては、以下のものを空白文字で区切って指定します。

パラメータ順番	説明	例
1	ログファイル名（混乱を避けるため、フルパスで指定するのが良いでしょう）	C:¥TMP¥LOG¥REQ.LOG
2	ファイルの同期の有無を指定します。ログの同期については、2.13「ログの「同期」とは？」を参照してください。	Y（同期あり）
3	ログ出力のレベル：ログの詳細さを設定します。ログ出力のレベルについては、2.12「ログ出力のレベルとは？」を参照してください。	S（サポート）

以下に、MRB 通信ログの例を示します。これは、MyApp1 という名前のアプリケーションを実行しているエンタープライズサーバで、MyProg1 という公開プログラム名のプログラムを実行させたときのログです。

```
      - -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
,2892,2908   15:39:59,28437
,2068,2056   15:52:22,71406  Version uniPaaS 1.8 SP1a, Log = C:¥TMP¥LOG¥MRB.LOG,  Level = Support/QA,  Sync = Reopen,
Date = 01/09/2010
,2068,2056           ,71406  ==> Version uniPaaS 1.8 SP1b Broker (build Mar  3 2010) : starting INITIALIZATION
,2068,2056           ,71406   BrokerPort = /5215
,2068,2056           ,71406   CommTimeout = 1000
,2068,2056           ,71406   ReLoad     = TRUE
,2068,2056           ,71406   FloatingLicense = FALSE
,2068,2056           ,71421   Exe Entry : Online
,2068,2056           ,71421     command line        : uniRTE.exe /DeploymentMode=R
,2068,2056           ,71421     start directory     : C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus
,2068,2056           ,71421     username, password  : ,
,2068,2056           ,71421     on startup count    : 0
,2068,2056           ,71421     on autoload count   : 0
,2068,2056           ,71421  <== uniPaaS 1.8 SP1b Broker : INITIALIZED -OK (0)
      - -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
,2068,2056           ,71437
,2068,2052           ,71437
,2068,2228   15:52:27,77093  ==> mrb_incoming_msg()
,2068,2228   ( 5 sec),77093   Message ENGINE, -OK (0)
,2068,2228           ,77093   From engine : MyServer/1501, 192.168.0.1 (pid 660)
,2068,2228           ,77093   INITIALIZATION,
,2068,2228           ,77093   Opened application : "MyApp1"
,2068,2228           ,77093   pool_app_insert ("MyApp1")
,2068,2228           ,77093  <== mrb_incoming_msg() INF-ACK_SENT (-44)
      - -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
,2068,2228           ,77093  ==> mrb_incoming_msg()
,2068,2176   15:52:37,86796   ==> mrb_incoming_msg()
,2068,2176   (10 sec),86796    Message REQUEST SERVICE, -OK (0)
,2068,2176           ,86796    Reqid    : 0
,2068,2176           ,86796    Program  : MyProg1 ("MyApp1"), context -1, session 0
,2068,2176           ,86796    Priority : 0
,2068,2176           ,86796    Client   : 127.0.0.1 (pid 2232)
,2068,2176           ,86796    Filter   :
,2068,2176           ,86796    ==> Message ENGINE : MyServer/1501 (pid 660)
,2068,2176           ,86796    <== -OK (0)
,2068,2176           ,86796    ==> log_update(1) IN_PROGRESS, 0, 0, 0
,2068,2176           ,86796    <== log_update(1) 0 secs
,2068,2176           ,86796   <== mrb_incoming_msg() -OK (0)
,2068,2176           ,86796   ==> mrb_incoming_msg()
,2068,2228           ,86828    Message ENGINE, -OK (0)
,2068,2228           ,86828    From engine : MyServer/1501, 192.168.0.1 (pid 660)
,2068,2228           ,86828    COMPLETED (Reqid 1) -OK (0)
,2068,2228           ,86828    ==> log_update(1) DONE, 0, 0, 0
,2068,2228           ,86828    <== log_update(1) 0 secs
,2068,2228           ,86828   <== mrb_incoming_msg() -OK (0)
,2068,2228           ,86828   ==> mrb_incoming_msg()
,2068,2176   15:52:41,90796    Message END CONNECTION, INF-DONT-SEND-ACK (-21)
,2068,2176   ( 4 sec),90796   <== mrb_incoming_msg() INF-END-THREAD (-2)
,2068,2296   15:52:45,95203   ==> pool_delete() MyServer/1501
,2068,2296   ( 4 sec),95203   <== pool_delete() "OK" (0)
,2068,2228           ,95203   ==> pool_delete() MyServer/1501
,2068,2228           ,95203   <== pool_delete() ｸ･#
,2068,2228           ,95203  <== mrb_incoming_msg() INF-END-THREAD (-2)
      - -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
,2068,2056   15:52:49,99328
```

3.4. MRB イベントログとは？

MRB イベントログというのは、MRB の活動の中で、特に重要なイベント(MRB の起動、終了、アプリケーションサーバの起動・登録・削除など)が記録されているログファイルで、MRB のモジュール（uniRQBroker.exe）が存在するディレクトリに、mrb_event.log という名前で保存されます。

ここには、

- MRB の起動と終了
- [MRB_EXECUTABLES_LIST] に指定したプログラム自動起動
- Magic アプリケーションサーバの MRB への登録と登録削除（異常終了の場合も含む）

などの情報が、日時、プロセス ID、その他のパラメータと共に記録されます。

Web 対応アプリケーションあるいはリモートコールを使うアプリケーションにおいて、Magic や MRB の動作に問題がある場合にここの記録を見ることにより、問題原因の手がかりを得ることができます。

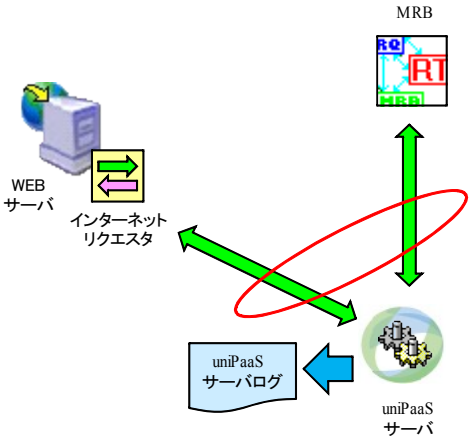
MRB イベントログを出力させるための設定はありません。設定せずとも自動的に出力されます。逆にいうと、「出力させない」あるいは「ログレベルを変更する」ということはできません。

例: 以下に mrb_event.log の例を示します。赤字で示したところは、Magic のアプリケーションサーバが異常終了したエラー記録の部分です。

```
1548   23:21:56,81718  28/08/2010 Startup  (Version uniPaaS 1.8 SP1a, build Dec  2 2009)
1548           ,81718  28/08/2010   BrokerPort = /5215
1548           ,81718  28/08/2010   CommTimeout = 1000
1548           ,81718  28/08/2010   ReLoad     = TRUE
1548   23:21:59,84448  28/08/2010   Remote Spawn on "192.168.1.3/5225" ("MyApp1") : "OK" (0)  (args: -
[MAGIC_SERVERS]__internal_broker=2,MyEntServer/5215,,[[[RemoteApp1]]]********,,,1 -MessagingServer=__internal_broker
-ActivateRequestsServer=Y)
1548   23:22:01,86975  28/08/2010   Remote Spawn on "192.168.1.3/5235" ("MyApp1") : "OK" (0)  (args: -
[MAGIC_SERVERS]__internal_broker=2,MyEntServer/5215,,[[[RemoteApp3]]]********,,,1 -MessagingServer=__internal_broker
-ActivateRequestsServer=Y)
1548   23:22:03,89299  28/08/2010   Remote Spawn on "192.168.1.3/5245" ("MyApp1") : "OK" (0)  (args: -
[MAGIC_SERVERS]__internal_broker=2,MyEntServer/5215,,[[[RemoteApp5]]]********,,,1 -MessagingServer=__internal_broker
-ActivateRequestsServer=Y)
3544           ,89299  28/08/2010   Error: "TCP/IP: Connection reset" (-144)  (/0)
2880   23:22:04,89970  28/08/2010   Enterprise Server MyEntServer/1612 : Inserted  (pid  2424  , UniCacklePlus  ,
license 0 threads | 0 requests, "RemoteApp1")
3424   23:22:06,91514  28/08/2010   Enterprise Server MyEntServer/1515 : Inserted  (pid  4016  , UniCacklePlus  ,
license 0 threads | 0 requests, "RemoteApp2")
3244   23:22:07,93106  28/08/2010   Enterprise Server MyEntServer/1607 : Inserted  (pid  3332  , UniCacklePlus  ,
license 0 threads | 0 requests, "RemoteApp3")
2380   23:32:41,26517  28/08/2010 ......  Enterprise Servers shutdown: ......
2380           ,26641  28/08/2010   Enterprise Server MyEntServer/1515 : Instructed to terminate : "OK" (0)
2380           ,26641  28/08/2010   Enterprise Server MyEntServer/1515 : Removed : "OK" (0)
3424           ,26719  28/08/2010   Enterprise Server MyEntServer/1515 : Notified termination
2380           ,27016  28/08/2010   Enterprise Server MyEntServer/1612 : Instructed to terminate : "OK" (0)
2380           ,27016  28/08/2010   Enterprise Server MyEntServer/1612 : Removed : "OK" (0)
2880           ,27094  28/08/2010   Enterprise Server MyEntServer/1612 : Notified termination
2380           ,27265  28/08/2010   Enterprise Server MyEntServer/1830 : Instructed to terminate : "OK" (0)
2380           ,27265  28/08/2010   Enterprise Server MyEntServer/1830 : Removed : "OK" (0)
2380           ,27265  28/08/2010 ......  Broker shutdown: ......
1548   23:32:48,33989  28/08/2010 ......  Shutdown completed ......
```

3.5. uniPaaS サーバログの目的と設定は？

uniPaaS サーバログは、uniPaaS サーバが出力するログファイルで、インターネットリクエスタや MRB との通信の状況を記録しています。uniPaaS サーバ動作中に、接続が不安定になってエラーが発生するなど、ネットワークレベルの問題が疑われる場合に収集して確認します。



MRB 通信ログを出力させるには？

UniPaaS サーバログは、uniPaaS サーバモジュール uniRTE.exe が存在するディレクトリ（uniPaaS のインストールディレクトリ）と同じディレクトリにある、MGREQ.INI ファイルで指定します。

1. uniPaaS ディレクトリ中に、MGreq.ini というファイルがありますので、テキストエディタで開きます。
2. 「Log = ・・・」パラメータの行を探します。
3. デフォルトでは、行の最初にセミコロン「;」が入っていて、コメントアウトされていますので、セミコロンをはずします。
4. 「Log =」に続けて、ログファイル名、同期の有無、ログレベルを指定します。
5. ファイルを保存します。
6. 設定を有効にするためには、uniPaaS サーバを再起動する必要があります。

例：

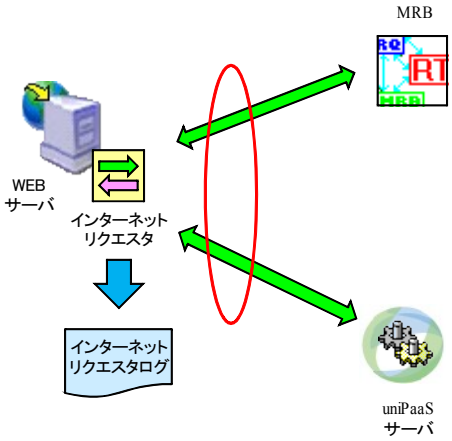
```
[REQUESTER_ENV]
Log = C:¥TMP¥LOG¥REQ.LOG Y S
```

「Log =」に続くパラメータとしては、以下のものを空白文字で区切って指定します。

パラメータ順番	説明	例
1	ログファイル名（混乱を避けるため、フルパスで指定するのが良いでしょう）	C:¥TMP¥LOG¥REQ.LOG
2	ファイルの同期の有無を指定します。ログの同期については、2.13「ログの「同期」とは？」を参照してください。	Y（同期あり）
3	ログ出力のレベル：ログの詳細さを設定します。ログ出力のレベルについては、2.12「ログ出力のレベルとは？」を参照してください。	S（サポート）

3.6. インターネット リクエスタ ログの目的と設定は？

インターネット リクエスタ ログは、インターネット リクエスタが MRB や uniPaaS サーバにリクエスト処理を依頼するために行う通信内容を記録するものです。
Web サーバ経由でクライアントから受け取ったリクエストを処理する際に、MRB や uniPaaS サーバとの接続が不安定になってエラーが発生するなど、ネットワークレベルの問題が疑われる場合に収集して確認します。



インターネット リクエスタ ログを出力させるには？

1. インターネットリクエスタのあるスクリプト ディレクトリの位置を確認します。デフォルトでは、uniPaaS をインストールしたディレクトリの直下にある Scripts という名前のサブディレクトリです。
2. そのディレクトリにある MGREQ.INI ファイルを開きます。
3. 「Log = ・・・」パラメータの行を探します。
4. デフォルトでは、行の最初にセミコロン「;」が入っていて、コメントアウトされていますので、セミコロンをはずします。
5. 「Log =」に続けて、ログファイル名、同期の有無、ログレベルを指定します。全体を括弧「(・・・)」でくくってください。
6. ファイルを保存します。
7. 設定を有効にするためには、IIS を再起動する必要があります。

例:

```
[REQUESTER_ENV]
Log = C:¥TMP¥LOG¥INETREQ.LOG Y S
```

「Log =」に続くパラメータは、uniPaaS サーバログの場合と同じです（3.5「uniPaaS サーバログの目的と設定は？」参照）。

ここで指定するログファイル名は、Magic ディレクトリの MGreq.ini あるいは MGrb.ini に指定したのとは別のファイルを指定することをお勧めします。同じファイルを指定しても同期は取られますが、多くのプログラムからのログが混在することになり、ログを読むのが難しくなってしまうからです。

第4章　uniPaaS 製品添付のユーティリティ

本章では、uniPaaS 製品に添付されている各種ユーティリティの目的と使い方について説明します。

種類	出力内容	設定	GUI ツール	CMD ツール
MRB/リクエストの状態	コマンドラインリクエスタ	(コマンドライン引数)		○
	MRB モニタ		○	
ライセンス利用状況 (FlexLM)	FlexLM ユーティリティ	Lmtools	○	
ライセンス利用状況 (uniPaaS)	ライセンス チェック ユーティリティ	mgstations		○

このうち、FlexLM ユーティリティについては、インストールガイドの「トラブルシューティング」に詳しい解説があるので、本書では省略します。

また、ライセンス チェック ユーティリティ (mgstations) については、リファレンスガイド「ユーティリティ → MGSTATIONS」に説明があるので、これも本書では省略します。

4.1. コマンドラインリクエスタの目的と利用方法は？

4.1.1. コマンドラインリクエスタとは？

コマンドラインリクエスタというのは、MRB に対してリクエストを発行するプログラムで、コマンドライン(DOS プロンプト)から実行するので、「コマンドライン」という名がついています。このプログラムは Magic をインストールしたディレクトリにインストールされ、名前は MGRQCMDL.EXE です。

コマンドラインリクエスタを使って、MRB モニタで表示されるのと同様な情報を表示させることができます。一般には、MRB モニタは MRB に大きな負荷を掛ける傾向にあるのに対し、コマンドラインリクエスタは MRB にかける負荷が小さくすることができますので、MRB の状態を監視するには、MRB モニタを使うより、コマンドラインリクエスタを推奨します。

コマンドラインリクエスタが持つ機能には多くのものがありますが、ここでは MRB の情報を取得するための基本的なコマンド使用法について説明します。詳しくは、Magic 添付のマニュアルを参照してください。

4.1.2. 現在実行中の uniPaaS サーバの一覧を見るには？

現在実行中(MRB の管理下にある)Magic アプリケーションサーバの一覧を見るには、「-query=rt」オプションを使います。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus> mgrqcmdl -query=rt

                Enterprise Servers of (MyServer/5215)
  ------------------------------------------------------------------------------
#          EnterpriseServer              Pid        Status        License              Application
==============================================================================================
1|        MyServer/1501  192.168.0.1   3120   Avail Idle    : 0    5    5   , 0 , 51 | MyApp1
2|        MyServer/1797  192.168.0.1   1648   Avail Idle    : 0    5    5   , 0 , 49 | MyApp1
```

この例では、次のことがわかります。

- 二つのエンタープライズサーバが起動している。
- それぞれ MyApp1 というアプリケーションを実行している（Application 欄）

各行は、ひとつのエンタープライズサーバについての情報を表していて、それぞれ、次のことを意味しています。（「値」は例の１行目）。

欄	値	意味
#	1	実行エンジンの番号
Enterprise Server	MyServer/1501 192.168.0.1	実行エンジンが実行されているホスト名と、エンジンの待ち受けポート番号、IP アドレス。
Pid	3120	実行エンジンのプロセス ID。
Status	Avail Idle	実行エンジンの実行状態。
License	0 5 5 , 0 , 51	（後述）
Application	MyApp1	実行エンジンが開いているアプリケーション名。

「License」欄の数字は、5つの数字（上記の例の1番目のサーバでは、「0 5 5 , 0 , 51」）からなっていますが、それぞれ、次のことを意味しています。

欄	値	意味
1	0	現在実行中のスレッド数: 実行中のスレッドはなし。
2	5	ピークスレッド数: 最大 5 スレッドまで実行された。
3	5	ライセンスで割り当てられているスレッド数: 5 スレッドまで割り当てられている。
4	0	利用可スレッド数: 0 の場合には、割り当てられている最大スレッド数まで。
5	51	処理したリクエスト数: 現在までに、51 リクエストを処理した。

4.1.3. リクエストの統計情報を見るには？

リクエストによってマシンに掛かっている統計情報を見るには、「-query=load」オプションを利用します。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus> mgrqcmdl -query=load

                Statistics of (MyServer/5215, 10:51:24)
  ------------------------------------------------------------------------
   Average
  Queue Time      Total        Pending     Executing     Completed    Failed
==========================================================================
      0.09         112            0            0             110          2
```

この例では、次のことがわかります。

欄	値	説明
Queue Time	0.09	リクエストの平均待ち時間は 0.09 秒（すぐに処理されているということなので、かなり余裕がある状態）
Total	112	今までに受け付けたリクエスト数
Pending	0	待ち行列に待たされているリクエスト数はなし。
Executing	0	現在実行中のリクエストはなし。
Completed	110	成功して完了したリクエスト数は 110。
Failed	2	何らかの理由で失敗したリクエスト数は 2。

4.1.4. 個々のリクエストについて表示させるには？

MRB 内に記録されているリクエストを表示するには、「-query=log」オプションを利用します。ここに表示されるリクエストは、処理が完了(成功、失敗を問わず)したもの、処理中のもの、未処理のものすべてが含まれます。

一般にはリクエスト数は何千何万という数になるので、出力するリクエスト数に制限をつけるのが一般的です。ここでは、

- 最近の 20 個のリクエストだけを表示する
- リクエスト ID を特定して、一つのリクエストに関する情報だけを表示させる
- リクエスト ID の範囲を指定して、複数のリクエストに関する情報を表示させる

という場合のやりかたを示します。

例1：最近のリクエスト20個の表示

最近の 20 個のリクエストを表示させるには、単に「-query=log」と指定します。リクエストの古い順に(リクエスト ID の昇順で)表示されます。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus>mgrqcmdl -query=log

                Log of requests  (MyServer/5215, 16:38:46)
  ------------------------------------------------------------------------------

#   Request  Status          Start      Elapsed     Completion Codes
      Id                     Time       (sec.)     (Mri, Runtime, Dbms)
================================================================================
1  | 100  |DONE        | 01/09 16:31:40 | 0    | -OK (0) 0 0
Program  : "e" ("MyApp1")
Priority : 0
Client : MyServer (pid 3576),   EnterpriseServer : MyServer/1501
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
2  | 99   |DONE        | 01/09 16:31:40 | 0    | -OK (0) 0 0
Program  : "e" ("MyApp1")
Priority : 0
Client : MyServer (pid 3576),   EnterpriseServer : MyServer/1797
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
・・・  (以下省略)
```

例2：リクエストIDを指定して表示

調べたいリクエストのリクエスト ID がわかっている場合には、そのリクエスト ID を指定して表示させます。これには、「-query=log=(リクエスト ID)」というパラメータを使います。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus>mgrqcmdl -query=log=132

                Log of requests  (MyServer/5215, 16:52:40)
  ------------------------------------------------------------------------------

#   Request  Status          Start      Elapsed     Completion Codes
      Id                     Time       (sec.)     (Mri, Runtime, Dbms)
================================================================================
1  | 132  |DONE        | 01/09 16:51:13 | 0    | -OK (0) 0 0
Program  : "e" ("MyApp1")
Priority : 0
Client : MyServer (pid 2420),   EnterpriseServer : MyServer/1782
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

例3：リクエストIDの範囲を指定して表示

複数のリクエストを表示させるには、開始と終了のリクエストIDをそれぞれ指定して、「-query=log=(開始リクエストID)-(終了リクエストID)」の形で指定します。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus> mgrqcmdl -query=log=132-142

               Log of requests  (rd2003r2/5215, 16:54:44)
  ------------------------------------------------------------------------

#   Request  Status          Start      Elapsed     Completion Codes
      Id                     Time       (sec.)     (Mri, Runtime, Dbms)
==============================================================================
1  | 132  |DONE        | 01/09 16:51:13 | 0     | -OK (0) 0 0
Program  : "e" ("MyApp1")
Priority : 0
Client : rd2003r2 (pid 2420),   EnterpriseServer : rd2003r2/1782
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
2  | 133  |DONE        | 01/09 16:51:13 | 0     | -OK (0) 0 0
Program  : "e" ("MyApp1")
Priority : 0
Client : rd2003r2 (pid 2420),   EnterpriseServer : rd2003r2/1907
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
・・・   (以下省略)
```

範囲を指定する場合、MRBに大きな負荷が掛からないように、必要最小限として、大きな範囲を指定しないようにしてください。

4.1.5. 待ち行列中のリクエストを表示させるには？

MRBに記録されているリクエストのうち、まだ処理がされていないリクエストを表示させるには、-query=queueを指定します。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus> mgrqcmdl -query=queue

               Requests Queue of (192.168.73.136/5215, 16:59:42)
  ------------------------------------------------------------------------
#    Application     User      Prio Reqid    Client        (pid)     Submit time
==============================================================================
1  | MyApp1                        |        | 0  | 242   | rd2003r2       3612    | 16:59:39
2  | MyApp1                        |        | 0  | 243   | rd2003r2       3612    | 16:59:39
3  | MyApp1                        |        | 0  | 244   | rd2003r2       3612    | 16:59:39
4  | MyApp1                        |        | 0  | 245   | rd2003r2       3612    | 16:59:39
...
```

4.1.6. 実行可能なアプリケーションの一覧を表示させるには？

MRBの配下で実行しているuniPaaSサーバが開いているアプリケーション名の一覧を表示させるには、-query=appを指定します。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus> mgrqcmdl -query=app

              Applications supported by (MyServer/5215)
  ------------------------------------------------------------------------------
       #        Application                    EnterpriseServer
       ========================================================
       1    | MyApp1                          |        MyServer/1501
       2    | MyApp1                          |        MyServer/1594
```

4.1.7. 別マシンの MRB を指定するには？

今までの例では、同一サーバマシン上で実行している MRB に対してコマンドラインリクエスタを使っていました。もし、別マシンに MRB がある場合、あるいは同一サーバ上でも別ポート番号で複数の MRB が動作している場合には、MRB のホスト名・ポート番号を明示的に指定してやる必要があります。

デフォルトの設定（つまり、MRB を明示的に指定しない場合)では、コマンドラインリクエスタは、カレントディレクトリ内にある MGREQ.INI に MessagingServer パラメータで指定されている MRB に対してリクエストを発行します。

例 1: 以下の MGREQ.INI の例は、MRB はホスト名 MyServer のサーバマシン上で、ブローカポート 5215 で実行している場合です。

```
[REQUESTER_ENV]
Gateway = 1
MessagingServer = MyServer/5215
```

MRB が同一の PC 上で実行している場合には、ホスト名およびその直後のスラッシュ文字「/」は省略することができます。

例 2: 以下の例は、MRB が同一の PC 上で、ブローカポート番号 5215 で実行している場合の例です。

```
[REQUESTER_ENV]
Gateway = 1
MessagingServer = 5215
```

もし、MGREQ.INI に指定してある MRB 以外の MRB を指定して、要求を発行するには、「-host=(ホスト名)」および「-port=(ブローカポート番号)」で指定します。

例 3: ホスト名 MyServer、ブローカポート番号 5215 で実行している MRB の配下で実行中のサーバエンジンの一覧を表示させるには、次のようにします。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus> mgrqcmdl -host=MyServer -port=5215 -query=rt

              Enterprise Servers of (MyServer/5215)
  ------------------------------------------------------------------------------

#          EnterpriseServer             Pid        Status        License              Application
=============================================================================================
1|        MyServer/1501  192.168.0.1   3120   Avail Idle    : 0    0    5   , 0 , 0  | MyApp1
2|        MyServer/1797  192.168.0.1   1648   Avail Idle    : 0    0    5   , 0 , 0  | MyApp1

C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus>
```

4.1.8. コマンドラインリクエスタのパラメーター覧を見るには？

コマンドラインリクエスタには、このほかにも多くのパラメータを指定できます。MGRQCMDL.EXE をパラメータなしで起動すると、下記のような簡単なヘルプが出てきます。

コマンドラインリクエスタの詳細な情報については、uniPaaS 製品のリファレンスマニュアルで「分散アプリケーション > アプリケーションパーティショニング > コマンドラインリクエスタ」を参照してください。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus> mgrqcmdl
Command Line Requester, Version uniPaaS 1.8 SP1a-0 12-345-6789

-APPNAME -PRGNAME : Application and program names
[-ARGUMENTS  ]    : Program arguments, separated by commas
[-VARIABLES  ]    : Named variables, separated by commas
[-PRIORITY   ]    : Priority of execution (0 - 9)
[-USERNAME   ]    : Username required by the application
[-PASSWORD   ]    : Password required by the application or by the Broker
[-FILENAME   ]    : Name of a file to contain the request results
[-NOWAIT     ]    : Asynchronous request mode
[-HOST, -PORT]    : Broker's address

Data types : -A : alphanumeric      -U : NULL      -L : logical (True/False)
             -N : integer           -F : float     -D : double

Example : -APPNAME=Pet Shop Demo -PRGNAME=Orders List -PRIORITY=4
-USERNAME=supervisor -PASSWORD=mypass -FILENAME=MGRQCMDL.OUT
-ARGUMENTS=string value,-N1000,-LTRUE,-U
-VARIABLES=var 1=string value,var 2=-N1000,var 3=-LTRUE

-QUERY      : Query requests -
  RT        : [(appname)] Registered Enterprise Servers
  APP       : [(host/port)] Applications supported by Enterprise Server(s)
  CTXS      : (host/port) Contexts supported by Enterprise Server
  QUEUE     : [(appname)] Requests in queue
  LOG       : [(appname)][=reqid[-reqid]] Historic information
  LOAD      : [(appname)] Statistics about application or the Broker
  PENDING   : =<reqid> : Number of requests pending before the request
-REQID      : =<reqid> : Request manipulation (priority, removal)
-CLEAR      : Removing a request from the queue
-EXE        : =<ExeEntry>[/<args>] Activating an executable by the Broker
-TERMINATE  : Termination requests -
  ALL       : all Enterprise Servers, including the Broker
  RTS       : all Enterprise Servers, but not the Broker
  host/port : a specific Enterprise Server
  TIMEOUT   : terminate the engine within this time period (seconds)

Examples : -QUERY=RT      -QUERY=RT(Pet Shop Demo)   -QUERY=APP(my_server/1500)
           -QUERY=QUEUE   -QUERY=LOG=100-90          -QUERY=LOAD(Pet Shop Demo)
           -EXE=Background//StartApplication=1
           -TERMINATE=ALL    -TERMINATE=my_server/1500
           -CLEAR -REQID=1
```

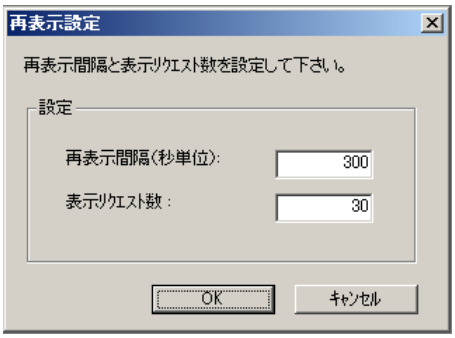
4.2. ブローカモニタ (MRB モニタ)で何ができますか？

ブローカモニタ (MRB モニタ)は、ブローカの状態を監視するためのツールです。前述のコマンドラインリクエスタが、コマンドラインからのツールであったのに対し、ブローカモニタは GUI を備えたツールとなっていて、一定間隔で表示を自動的に更新します

ブローカモニタは便利なツールではありますが、定期的に多くの情報を取得するため、MRB にかかる負担は大きく、運用に支障を来たすことがあります。基本的に開発環境やテスト環境で利用し、運用環境では常時使わないようにしてください。
ブローカモニタを常時利用している際に、サーバで不定期に問題が起こるようなことがあったら、まずはブローカモニタの利用を停止して様子を見てください。

ブローカモニタをどうしても使いたい場合には、次の設定により MRB にかかる負担を軽減することができます。

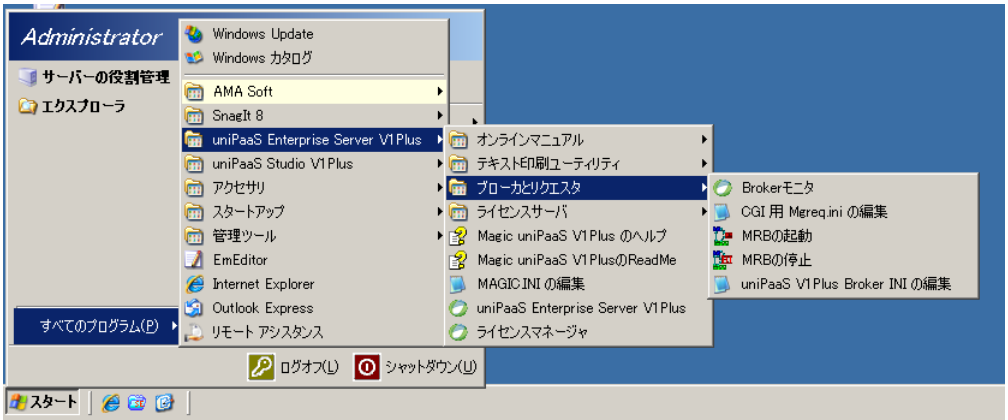
- メニュー「オプション → 再表示設定」から、「再表示間隔」を長くとる（例：300 秒）。また、「表示リクエスト数」を少なめにする（例：30）
- 「コンテキスト」画面を閉じる。



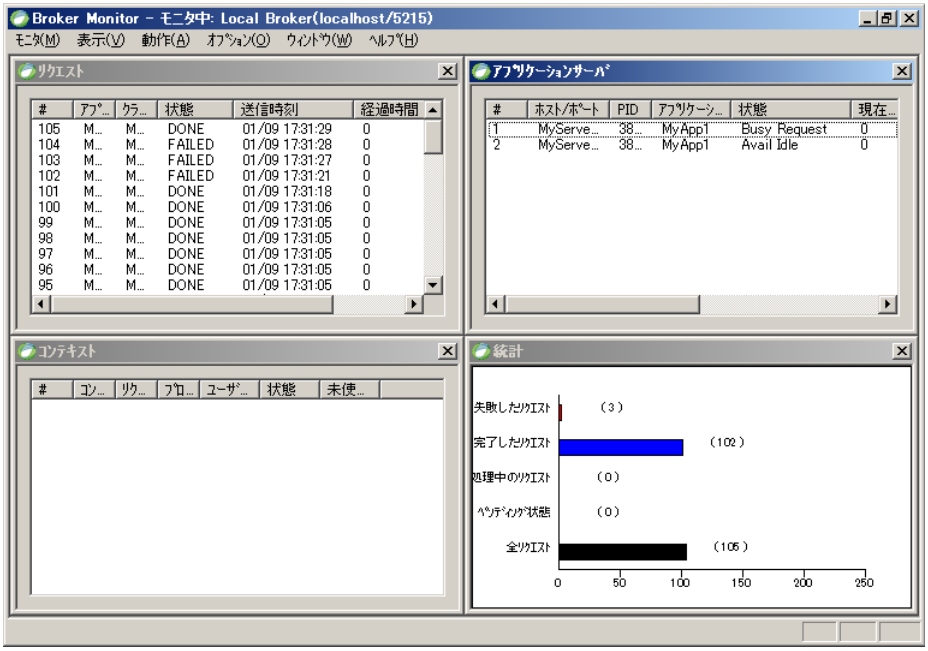
ブローカモニタの使い方の詳細については、リファレンスマニュアル「分散アプリケーション → Broker モニタ」を参照してください。

4.2.1. 起動方法

ブローカモニタは、Windows の「スタート」メニューから「uniPaaS Enterprise Server V1Plus（インストールした uniPaaS 製品名）→ ブローカとリクエスタ → Broker モニタ」を選んで起動します。



起動直後には、次のような画面が表示されます。



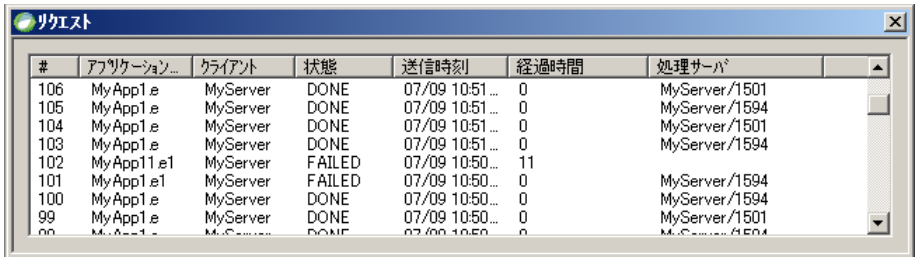
4.2.2. アプリケーションサーバ

現在この MRB の配下で実行中のアプリケーションサーバ（Enterprise Server、あるいは Richclient Server）の一覧を表示します。
コマンドラインリクエスタでの -query=rt に相当します。

4.2.3. リクエスト

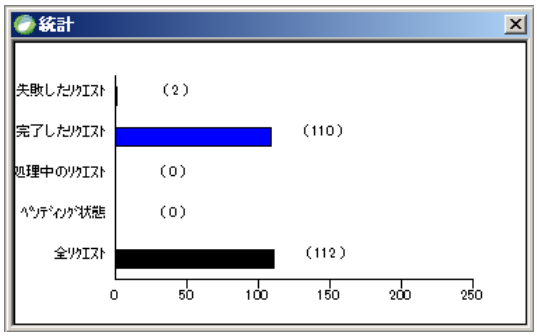
この MRB が受け付けたリクエストの一覧を表示します。

コマンドライアンリクエスタでの -query=log に相当します。



4.2.4. 統計

この MRB が受け付けたリクエストの状態についての統計グラフを表示します。
コマンドラインリクエスタでの -query=load に相当します。



4.2.5. 利用可能アプリケーション

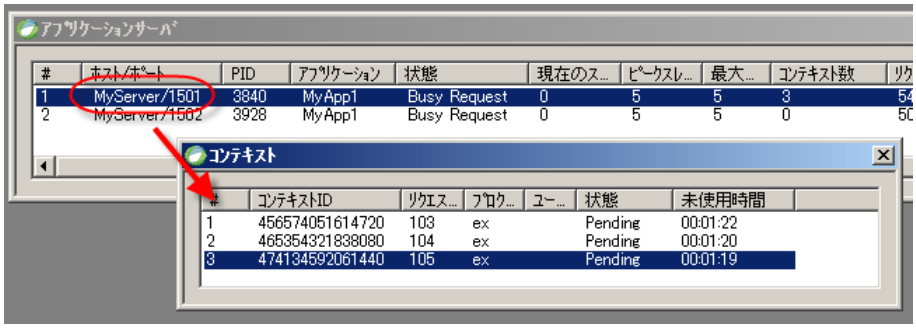
この画面は、「アプリケーションサーバ」画面と連動しています。

「アプリケーション サーバ」画面に表示されている uniPaaS サーバのインスタンスをマウスクリックで選択します。すると、そのアプリケーションサーバで実行されているアプリケーションが「利用可能アプリケーション」画面に表示されます。

コマンドラインリクエスタでの -query=app に相当するものです。

4.2.6. コンテキスト

「コンテキスト」画面も「アプリケーションサーバ」画面と連動しています。
「アプリケーション サーバ」画面に表示されている uniPaaS サーバのインスタンスをマウスクリックで選択します。すると、そのアプリケーションサーバで保持されているコンテキストの一覧が表示されます。



ブローカモニタが MRB に大きな負荷を与えて動作を不安定にすることがあるので、「コンテキスト」画面はできるだけ閉じておくことを推奨します。

第5章　ダンプファイル

ダンプファイルというのは、Windows 上で実行しているプログラムについて、ある特定の時点でのプロセスのメイン メモリの状態と、プロセスに関連する情報をハードディスクに保存したものです。ダンプファイルの種類にもよりますが、プロセスの仮想空間のデータを保存するので、最大数百 MB にもなることがあります。
ダンプファイルは、プログラムの実行に異常（異常終了、ハングアップ、不明なエラーなど）があった場合に、原因を解析するのに非常に有効ですので、uniPaaS システム実行中に問題が起きた場合に、MSJ のサポートセンターからダンプファイルの取得を依頼することがあります。
本章では、ダンプファイルの取得の方法について説明します。

ダンプファイルの解析には、プログラムのソースコードをはじめとして、さまざまな内部情報が必要になるので、ダンプファイルの解析は MSJ において行ないます。uniPaaS アプリケーションの開発者が解析を行うことはできません。

ダンプファイルの作成は、Windows の機能ですので、技術的詳細については、Microsoft 社の Web サイト、あるいはインターネットの検索で「Windows ダンプファイル」などをキーワードとして検索すれば多数見つけることができます。

5.1. ダンプファイルを取得するタイミングと方法は？

ダンプファイルは、プログラムの実行中にある一時点におけるプロセスの状態をファイルに保存するものですが、効果的に解析するには、取得するタイミングが重要です。
一般には、問題が起こった時点でのプロセスの状態をダンプファイルにとります。例えば、

- プログラムが異常終了する。（クラッシュ）
- プログラムが反応しなくなる。（ハングアップ）
- エラーダイアログが表示される。

などです。

ダンプファイルを取得するための方法は、Windows のバージョンにより異なります。

- Windows XP および Windows Server 2003 では、ワトソン博士 を使います。
- Windows Vista、Windows Server 2008、Windows 7 では、タスクマネージャを使います。

以下、(1) 異常終了時にダンプファイルを作成させる方法、(2) ハングアップ時にダンプファイルを作成させる方法、のそれぞれについて、各 Windows バージョンに分けて説明します。

エラーダイアログが表示される場合には、ハングアップの場合と同じ方法により、ダンプファイルを作成することができます。

5.2. サーバモジュールのダンプファイル取得時の注意事項

uniPaaS のサーバ製品 (Enterprise Server、RichClient Server) をマルチスレッドのバックグラウンドモードで実行している場合には、以下のパラメータを MAGIC.INI に設定してください。

```
[MAGIC_SPECIALS]
ExceptionMessageBoxDisplay = Y
```

このパラメータは、uniPaaS 実行エンジンのエラー処理の方法を制御します。

- N の場合 (デフォルト)では、uniPaaS 実行エンジンがマルチスレッドでアプリケーションを実行中に、あるスレッドで実行を継続できなくなるような例外状態 (メモリアクセス違反など) が発生した場合、そのスレッドは終了させるけれども、エンジン自体はリカバリー処理を行って、継続して他のスレッドの処理の実行を続けます。これは、通常の運用時に適当な設定であり、一つのスレッドの異常が他のスレッドの実行に影響を与えないようになり、可用性が向上します。
- Y の場合には、例外状態が発生した場合には、その後の処理は Windows のデフォルトのエラー処理に任せます。メモリアクセス違反などの OS レベルでの例外が起こった場合には、通常、Windows はダンプファイルを作成してそのプロセスを停止します。このとき、同時に実行していた他のスレッドや、そのプロセス(インスタンス)に格納されていたコンテキストなどは、プロセスの終了と同時に中断・破棄されます。このため、運用上の可用性は低くなります。

通常の運用時には N に設定しておくのが適当ですが、この設定では異常事態が発生した場合にリカバリ処理が働いてダンプファイルが作成されず、「ダンプファイルを調査して原因を追求する」という方法が使えなくなってしまいます。このため、デバッグ時に限っては、クラッシュダンプを作成するために、Y に設定して、あえてプロセスを強制終了させるようにする必要があります。
ただし、この設定では、上記のように運用上の可用性が低くなるため、デバッグ時に限り設定するようにして、調査が終わったら、元通り N にする (あるいはパラメータをコメントアウトする)ようにしてください。

5.3. 異常終了時にダンプファイルを作成させるには？（Windows XP および Windows Server 2003 の場合）

Windowx XP および Windows Server 2003 では、ワトソン博士 drwtsn32.exe をデフォルトのデバッガとして Windows に登録しておくことにより、プログラムが異常終了した際に自動的にダンプファイルを作成するように設定しておくことができます。

ワトソン博士をデフォルトのデバッガとして登録するには

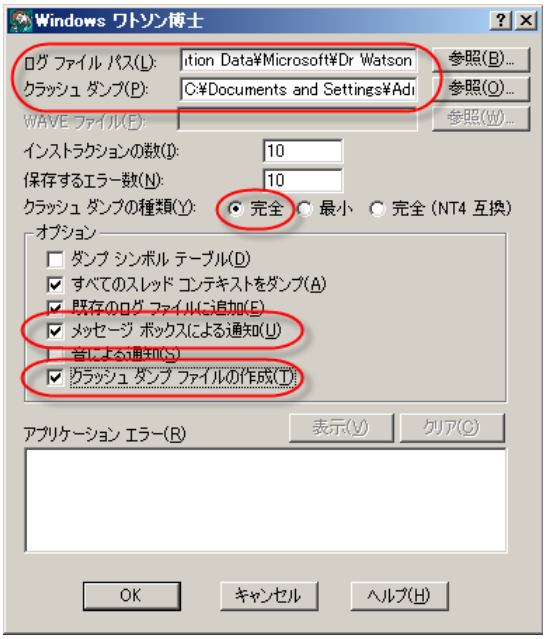
4. Administrator 権限のあるユーザとしてログインします。
5. コマンドプロンプト(DOS 窓)で、「drwtsn32 –i 」とタイプします。右図のようなダイアログが表示されます。

6. コマンドプロンプトから、今度は引数なしで「drwtsn32」を実行すると、右図のような設定ダイアログが表示されます。

   次のような設定を行ってください：

   「ログファイルパス」のファイル名、および「クラッシュダンプ」欄のパス名は、十分な空き容量のあるディスク上にあることを確認してください。

   クラッシュダンプの種類：完全
   メッセージボックスによる通知：チェックする
   クラッシュダンプファイルの作成：チェックする



これで設定は終了です。この設定は Windows のレジストリに登録されますので、一度だけ行えば、その後ずっと有効です。

この設定を行って、実際にプログラムのテスト・運用を行ないます。実行中に、アクセス違反、ハンドルされない例外などが発生してプロセスが異常終了したら、次の手順でダンプファイルを採取してください。

プログラムが異常終了した場合には・・・

1. プログラムが異常終了した場合には、Windows は自動的にワトソン博士を起動し、右図のようなダイアログが表示されます。
   ここのボタンが「キャンセル」になっている間は、ログファイルを作成中ですので、ボタンを押さないでください。

2. しばらくすると（最大1分程度）、ボタンが「キャンセル」から「OK」に変わります。
   ボタンが「キャンセル」から「OK」に変わったら、ログファイルおよびクラッシュダンプファイルの作成は完了ですので、OK を押してワトソン博士を終了させてください。

作成されたダンプファイルは、ワトソン博士の「クラッシュダンプ」欄に指定されたディレクトリに保存されています。デフォルトでは、Windows XP では

C:¥Documents and Settings¥All Users¥Application Data¥Microsoft¥Dr Watson¥user.dmp

あるいは、Windows Server 2003 では

C:¥Documents and Settings¥Administrator¥Local Settings¥Application Data¥Microsoft¥Dr Watson¥user.dmp

です。

5.4. 異常終了時にダンプファイルを作成させるには？（Vista、Windows 7、Windows Server 2008 の場合）

Windows Vista、Windows 7、Windows Server 2008 の場合には、ワトソン博士はなく、レジストリにデフォルトのデバッガを設定します。

ダンプファイル作成の設定を登録するには

1. レジストリエディッタを開いて、次のようにキーを設定します。

キー: HKEY_LOCAL_MACHINE¥Software¥Microsoft¥Windows¥Windows Error Reporting¥LocalDumps		
ダンプファイルを格納する場所	値の名前:	DumpFolder
	種類:	REG_EXPAND_SZ
	値のデータ:	c:¥CrashDumps
上記場所に格納できるダンプファイルの最大数。	値の名前:	DumpCount
	種類:	REG_DWORD
	値のデータ:	0xa
ダンプのレベル	値の名前:	DumpType
	種類:	REG_DWORD
	値のデータ:	0x2

ここで、「ダンプファイルを格納する場所」が c:¥CrashDumps になっていますが、任意の場所で構いません。空き容量が十分にあるハードディスク上のディレクトリを選択してください。

レジストリエディタを使わず、.reg ファイルから登録することもできます。
上の設定のままならば、次のような内容を tmp.reg など（名前は任意で構いませんが、拡張子は .reg とします）に保存してから、ダブルクリックしてレジストリに登録してください。

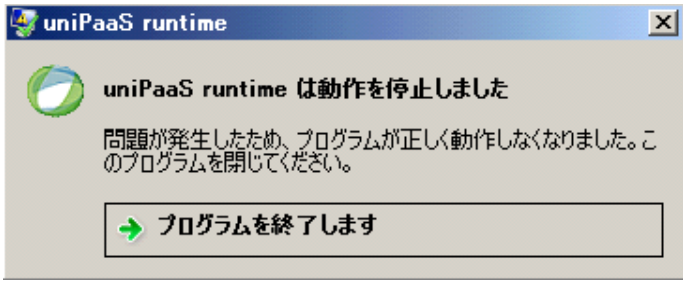
```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\LocalDumps]
"DumpFolder"=hex(2):63,00,3a,00,5c,00,43,00,72,00,61,00,73,00,68,00,44,00,75,\
 00,6d,00,70,00,73,00,00,00
"DumpCount"=dword:0000000a
"DumpType"=dword:00000002
```

設定はこれだけで、すぐに有効になります。

この設定を行ってから、uniPaaS アプリケーションをテスト実行・運用実行します。プログラムが異常終了した場合には、自動的にダンプファイルが作成されます。

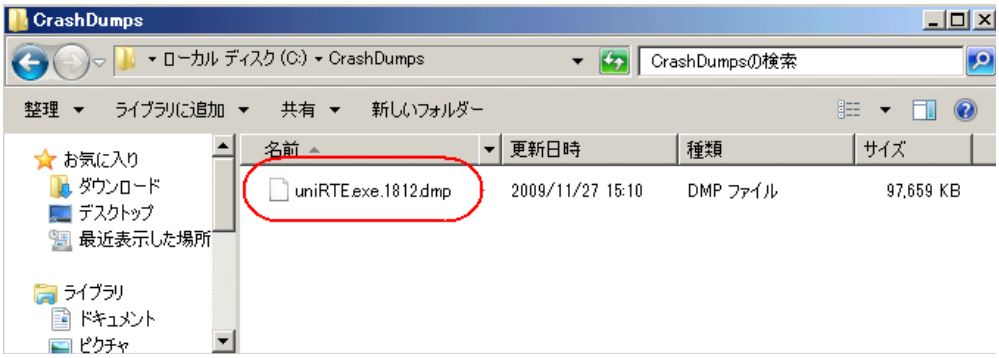
プログラムが異常終了した場合には・・・

1. 例外が起こったら、「・・・は動作を停止しました」というダイアログが出ます。



2. すぐに「プログラムを終了します」ボタンを押します。

ダンプファイルは、レジストリに設定したディレクトリ（今の例では C:¥CrashDumps）に作成されます。



この方法では、自動的にダンプファイルが作成されますが、予めレジストリを設定しておく必要があります。
レジストリの変更をせずに、異常終了時にダンプファイルを作成するには、手作業になりますが、ハングアップの時にダンプファイルを作成する方法を使うことができます（ 5.6「ハングアップ時にダンプファイルを作成させるには？（Vista、Windows 7、Windows Server 2008 の場合）」を参照）

5.5. ハングアップ時にダンプファイルを作成させるには？（Windows XP および Windows Server 2003 の場合）

Windows XP では、ハングアップ時/エラー時にダンプファイルを作成させるには、ダンプを取ろうとするプロセスのプロセス ID を指定してワトソン博士を呼び出します。

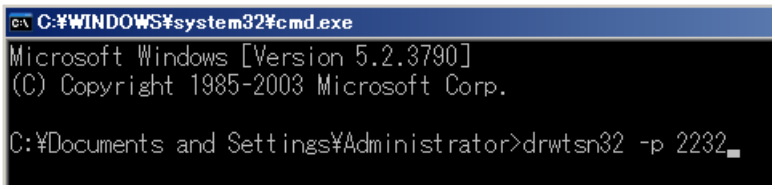
このための前準備として、タスクマネージャでプロセス ID を見つけることができるようにしておく必要があります。この方法は、5.7「タスクマネージャにプロセス ID を表示させるには？」を参照してください。

この設定をしてから、uniPaaS プログラムをテスト実行/運用開始します。
実行中にハングアップが発生した場合には、まず、ダンプをとるべきプロセスを特定する必要があります。プロセスを特定する方法は、5.8「ダンプを取るプロセスのプロセス ID を特定するには？」を参照してください。
ダンプを取るべきプロセスのプロセス ID がわかったら、そのプロセス ID を指定して、ワトソン博士のコマンドにより作成します。

プロセス ID を指定して、ダンプファイルを作成させるには

1. コマンドプロンプト（DOS 窓）を開きます。

2. drwtsn32 -p (プロセス ID) と入力し、Enter キーを押します。
右図は、プロセス ID = 2232 を指定した場合です。



3. 右図のようなダイアログが出るので、しばらく待ちます。

4. しばらくすると(1分以下)、「キャンセル」ボタンが「OK」ボタンに変わります。この後、OK ボタンを押します。
このとき、リッチクライアントの画面も消えるはずです。

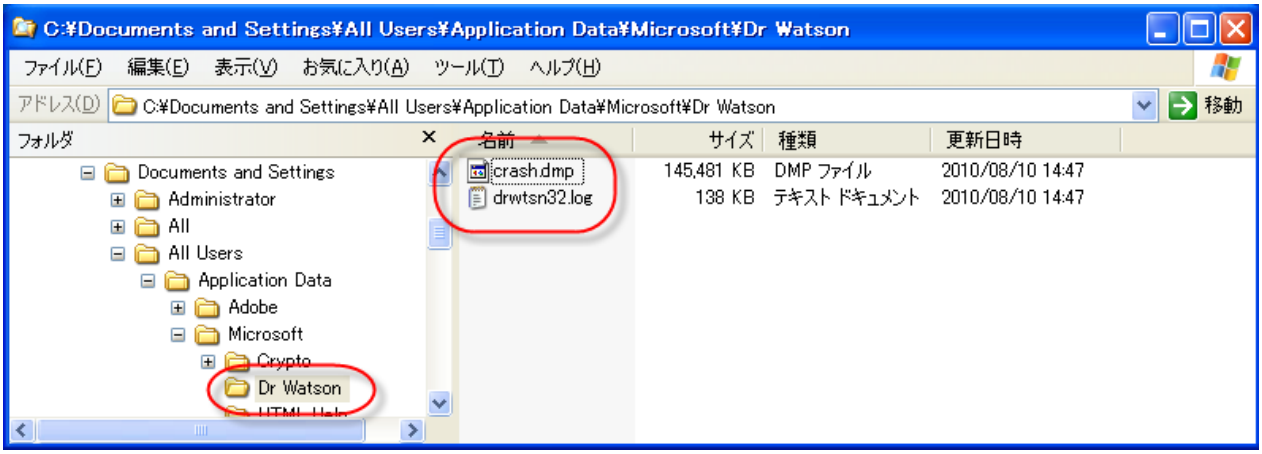
作成されたダンプファイルは、Windows XP の場合には

C:¥Documents and Settings¥All Users¥Application Data¥Microsoft¥Dr Watson

にあり、Windows Server 2003 の場合には

C:¥Documents and Settings¥Administrator¥Local Settings¥Application Data¥Microsoft¥Dr Watson

にあります。



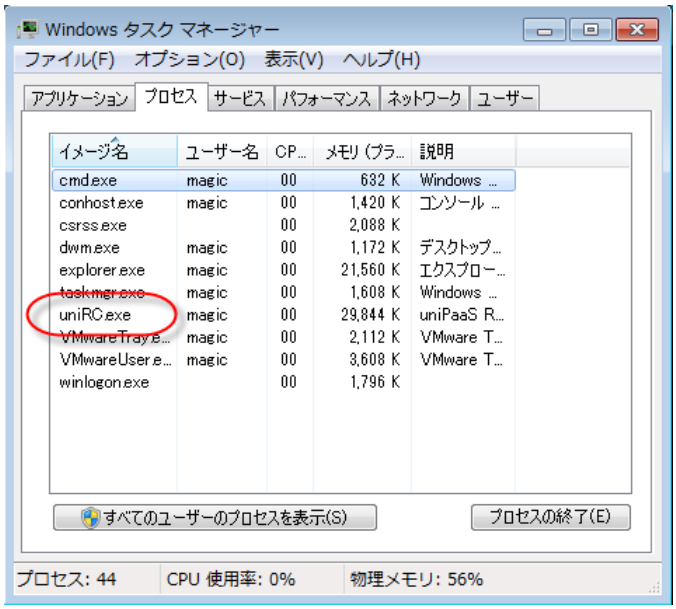
5.6. ハングアップ時にダンプファイルを作成させるには？（Vista、Windows 7、Windows Server 2008 の場合）

Windows XP の場合と異なり、タスクマネージャにプロセス ID を表示させるための前準備は必要ありません。

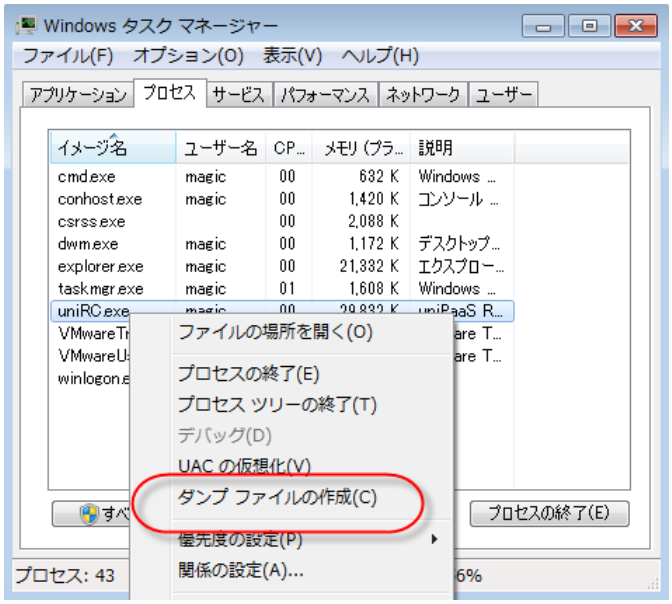
uniPaaS プログラムをテスト実行/運用している途中で、ハングアップが起きた場合には、問題のプロセスを特定する必要があります。これは Windows XP の場合と同様、5.8「ダンプを取るプロセスのプロセス ID を特定するには？」を参照してください。

ダンプを取るべきプロセスが特定できたら、次の手順でダンプファイルを作成します。

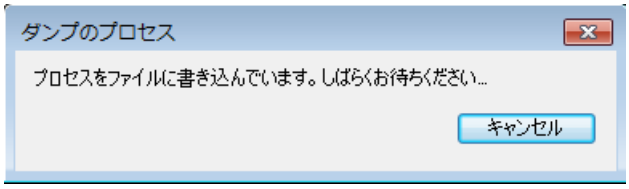
1. タスクマネージャで、ダンプを取るプロセスにカーソルを置きます。



2. 右クリックしてメニューを開き、「ダンプ ファイルの作成(C)」を選びます。

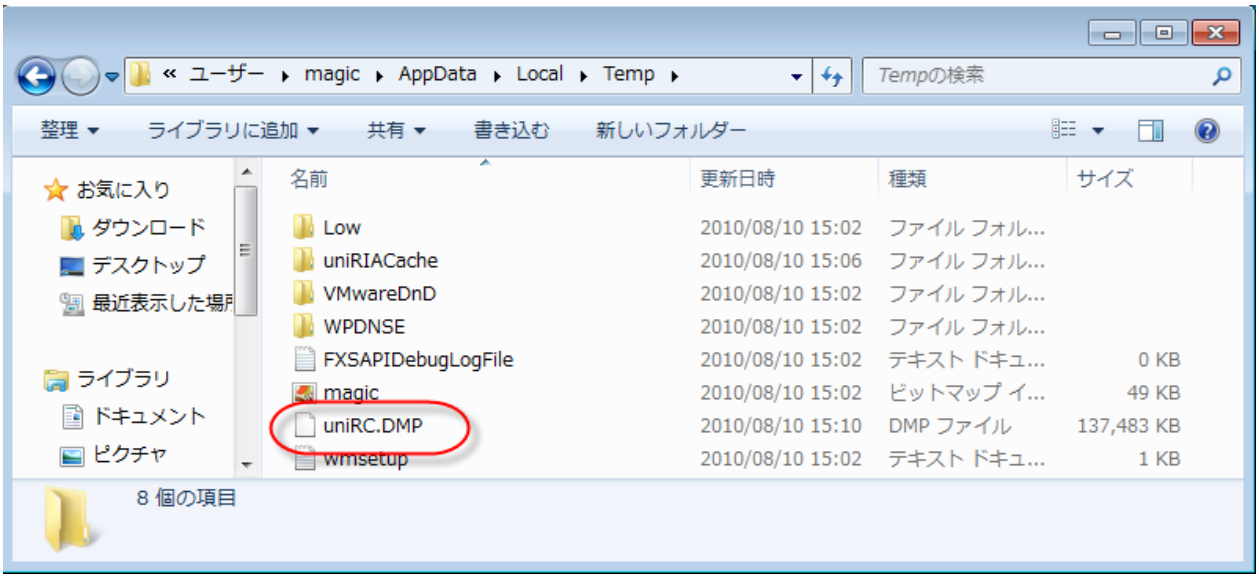


3. ダンプを書き込んでいる旨のダイアログが出て来ますので、しばらく待ってください。（通常 1 分以内）

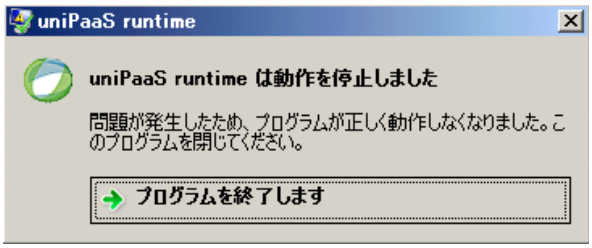


4. しばらくすると、右図のような画面になります。ファイル名を控えてから、「OK」ボタンを押します。

ダンプファイルは、上記のダイアログに示された場所にあります。



ここに説明した、タスクマネージャからダンプファイルを作成する方法は、異常終了時にダンプファイルを作成させる場合にも使うことができます。
異常終了時には、右図のようなダイアログが出ますので、このダイアログが出たら、タスクマネージャを開き、上記の手順でダンプファイルを作成できます。その後、「プログラムを終了します」ボタンを押します。

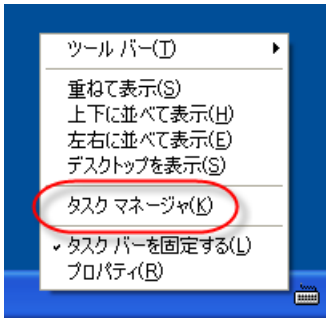


この方法を使うと、毎回手作業でダンプファイルを作成する必要がありますが、5.4「異常終了時にダンプファイルを作成させるには？（Vista、Windows 7、Windows Server 2008 の場合）」に説明したような、レジストリの変更を伴なう前準備は必要ありません。

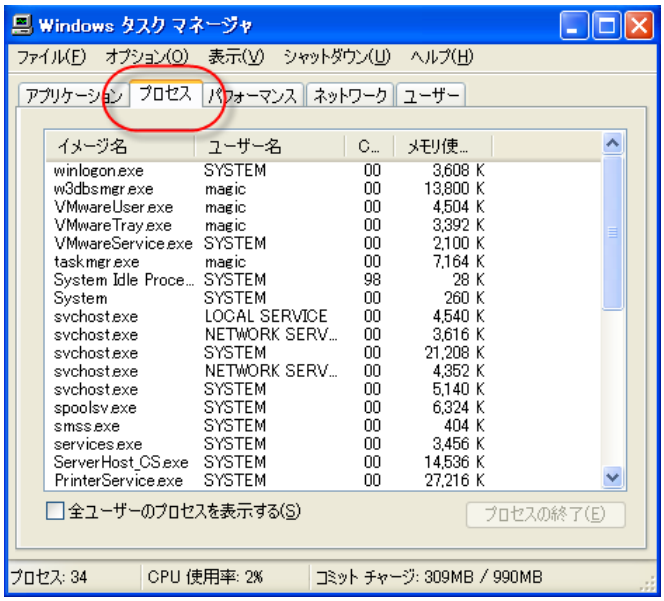
5.7. タスクマネージャにプロセス ID を表示させるには？

タスクマネージャにプロセス ID を表示させるには
（前準備として、一度だけ実行します）

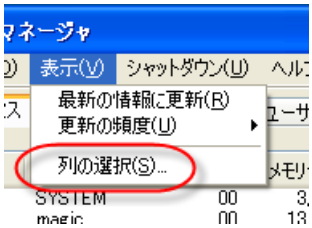
1. タスクバーから、タスクマネージャを起動します。



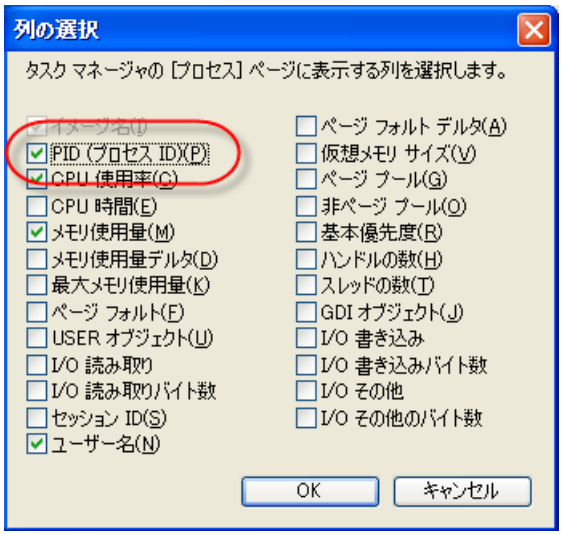
2. 「プロセス」タブを開きます。



3. メニュー「表示 → 列の選択」を選びます。

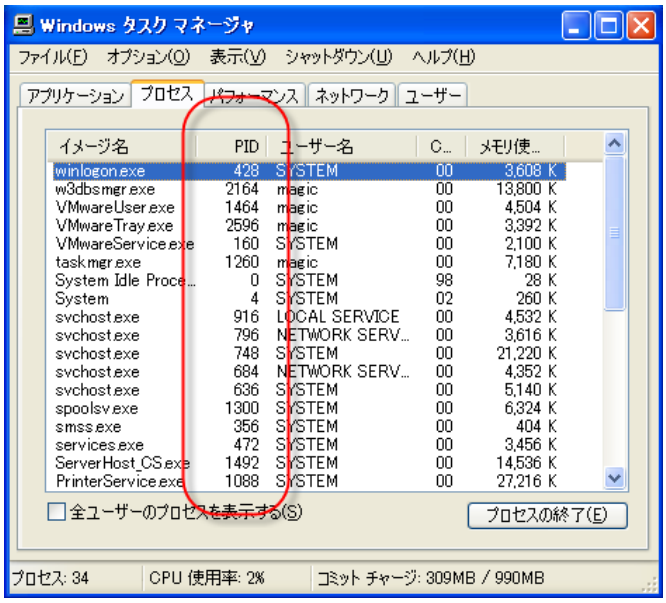


4. 「PID (プロセス ID) にチェックを入れ、「OK」を押します。



5. 「PID」という欄が追加されたことを確認します。

   この後、タスクマネージャは閉じてかまいません。



タスクマネージャの設定は、レジストリに保存されるので、この操作は一度実行すれば、その後ずっと有効になります。

5.8. ダンプを取るプロセスのプロセス ID を特定するには？

ハングアップを起こしているプロセスを特定するには、タスクマネージャのプロセスの一覧から状況をみて判断する必要があります。プロセスを特定したら、タスクマネージャの「PDI」欄からプロセス ID を知ることができます。

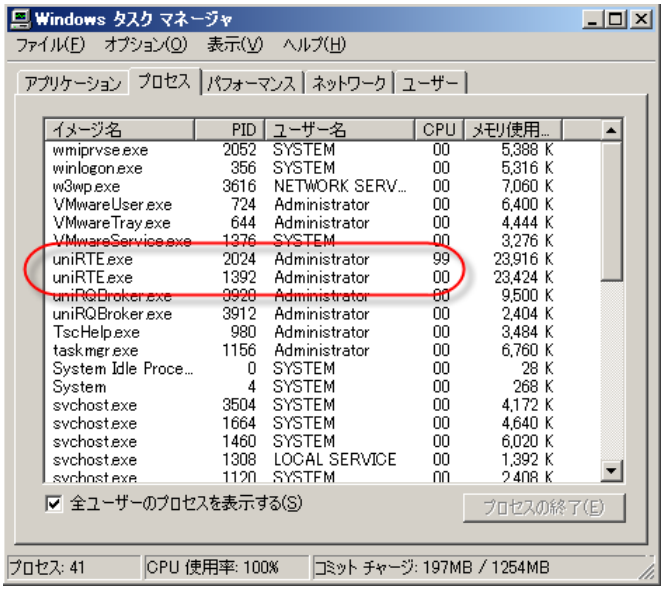
プロセスを特定するには、最初に「イメージ名」により、区別します。タスクマネージャの「イメージ名」欄には、プロセスの実行可能ファイル名が表示されます。uniPaaS 製品のモジュールでは、次のような名前になっています。

製品	イメージ名
uniPaaS Studio	uniStudio.exe
uniPaaS 実行エンジン（Client、Enterprise Server、RichClient Server）	uniRTE.exe
リッチクライアントのクライアントモジュール	uniRC.exe
MRB	uniRQBroker.exe

例えば、Enterprise Server を使っているシステムで、サーバエンジンが動作を停止してしまっているようであれば、「イメージ名」が「uniRTE.exe」であるものを探します。

タスクマネージャに表示されているプロセスで、イメージ名が一致するものが一つしかなかった場合には、これだけでタスクを特定できますが、イメージ名が一致するプロセスが複数ある場合には、その中から真犯人を区別しなければなりません。これには確実な方法はなく、プロセスの挙動を見て推測する必要があります。

- MRB（uniRQBroker.exe）の場合、プロセスは二つありますが、区別は簡単です。一つは監視用のプロセスであり、メモリ使用量が少なく、もう一つは実際のブローカとしての仕事を行っているプロセスであり、メモリ使用量は多いです。ですので、メモリ使用量が大きいものを選べばよいということになります。
- 実行エンジン（uniRTE.exe）の場合には、「CPU」欄を見て推測します。ハングアップしているプロセスは、CPU が 100%近くになっているか（無限ループに陥っている場合）、あるいはいつまでたっても 0% のままか（待ち状態が永遠に続いている場合など）のいずれかになっています。そのようなプロセスがあれば、それを選びます。
  右図の例では、プロセス ID が 2024 である uniRTE.exe が CPU 99% となっているので、これを選びます。



もし状況から判別するのが難しいようであれば、該当するプロセスのすべてについて、ダンプファイルを作成するのが一番確実です。

第6章　uniPaaS デバッガ

デバッガとは、アプリケーションのエラーを修正するために、開発者がアプリケーションを実行しながら、実行エンジンの状態やフローの流れなどを確認したり、手動で強制的に変更したりすることのできる、スタジオの機能です。uniPaaS では、開発用のスタジオと、実行用のエンジンが分離されているため、デバッガが非常に使いやすく、かつ堅牢になっています。
本章では、以下のような uniPaaS でのデバッガの基本機能について解説します。

- ブレイクポイントを設定する。
- ブレイク時にタスクの項目の値を確認する。
- ブレイクポイントに条件を設定する。
- ウォッチリストを作成し、実行時に特定の項目の値を確認する。
- 実行時に項目の値を手動で変更する。
- スタジオ内で、実行フローを確認する。
- コールスタックを見る。
- 複数のコンテキストの実行状況を見る。

本章の内容については、以下の技術文書に基本的な事項についての説明がありますのでご参照ください。本書の内容は、主に、これらの補足説明をするものです。

デバッガー般	「リファレンスマニュアル」 アプリケーションのテスト → デバッガ
デバッガの使い方	「マスタリング Magic uniPaaS」 第 29 章　アプリケーションのデバッグ

重要：RichClient プログラムのデバッガについて：

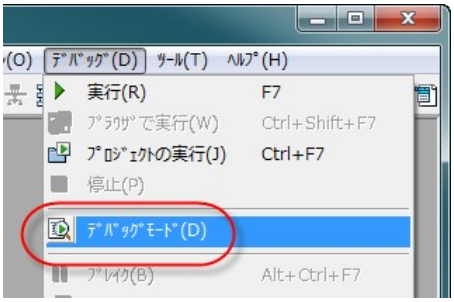
- uniPaaS V1Plus Ver1.8 の RichClient プログラムは、デバッガに対応していません。このため、RichClient プログラムを実行中にデバッガを使って、ブレイクをかけたり項目をウォッチしたりすることができません。この制限は将来のバージョンで対応される予定です。
- アクティビティモニタは RichClient プログラムでも対応しています。このため、RichClient プログラムの実行の様子をアクティビティモニタで追っていくことは可能です。アクティビティモニタはリモートデバッグでも利用することができます。(7.4「アクティビティモニタを表示するためだけに使うには？」を参照)

6.1. デバッガを開始する

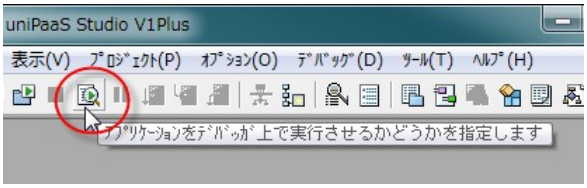
デバッガを利用するには、Magic uniPaaS Studio が **デバッグモード** で動作している必要があります。

デバッグモードに入るには・・・

デバッグモード は、メニュー **デバッグ(D)** から **デバッグモード(D)** を選択します。



デバッグモードは、ツールバーで設定することもできます。



デバッグモードを解除するには・・・

デバッグモード を解除するには、同じく **デバッグ(D)** メニューから **デバッグモード(D)** を再度選択します。
このメニューは、選択するたびにデバッグモードがトグルします。

6.2. ブレイクポイント

ブレイクポイントは、プログラム中に開発者が設定するもので、プログラムが実行時にその位置に到達したら、エンジンはプログラムの実行を一時的に中断(ブレイク)し、開発者がそのときのプログラムの状態を調査することができるようになります。

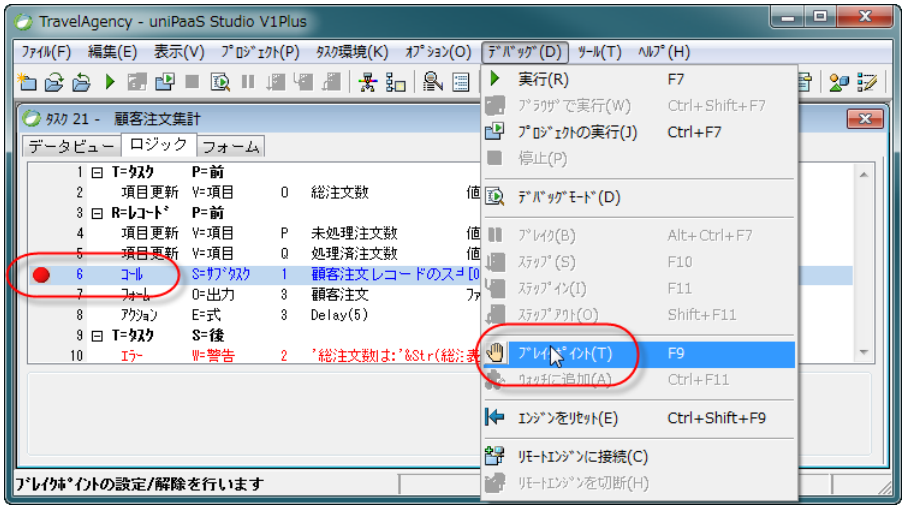
6.2.1. ブレイクポイントを追加する

Magic uniPaaS でのブレイクポイントは、プログラムリポジトリ中の任意のタスクのデータビューエディタ、あるいはロジックエディタ上の行に設定することができます。ただし、コメント行には設定することはできません。

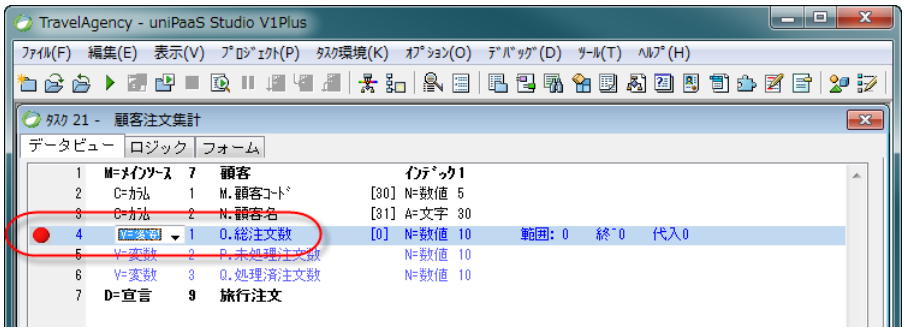
ブレイクポイントを設定するには・・・

1. データビューエディタ、あるいはロジックエディタを開きます。
2. ブレイクポイントを設定したい行にカーソルを置きます。
3. **デバッグ(D)** メニューから、**ブレイクポイント(T)** を選びます。F9 キーで設定することもできます。
   - ブレイクポイントが設定された行には、左側に赤色の ● 印が表示されます。
   - ブレイクポイントの設定は、**デバッグ(D)→ ブレイクポイント(T)** メニュー (あるいは **F9** キー)によりトグルします。

右図は、ロジックエディタにブレイクポイントが設定された例です。

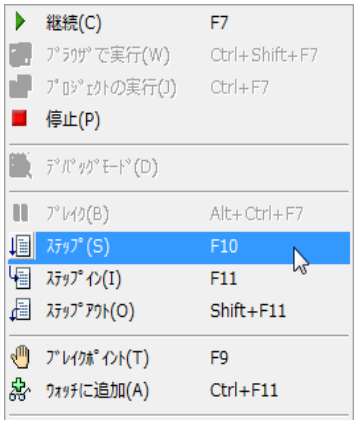


右図は、データビューエディタにブレイクポイントが設定された例です。



ブレイクポイントの設定は、プロジェクトファイルに記録されます。このため、プロジェクトを閉じた後に再度開いたときにも、前に設定されたブレイクポイントはそのまま有効になっています。

6.2.2. プログラムを実行する



ブレイクポイントの設定されているプログラムを実行すると、次のときにプログラムの実行がブレイク(一時中断)します。

- データビューエディタに設定されているブレイクポイントの場合には、ブレイクポイントの設定されている行で定義されている項目の値に変更があったとき。
- ロジックエディタにブレイクポイントが設定されている場合には、ブレイクポイントの設定されている行まで実行が進んだとき。

ブレイクポイントから実行を再開するには、**デバッグ(D)** メニューから、次のようなオプションを選択できます。

メニュー	キー	意味
継続	F7	実行を継続する。
停止		プログラムを強制的に終了する。
ステップ	F10	現在の行を実行し、次の行で再度ブレイクする。
ステップイン	F11	ステップ と同じだが、現在の行がコールコマンドやイベント実行コマンドの場合、あるいは開発者定義関数を含む場合には、それにより呼ばれるプログラム、タスク、イベントハンドラ、関数などの最初の行まで進んだところで再度ブレイクする。
ステップアウト	Shift+F11	現在実行中のタスク、ハンドラなどを最後まで実行し、呼出元に戻った時点で再度ブレイクする。

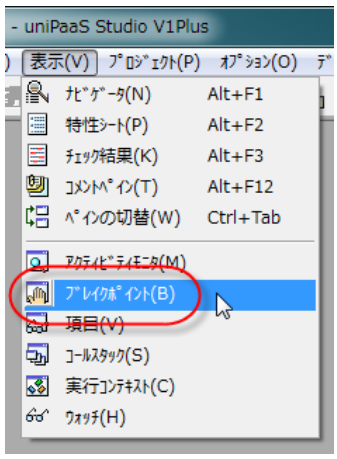
6.2.3. 実行中に強制的にブレイクする

プログラムの実行に予想以上に時間がかかる場合とか、期待していた動作でない場合には、強制的にブレイクをかけて、現在のプログラムの状態を調査したい場合があります。
このような場合には、プログラム実行中に、**デバッグ(D)** メニューから、**ブレイク(B)** を選択します。その時点で強制的にブレイクが起こり、スタジオでは、実行中のタスクが自動的に開いて、以下のように、現在実行中の行にカーソルが移動します。

- オンラインタスクでユーザのデータ入力待ちの場合には、データビューエディタが開き、現在フォーカスのある項目に対応する **C=カラム** あるいは **V=変数** などの行のカーソルが移動します。
- その他の場合(バッチタスクの実行中、あるいはオンラインタスクでもタスク前・後処理やレコード前・後処理実行中の場合など)には、ロジックエディタ上で、現在実行中の行にカーソルが移動します。

6.2.4. ブレイクポイントリポジトリ

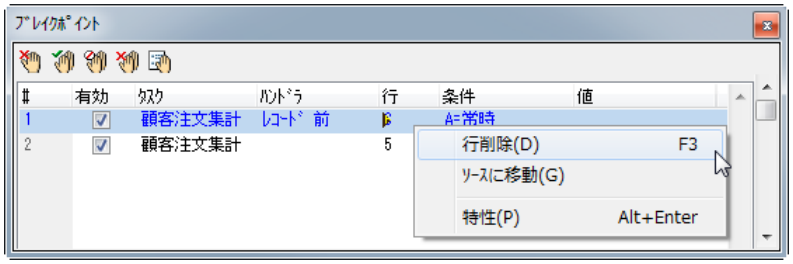
設定したブレイクポイントは、**ブレイクポイント リポジトリ** で一覧表示できます。



ブレイクポイント リポジトリは、**表示(V)** メニューから、**ブレイクポイント(B)** を選択すると表示されます。
ここには、設定されたブレイクポイントが一覧で表示されます。

有効 欄は、ブレイクポイントの有効性を示すチェックボックスです。開発者はチェックボックスをクリックすることにより、設定をトグルさせることができます。**有効** にチェックが入っていないブレイクポイントは一時的に無効となったブレイクポイントで、設定された行まで実行が進んでもブレイクはかかりません。

ブレイクポイント リポジトリで、右マウスクリックすると、コンテキストメニューが表示されます。



メニュー項目	意味
削除(D)	ブレイクポイントを削除します。
ソースに移動(G)	ブレイクポイントの設定されているタスクが開き、その行にカーソルが移動します。
特性(P)	ブレイクが起こる詳細な条件を設定することができます。詳しくは、後述の「条件付ブレイクポイント」で説明します。

ブレイクポイントオプション

ブレイクポイントリポジトリにはツールバーがあり、次のような操作を行うことができます。

アイコン	対応するメニュー	機能
	削除(D)	現在カーソルのある行のブレイクポイントが削除されます。
		現在設定されているすべてのブレイクポイントが削除されます。
		すべてのブレイクポイントが有効化されます。

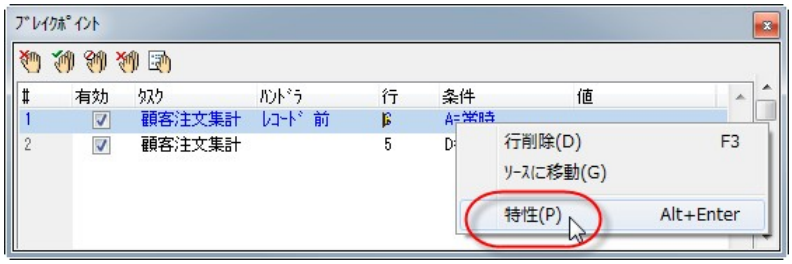
		すべてのブレイクポイントが無効化されます。
	特性(P)	ブレイクが起こる詳細な条件を設定することができます。詳しくは、後述の「条件付ブレイクポイント」で説明します。

条件付ブレイクポイント

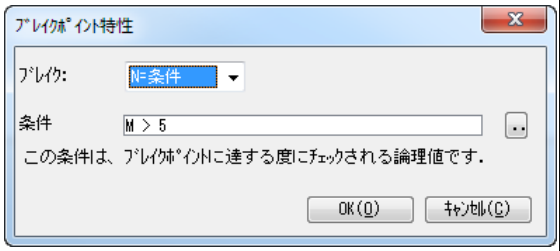
ブレイクポイントリポジトリでは、各ブレイクポイントに対し、ブレイクする条件を設定することができます。

ブレイクポイントの条件を設定するには・・・

1. ブレイクポイントリポジトリで、条件を設定したいブレイクポイントにカーソルを合わせます。
2. 右マウスボタンをクリックし、コンテキストメニューから **特性(P)** を選びます。



3. **ブレイクポイント特性** ダイアログが開きますので、ここでブレイクの条件を、次のオプションから選んで設定します。

ブレイク 値	意味	ダイアログイメージ
A=常時	(デフォルト) ブレイクポイントに達したら常にブレイクします。	
C=カウント	**回数** 欄に設定された回数だけ、このブレイクポイントを通過するたびにブレイクします。	
N=条件	**条件** 欄に式を設定し、ブレイクポイントに達したときに、この式を評価した結果、真になった場合にのみブレイクします。 **条件** 欄に設定するには、プログラムリポジトリで、ブレイクポイントの設定されているタスクを開いておく必要があります。その状態で、**条件** 欄からズームして、式テーブルで設定します。	

条件は、ロジックビューで定義されたブレイクポイントに対してだけ設定することができます。データビューに設定されたブレイクポイントには、条件を設定することはできません。

6.3. 行の無効化

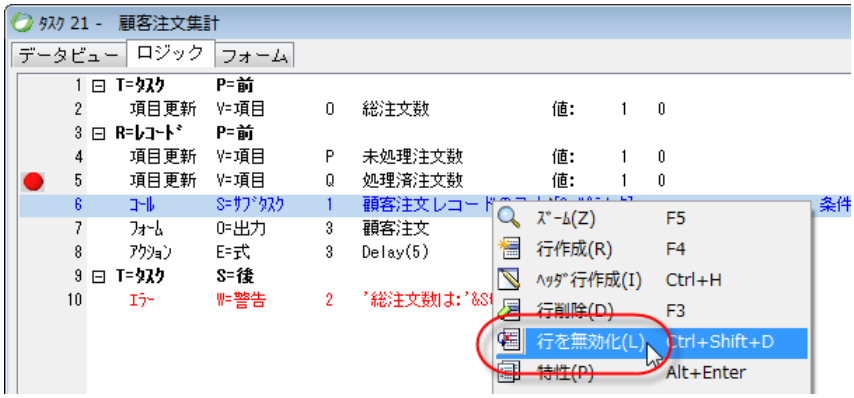
デバッグを行っている段階で、コールコマンドや項目更新コマンドなどを一時的に実行しないようにして、結果がどう違ってくるかを確認したい場合があります。また、実行結果に影響を与えないと分かっているプログラムの実行を一時的に省略したいときがあります。

このような場合、今までは処理コマンドの「条件」欄を No に設定したり、あるいは「デバッグフラグ論理変数」のようなものを定義しておいて、コマンドの「条件」欄にその変数を設定したりしていました。しかし、これはプログラムの変更を伴なう作業であり、デバッグが終了した後に設定を戻し忘れたりする可能性がありました。

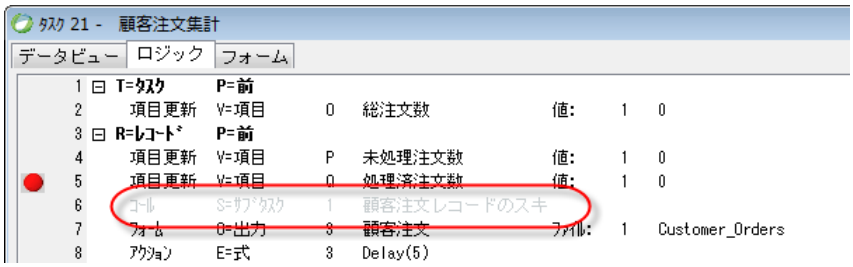
uniPaaS V1Plus では、デバッガで実行中に、特定の処理コマンドを一時的に無効化する機能があります。この機能はデバッグ途中でダイナミックに設定・解除することができるので、いちいちプログラムを変更する必要がなくなりました。

行を無効化するには

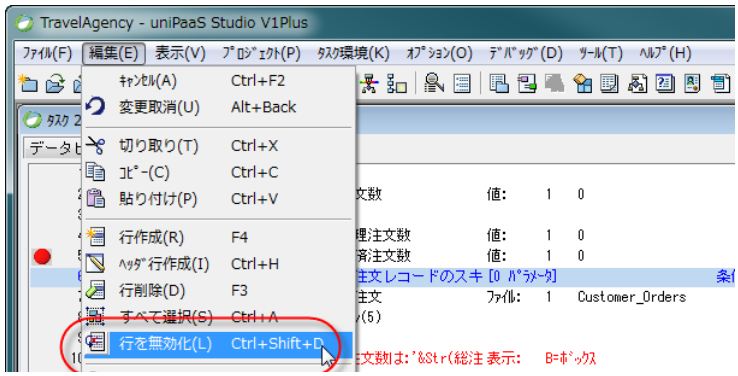
6. プログラムリポジトリを開き、無効化したい行にカーソルを置きます。
7. 右マウスクリックでコンテキストメニューを開き、「行を無効化」を選択します。



無効化された行は、無効化されていることがわかるように、薄い灰色で表示されます。



無効化は、メニュー「編集 → 行を無効化」を選択しても行えます。



無効化された行を元に戻すには、無効化の場合と同様に、コンテキストメニューから「行を有効化」を選択するか、メニュー「編集 → 行を有効化」を選択します。

6.4. ブレイクポイントと行の無効化についての注意

グレイクポイントやウォッチリストなどのデバッグ設定情報は、プロジェクトファイルには保存されません。プロジェクトの .EDP ファイルと同じディレクトリにある .OPT ファイルに保存されます。この .OPT ファイルは次のような性質があります。

- プロジェクトに必須のものではありません。Studio がプロジェクトををオープンする際に見つからなければ、Studio が自動的に作成します。
- ブレイクポイントやウォッチリストなど、デバッガの設定情報を保存します。
- .ECF ファイルやリポジトリ出力ファイルには、この情報は反映されません。このため、(1) ECF ファイルで実行時には影響がありません、(2) リポジトリ出力・入力でプロジェクトを移行すると、デバッグ情報は消えます。
- .OPT ファイルを削除すると、デバッグ設定情報はクリアされます。

一方、行の無効化の設定情報は、プロジェクトファイル (各プログラムに対応した、Sources¥Prg_xxxx.xml ファイル) に記録されます。このため、

- 無効化されている状態で ECF ファイルを作成し実行すると、実行時にも無効化されたままとなります。
- プロジェクトをコピー (リポジトリ出力・入力) しても、無効化情報はそのまま移行されます。

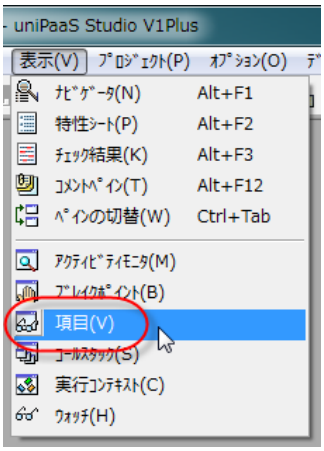
デバッグが終了した時には、無効化されている行を元に戻すことを忘れないように注意してください。

6.5. 項目一覧

項目 オプションでは、現在実行中のコンテキスト中の、項目の値を調べ、必要に応じて強制的に変更することができます。

6.5.1. 項目一覧の表示

項目 一覧は、**表示(V)** メニューから **項目(V)** を選択して表示させます。



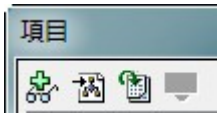
右図に示すように、現在有効な項目名、型、データソース名、現在の値が表示されます。

項目 一覧は、プログラムがブレイクしている状態でだけ有効です。

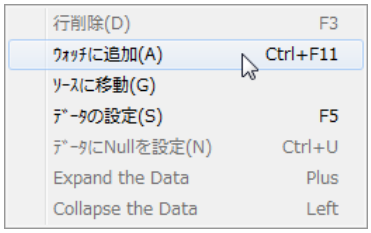
6.5.2. 項目一覧のオプション

項目一覧画面では、次のようなオプションが、右マウスボタンのコンテキストメニュー、あるいはツールバーから操作することができます。

ツールバー

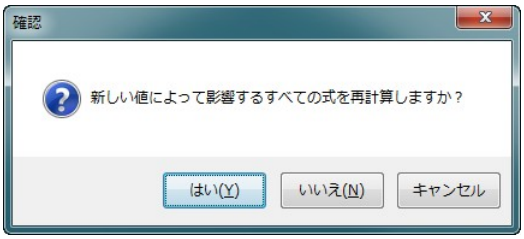


コンテキストメニュー



アイコン	メニュー	意味
	ウォッチに追加	この変数を、ウォッチリスト に追加します。ウォッチリスト については、**6.6 ウォッチリスト** を参照してください。
	ソースに移動	この変数が定義されている、データビューエディタの行に移動します。
	データの設定	この項目の値を手動で変更します。
	データに Null を設定	この項目の値を、Null に設定します。

データの設定(S) あるいは **データに Null を設定(N)** によって項目の値が変更された場合には、「新しい値によって影響する全ての式を再計算しますか？」という確認ダイアログが表示されます。「はい」と答えたら再計算ルールに従って再計算が起こり、「いいえ」ならば再計算が起こりません。「キャンセル」と答えると、変更した内容が元の値に戻されます。



6.6. ウォッチリスト

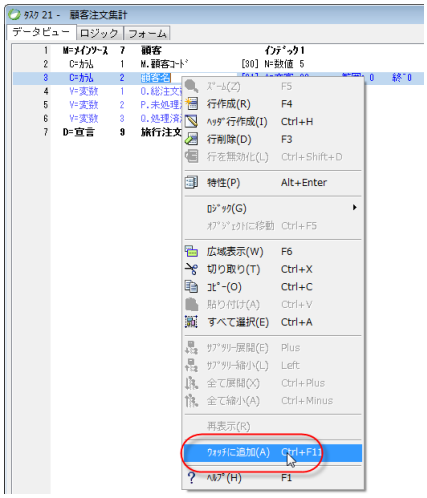
ウォッチリスト は、項目一覧と同様、項目の現在の値を調査したり設定したりするものですが、項目一覧の場合、現在有効な項目がすべて表示されるのに対し、ウォッチリストは、開発者が指定した項目だけを表示するところが異なります。

6.6.1. 項目をウォッチに追加する

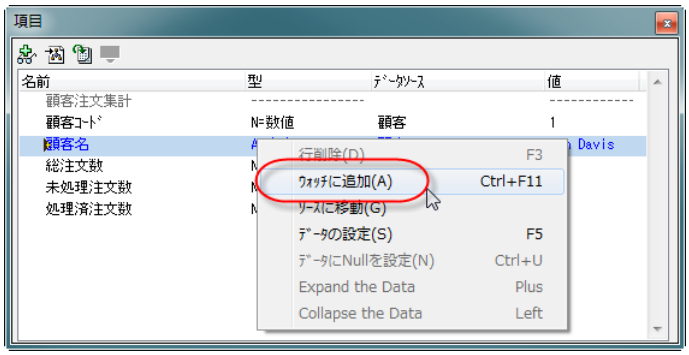
ウォッチリストは、初期状態では空になっています。

ウォッチリストに項目を追加するには、次のいずれかによります。

- プログラムリポジトリのデータビューエディタで項目定義をしている行にカーソルを置き、右マウスボタンでコンテキストメニューを出して、**ウォッチに追加(A)** を選びます。



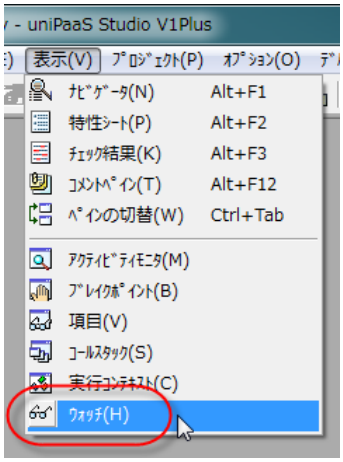
- 項目一覧 から、コンテキストメニューで **ウォッチに追加(A)** を選択します。



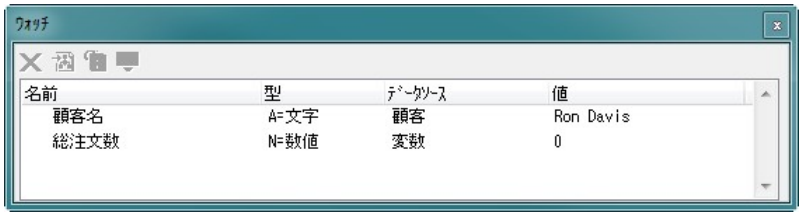
ウォッチリストでは、項目一覧と同じように、**ソースに移動**、**データの設定**、**データに Null を設定**、などの操作を、ツールバー、あるいはコンテキストメニューから行うことができます。

6.6.2. ウォッチリストの表示

ウォッチリストは、**表示(V)** メニューから **ウォッチ(H)** を選んで表示させます。



ウォッチリストには、項目一覧と同じように、項目名、型、データソース名、および、現在の値が表示されます。



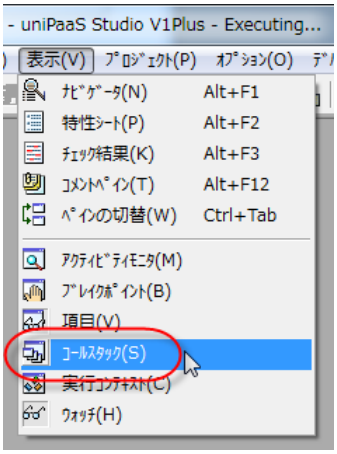
項目一覧の場合と同様、これらの表示は、プログラムがブレイクしている場合にのみ有効です。

6.7. コールスタック

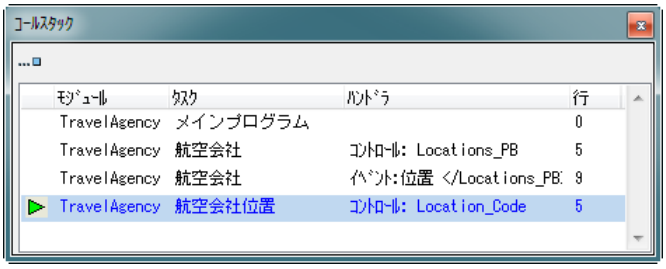
コールスタックは、メインプログラムから現在実行中のプログラムまでの、呼び出しの経路を表示します。

6.7.1. コールスタックの表示

コールスタックは、**表示(V)** メニューから **コールスタック(S)** を選んで表示させます。



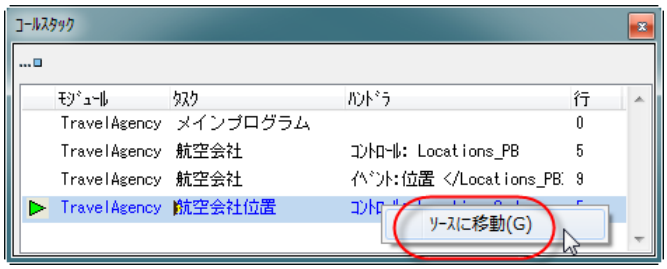
コールスタック ウィンドウには、モジュール名(プロジェクト名)、タスク名、ハンドラ名、およびそのハンドラ中での行数が表示されます。

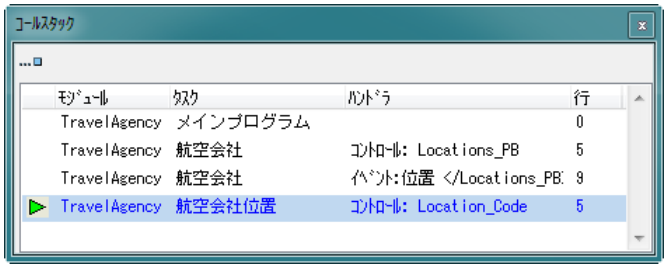


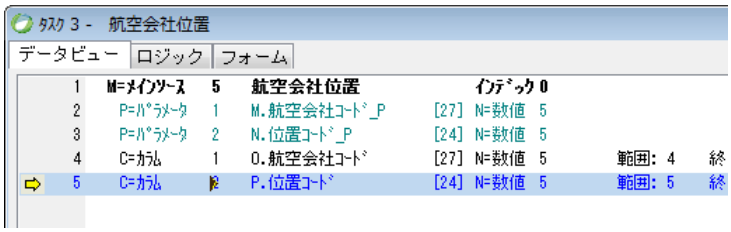
ここでの表示は、プログラムがブレイクされて、実行が中断している状態でのみ有効です。

6.7.2. コールスタックのオプション

コールスタックウィンドウでは、コンテキストメニューから **ソースに移動(G)** を選ぶと、そのタスクで現在実行中の行を、プログラムリポジトリ上で表示します。







オンラインプログラムで、ユーザからのデータ入力待ちの状態のとき、あるいは、データビュー上でカラムや変数にブレイクポイントが設定されている場合にブレイクがかかったときには、**ソースに移動(G)** を行うと、データビューエディタが開き、その項目を定義している行が表示されます。

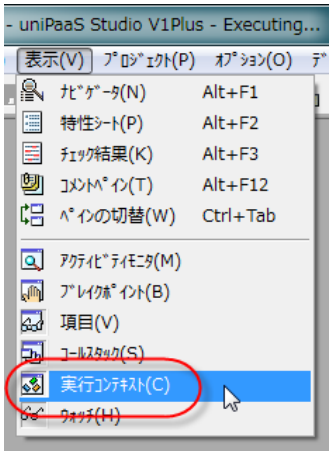
ロジックエディタ設定されているブレイクポイントでブレイクしたときには、**ソースの移動(G)** を行うと、ロジックエディタが開き、ブレイクの起こった行が表示されます。

6.8. 実行コンテキスト

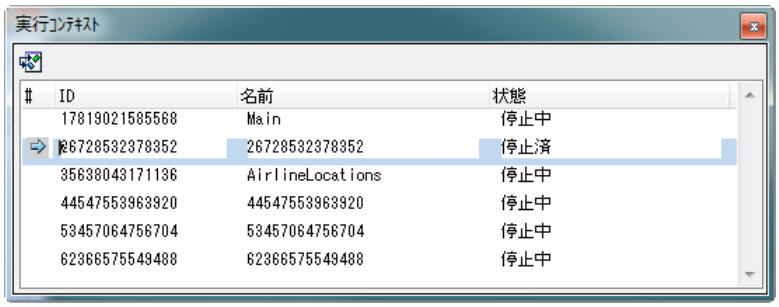
uniPaaS の実行エンジンはマルチスレッドで並行実行をサポートしています。
並行実行を使わないアプリケーションならば、実行コンテキストは一つなので、コンテキストの選択について気を使う必要はありませんが、並行実行を使うアプリケーションでは、複数のコンテキストが同時に実行されていることがあります。
今まで見てきたコールスタック、項目一覧などは、各コンテキスト毎に独立して存在するものなので、正しい値を調査するには、正しいコンテキストを選択することが必要です。

6.8.1. 実行コンテキストの表示

実行コンテキストは、**表示(V)** メニューから、**実行コンテキスト(C)** を選択して表示させます。



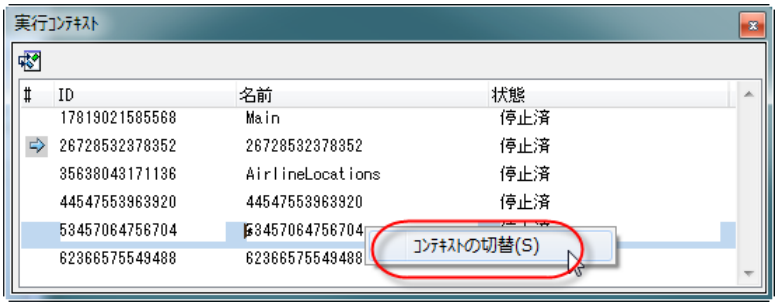
実行コンテキスト ウィンドウには、コンテキスト ID (Magic 実行エンジンが内部で管理している番号)、コンテキスト名 (CtxSetName 関数で設定された名前)、およびコンテキストの状態が表示されます。
右図は、TravelAgency プロジェクトを実行して、いくつかの並行実行プログラムを起動した状態で、ブレイクをかけたときの **実行コンテキスト** ウィンドウの例です。



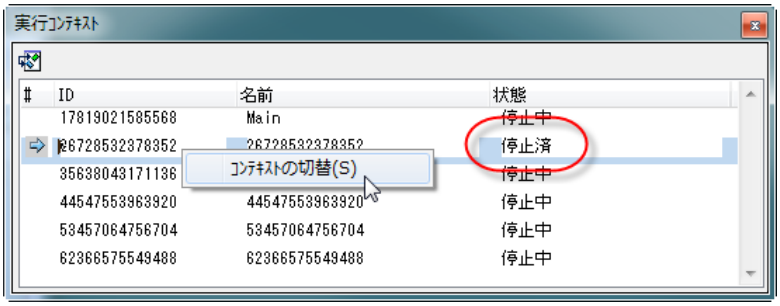
この中で、# の欄に矢印  が表示されている行がありますが、これは現在選択されているコンテキストを表しています。項目一覧、ウォッチリスト、コールスタックなどの表示は、現在選択されているコンテキストのものが表示されます。

6.8.2. コンテキスト切替え

選択されているコンテキストを切り替えるには、実行コンテキスト ウィンドウのコンテキストメニューあるいはツールバーから、**コンテキストの切り替え(S)** を選択します。コンテキストを切り替えると、項目一覧、ウォッチリスト、コールスタックなどの表示が更新され、新しく選択されたコンテキストについての情報が表示されるようになります。



コンテキストを切り替えるには、状態が **停止済** になっている必要があります。状態が **停止中** になっている場合には、いったん、**停止済** 状態にあるコンテキストに対して、**コンテキストの切り替え(S)** をおこなってください。停止可能な状態であれば、**停止中** が **停止済** に変更されます。



第7章　リモートデバッガ

リモートデバッガは、uniPaaS Studio 製品のデバッガ機能の一部として実装されていて、運用環境で、Client あるいは サーバ (Enterprise Server、RichClient Server) 製品を実行している環境でデバッグを行うための機能です。

開発が終了し運用開始した後に、問題が発生した場合、問題が運用環境でだけしか起こらないとか、問題の再現が非常に難しい、ということがよくあります。このような場合には、運用環境でのデバッグが必要になってくることがあります。しかし、運用環境では Studio 製品が動いているわけではないので、前章で説明したような、Studio 製品を使ってテスト実行しながらデバッグすることができません。
このような状況のために、uniPaaS では、通常の運用環境で運用をしながら、別 PC にある Studio を使ってデバッグを行う、という方法をサポートしています。これを「リモートデバッガ」と呼びます。

本章では、リモートデバッガのしくみと使い方について説明します。

リモートデバッガについては、リファレンスマニュアルの「アプリケーションのテスト > デバッガ > リモートデバッガ」の項も参照してください。

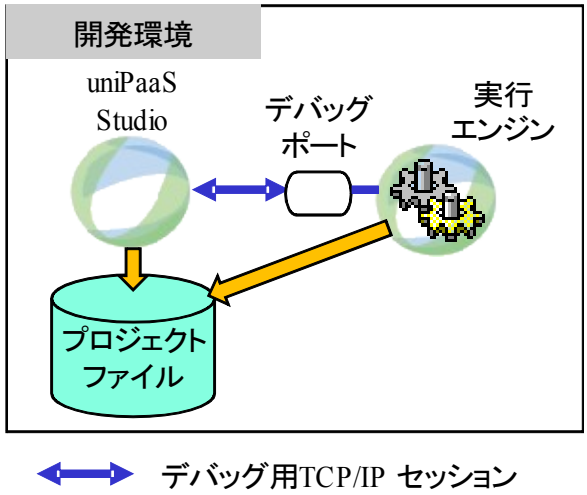
7.1. リモートデバッガのしくみはどうなっていますか？

前章で説明した通常のデバッガは、開発環境で Studio 製品を使って、プロジェクトを実行しているときの機能でした。リモートデバッガについて説明する前に、通常のデバッグ環境における動作をより詳しく見ていきましょう。

7.1.1. ローカルデバッグのしくみ

Studio を起動すると、開発環境としての Studio のプロセス (uniStudio.exe) の他に、背後では、テスト実行用に実行エンジン (uniRTE.exe) のプロセスが立ち上がっています。実行エンジンのプロセスは、開発状態では隠れていて表には現れませんが、F7 でテスト実行を行うときに表示されます。
テスト実行の際には、Studio と 実行エンジンの間で TCP/IP による通信セッションが作成され、デバッグの情報 (ブレイクポイント、項目の値、アクティビティモニタの出力、その他の制御信号)がそのセッションを通してやりとりされています。この時に使われる TCP/IP のポートを「デバッグポート」と呼びます。
通常のデバッグでは、デバッグポートは uniPaaS が自動的に決定するので、開発者はデバッグポートについて気にする必要はありません。

通常のデバッグ状態

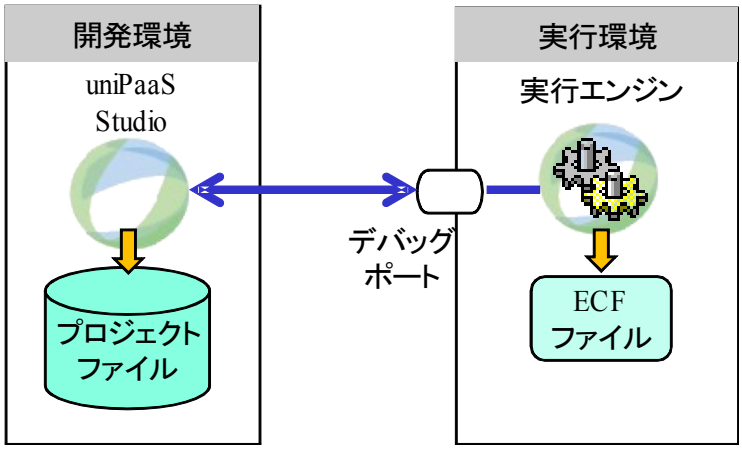


なお、通常のデバッグの場合には、Studio はプロジェクトファイルを編集し、実行エンジンは、プロジェクトファイルを直接実行します。

7.1.2. リモートデバッグのしくみ

リモートデバッグでは、このうちの実行エンジンが別マシン上で実行されるような形となります。ここでは、次のようになります。
実行環境では、実行エンジンだけが動作しており、実行エンジンはプロジェクトのキャビネットファイル (ECF ファイル)を実行しています。
実行エンジンはデバッグポートを開いており、ECF ファイルの実行を行うと並行して、リモートデバッグで Studio から接続されるのを待機しています。
開発環境は、実行環境とは異なる PC 上で、Studio を起動しています。ここには、ECF ファイルを作るもととなった プロジェクトファイル (EDP ファイルおよびその Souces ディレクトリ下のファイル) があります。

リモートデバッグのしくみ



Studio はプロジェクトを開き、実行エンジンのデバッグポートに対して、デバッグ用の TCP/IP 接続を行ないます。その後は、通常のデバッグの場合と同じように、Studio と実行エンジンとがデバッグ情報をやりとりしながら、実行を進めていきます。

7.2. リモートデバッグにはどんな準備が必要ですか？

リモートデバッグを行うには、アプリケーションを実行している実行環境の他に、次の準備が必要です。

- 開発環境用の PC： Studio をインストールして利用するのに十分な性能を持った PC を1台用意します。この PC は、実行環境の LAN に接続可能で、TCP/IP が正しく設定されている必要があります。
- Studio 製品： その PC に uniPaaS Studio をインストールし、ライセンスを正しく設定します。
- プロジェクト： 実行している ECF ファイルのもととなったプロジェクトファイル (EDP ファイルおよびその Sources ディレクトリ下にある全 XML ファイル)を、開発環境 PC にコピーします。Studio で正常にプロジェクトを開けることを確認しておいてください。このプロジェクトは、ソースファイルを参照するためだけのものですので、開発環境 PC 上で実行できるように設定しておく必要はありません。
- ファイアウォールの設定： リモートデバッガは TCP/IP の通信を行うので、実行環境の側で、ファイアウォールのポートを開けておく必要があります。(7.3 「リモートデバッグはどのように設定しますか？」参照)。
- デバッグユーザの ID とパスワード： セキュリティのために、リモートデバッグを行うためには、uniPaaS のセキュリティ機能を用いたユーザ認証が必要です(7.6 「リモートデバッグのセキュリティは？」参照)。このために、リモートデバッガ権利を持つユーザの ID とパスワードを知っている必要があります。

7.3. リモートデバッグはどのように設定しますか？

7.3.1. デバッグポートの決定

最初に、デバッグ用の TCP/IP 通信に使う、デバッグポートを決めます。デバッグポートの番号について特に制限はありません。未使用のポートのなかから適当に選びます。

ここでは、説明の例のために、4040 番を利用するようにします。

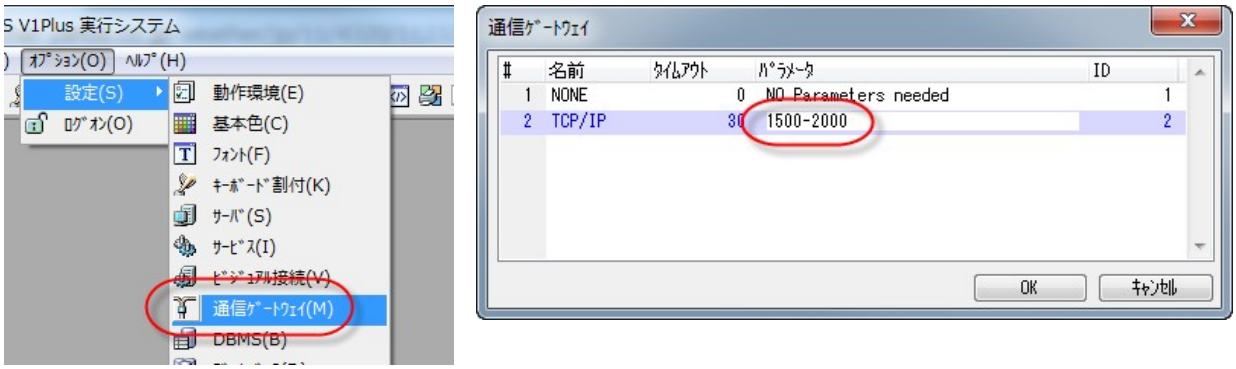
エンタープライズサーバやリッチクライアントサーバなどで、複数のインスタンスを起動する場合には、個々のインスタンスごとに異なるデバッグポートを割り当てる必要があります。同一デバッグポートを指定して複数のインスタンスを起動すると、実行時のエラーやハングアップが起こることがあります。

7.3.2. ファイアウォールの設定

リモートデバッグでは、TCP/IP を通して、実行エンジンと Studio とが通信するので、実行環境の側のファイアウォールで、次のポートを開いておく必要があります。

ポート	説明	例
デバッグポート	未使用のポートのなかから適当に選びます。	4040
通信 GW ポート	「通信ゲートウェイ」テーブルの TCP/IP のエントリで、「パラメータ」欄に定義されたポート番号の範囲。	1500-2000

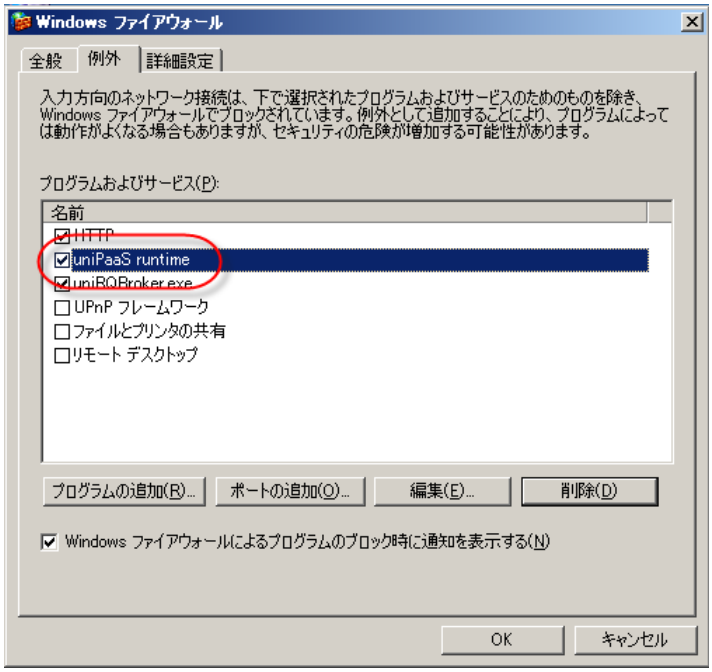
「通信ゲートウェイ」テーブルは、実行エンジンをフォアグラウンドモードで開いて、メニュー「オプション(O) → 設定 (S) → 通信ゲートウェイ (M)」で開くことができます。



また、MAGIC.INI ファイルでは、[MAGIC_COMM] セクションの TCP/IP の行に設定があります。

```
[MAGIC_COMMS]
NONE = 1, 0, NO Parameters needed
TCP/IP = 2, 30, 1500-2000
```

Windows ファイアウォールの場合には、例外のプログラムとして実行エンジン uniRTE.exe が登録されていれば、上記のようにポートを別途開く必要はありません。

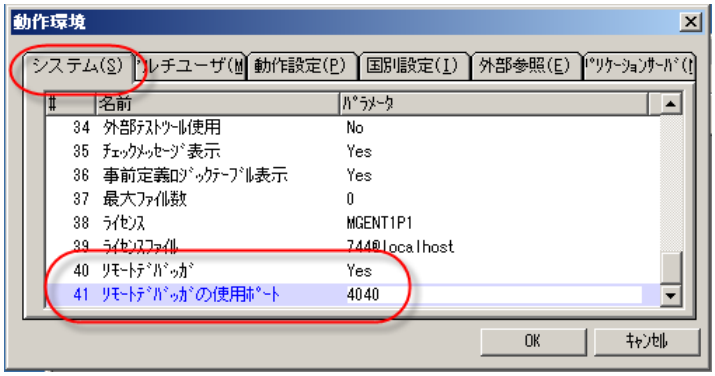


7.3.3. 実行エンジンの設定

実行エンジンでは、デバッグポートを開いて、Studio からの接続を待機するように、環境設定を行ないます。

デバッグ用に実行エンジンを設定するには

8. 実行エンジンを起動して、メニュー「オプション (O) → 設定(S) → 動作環境 (E)」を開きます。
9. 「システム(S)」タブを開きます。
10. 「リモートデバッガ」パラメータを Yes に、「リモートデバッガの使用ポート」として、先に決定したデバッグポート番号を指定します。この例では 4040 としています。



この設定は、MAGIC.INI の [MAGIC_ENV] セクションでも設定できます。

```
[MAGIC_ENV]
EnableRemoteDebugger = Y
RemoteDebuggerPort = 4040
```

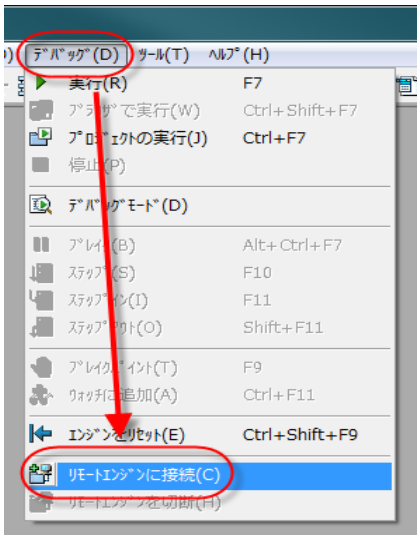
以上で、実行エンジン（サーバ）側の設定は終わりです。設定を有効にするには、実行エンジンの再起動が必要です。

7.3.4. Studio からの接続

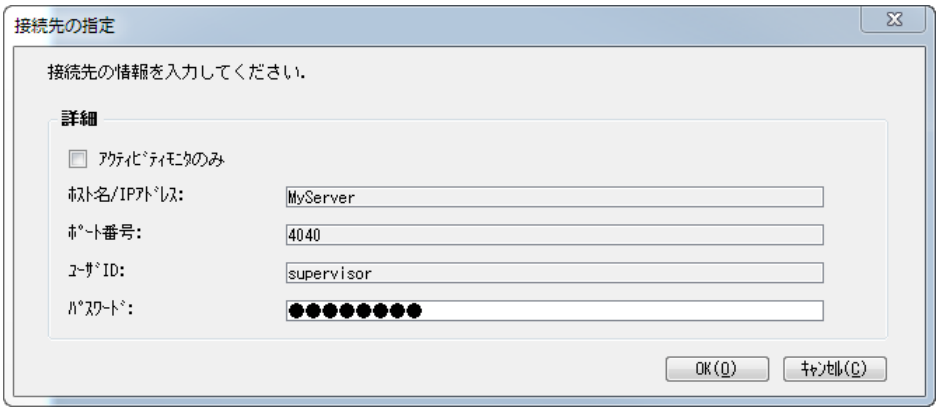
開発用 PC では、Studio から実行エンジンに接続を行ないます。

Studio から実行エンジンにリモート接続するには

1. 実行エンジンを起動した後に、開発用 PC で Studio を起動します。
2. 実行エンジンで実行しているアプリケーションの ECF ファイルのもととなった、プロジェクトファイル（EDP ファイル）を開きます。
3. メニュー「デバッグ (D) → リモートエンジンに接続(C)」を選択します。



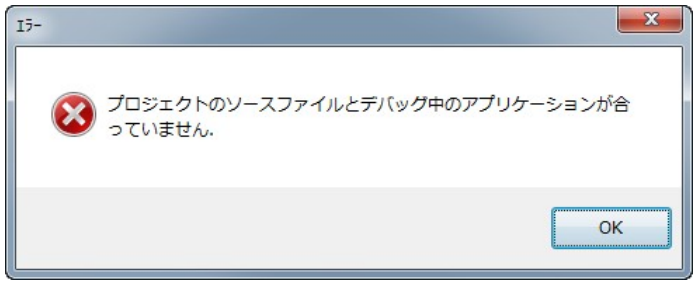
4. 「接続先の指定」ダイアログで、実行エンジンのあるサーバマシンのホスト名（あるいは IP アドレス）、デバッグポート番号、および接続権限のあるユーザ ID とパスワードを指定します。



接続時のユーザ ID/パスワードは、デバッグ時のセキュリティの確保のために指定する必要があります。デバッグ時のセキュリティについて特に設定していない場合には、supervisor として接続すれば良いでしょう。デバッグ時のセキュリティについては、7.6「リモートデバッグのセキュリティは？」を参照してください。

5. OK ボタンを押すと、実行エンジンに接続されます。

実行エンジンで実行しているアプリケーションの ECF ファイルと、Studio で開いているプロジェクトの対応が一致していないと、右のようなエラーメッセージが出て、接続に失敗します。プロジェクトと ECF ファイルの対応について確認してから、再接続してください。



7.3.5. デバッグ作業

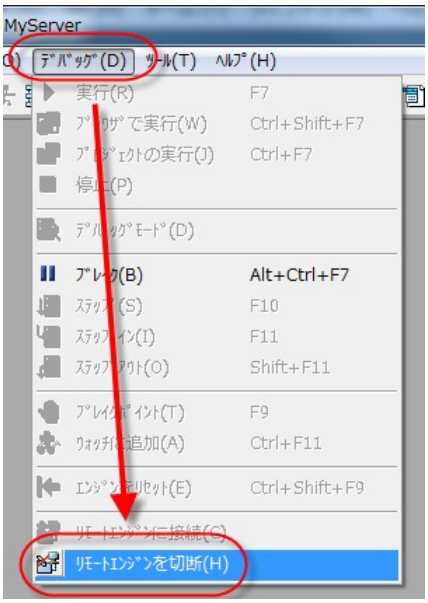
Studio から実行エンジンに正しく接続できれば、デバッグを行うことができます。
リモートデバッグの場合には、ローカルのデバッグの場合と同様、ブレイクポイントの設定、ステップ実行、アクティビティモニタの設定と表示、項目、コールスタック、実行コンテキストの表示などを行うことができます。
リモートデバッグの場合に利用できない機能も一部あります。例えば、リモートデバッグの場合、実行エンジンの停止を Studio から行うことはできません。また、リモートデバッグ中には、プロジェクトはすべて読み込み専用となり、修正することはできません。

7.3.6. 実行エンジンの切断

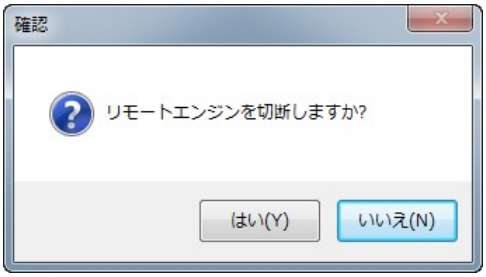
デバッグ作業が終了して、これ以上接続しておく必要がなければ、Studio と実行エンジンの接続を切断します。

デバッガを実行エンジンから切断するには

1. メニュー「デバッグ(D) → リモートエンジンを切断 (H) 」を選択します。



2. 確認画面がでるので、「はい(Y)」を押します。これで、実行エンジンとの接続が切断されます。

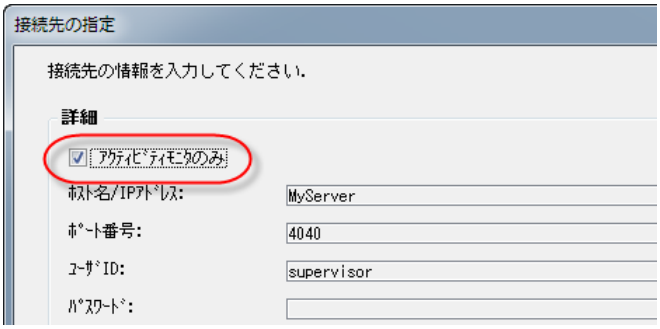


7.4. アクティビティモニタを表示するためだけに使うには？

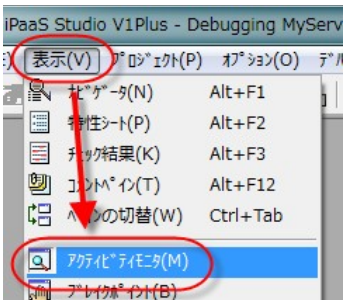
リモートデバッグ機能は、アクティビティモニタの表示のためだけに使う、特別なモードがあります。このモードでは、通常のデバッガとしての機能（ブレイクポイントの設定、項目値の確認や設定、ステップ実行など）は行うことができませが、アクティビティモニタをリアルタイムで見ることができます。また、アクティビティモニタの表示内容の設定も動的に変更することができます。

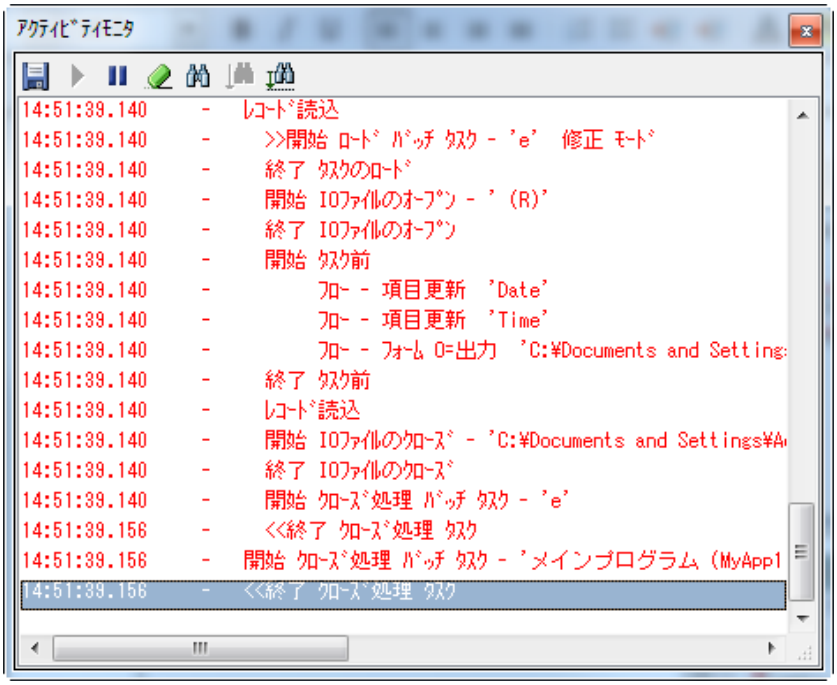
アクティビティモニタのみとしてリモート接続するには

1. メニュー「デバッグ(D) → リモートエンジンに接続 (C)」で接続する際の「接続先の指定」ダイアログで、「アクティビティモニタのみ」にチェックを入れます。
   その他のパラメータは、通常のリモートデバッグの場合と同じです。



2. メニュー「表示(V) → アクティビティモニタ(M)」を選んで、アクティビティモニタ画面を開きます。



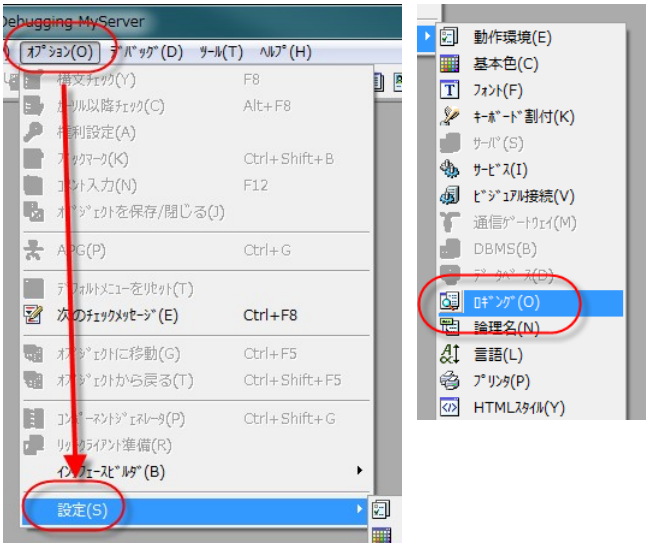


7.5. アクティビティモニタの内容を変更するには？

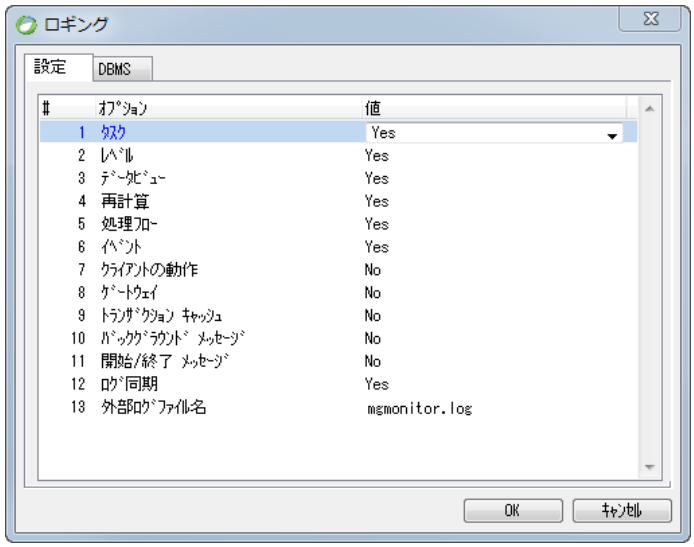
リモートデバッガで接続すると、実行エンジンの実行のようすをアクティビティモニタに表示させることができますが、ここに出力する内容の詳細を、ロギング ダイアログで動的に変更することができます。

アクティビティモニタに出力する内容を変更するには

1. メニュー「オプション (O) → 設定 (S) → ロギング」を選び、ロギング ダイアログを開きます。



2. ロギングダイアログで、設定を変更します。設定した内容は、即座に反映されます。



ロギング ダイアログの設定については、2.5「ロギングダイアログでフィルタを設定するには？」あるいは、リファレンスマニュアルの「設定 → ロギング → [設定] タブ、および [DBMS] タブ」を参照してください。

7.6. リモートデバッグのセキュリティは？

デバッガでは、実行中のアプリケーションについてほとんど全ての情報を取得することができます。例えば、プログラムのデータビューやロジックはもちろん、現在実行中のプログラムの項目の値やコールスタックなども見ることができます。また、項目の値やプログラムのコントロールもステップ実行により1ステップづつ、手作業で強制的に変更することも可能です。
このため、デバッガは大きなセキュリティホールになる可能性があります。これを防ぐには、デバッガのリモート接続についても、誰でも自由にできてしまうのは不適当であり、デバッガの接続のための権限設定を行って、一定の認証にパスした人だけが接続できるようにしておくべきです。
uniPaaS のリモートデバッガでは、セキュリティを確保するために、プロジェクトの「アプリケーション特性」で「リモートデバッガ権利」を設定できるようになっています。

リモートデバッガでのセキュリティの設定の設定にあたり、リモードデバッガ権利キーと、その権利を有するユーザを決定しておく必要があります。

- リモートデバッガ権利のキーを、半角英数字10文字以内で決定します。ここでは例として、「DEBUGCON」というキーを使いますが、実際にはなるべく推測しにくいキーを使ってください。
- リモートデバッガ権利を有するユーザの ID、パスワードを決定します。ここでは例として、ID は「debugusr」、パスワードは「debug」とします。実際にはなるべく推測しにくい ID/パスワードを使ってください。

セキュリティの設定は、次の3ステップで行ないます。

- リモートデバッガ権利をプロジェクトに設定します。（開発環境）
- 実行環境で、リモートデバッグ用のユーザを登録します。（実行環境）
- リモートデバッグ接続時に、ユーザ ID/パスワードを指定します。（開発環境）

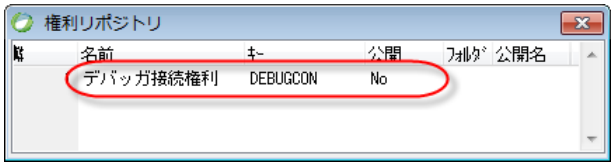
それぞれについて、以下に説明します。

7.6.1. リモートデバッガ権利の設定

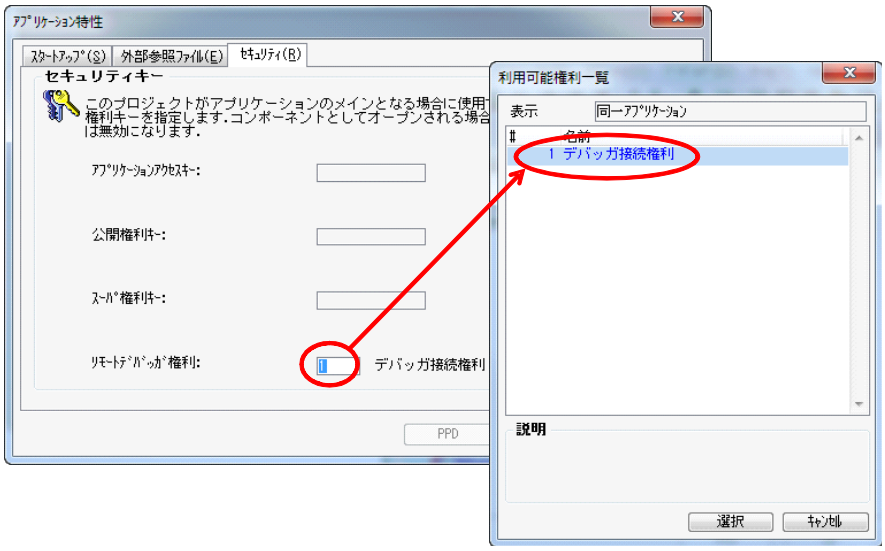
最初に、リモートデバッガの権利を定義し、プロジェクトに設定します。これは Studio で行ないます。

リモートデバッガ権利をプロジェクトに設定するには（開発環境）

1. 開発環境で Studio を起動します。
2. プロジェクトの 権利リポジトリを開き、リモートデバッグ用の権利を定義します。



3. アプリケーション特性を開き、「セキュリティ」タブの「リモートデバッガ権利」からズームします。「利用可能権利一覧」が表示されます。
4. 一覧の中から、上記ステップ2で作成したリモートデバッガ権利を選択します。
5. アプリケーション特性を閉じ、キャビネットファイル（ECFファイル）を再作成します。
6. 作成したECFファイルを、実行環境にコピーします。

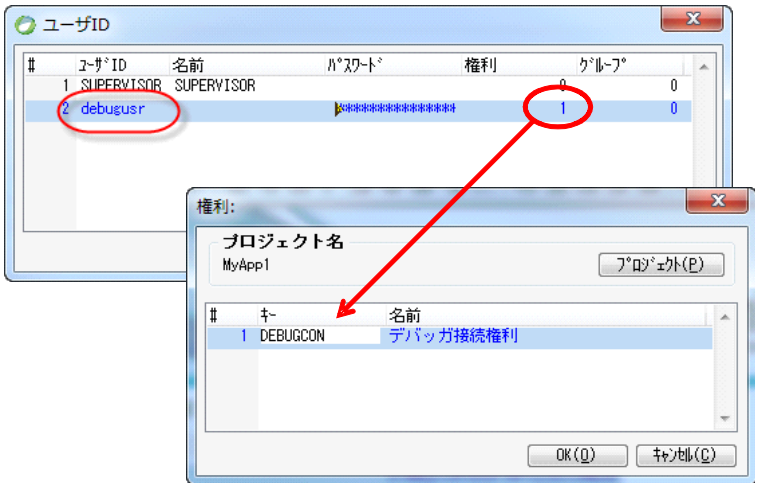


7.6.2. リモートデバッグ用のユーザの定義

次に、リモートデバッグ時にログインするためのユーザを、実行環境で定義します。このユーザには、リモートデバッガ権利を割り当てます。

実行環境で、リモートデバッグ用のユーザを登録するには

1. 実行エンジンをフォアグランドモードで起動します。
2. 「オプション (O) → 設定(S) → ユーザ ID (U) 」を選んで、ユーザ ID テーブルを開きます。
3. 新規ユーザを作成し、ユーザ ID (debugusr) とパスワード (debug) を設定します。
4. 「権利」欄からズームして、1 行作成します。
5. リモートデバッガ権利のキー (DEBUGCON) を割り当てます。

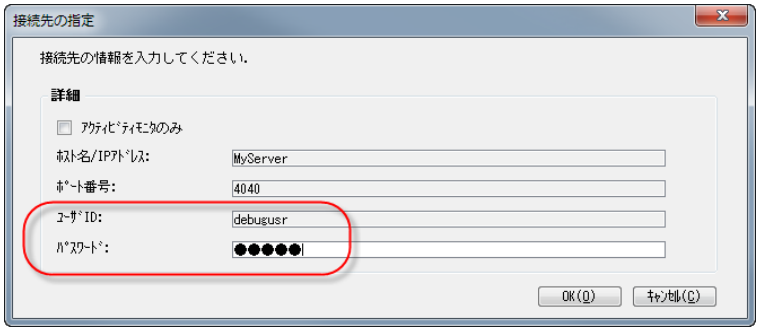


7.6.3. リモートデバッガの接続

以上の準備ができたら、Studio から実行エンジンにリモート接続します。

ユーザ ID/パスワードを指定してリモートデバッグ接続するには？（開発環境）

1. 接続の方法は、7.3.4「Studio からの接続」と同じですが、「接続先の指定」ダイアログで、ユーザ ID とパスワードとして、デフォルトの supervisor ではなく、先に定義したデバッグ用のユーザ ID とパスワード（この例では debugusr/debug）を指定するところが異なります。



第8章　DBMS のユーティリティ

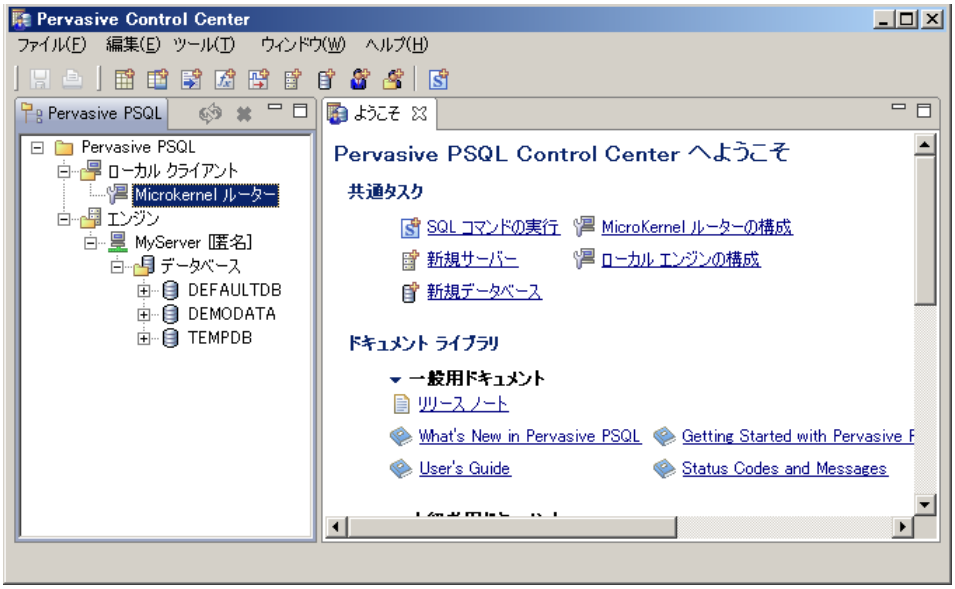
本章では、DBMS が提供しているユーティリティ群のうち、uniPaaS アプリケーションのデバッグの段階で便利に使えるツールを紹介します。

ここでは、uniPaaS アプリケーションのデバッグの段階において有用と思われるツールの機能にフォーカスして説明します。それぞれのツールの機能としては、本章で紹介する以上に、非常に豊富な機能を持っているものも多いのですが、本章では割愛します。また、各ツールの具体的な利用方法の説明も省略しています。これらについて、詳しくは、各 DBMS ユーティリティのマニュアルなどを参照してください。

8.1. Pervasive PSQL のツール

8.1.1. Pervasive Control Center

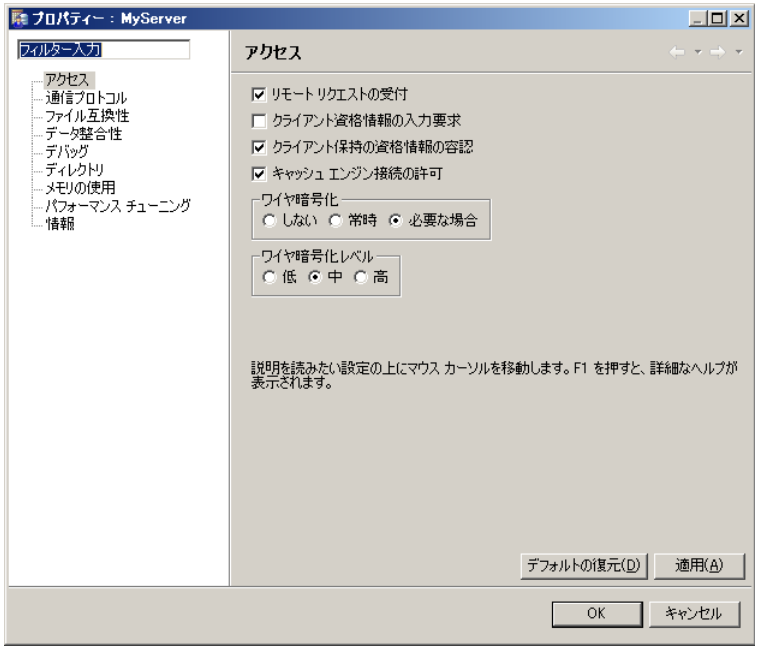
Pervasive Control Center は Pervasive PSQL (以下、Pervasive と略称)のエンジン、クライアント、データベース情報などの確認・設定のみならず、添付技術文書やログの確認など、Pervasive に関する作業の中心となるユーティリティです。



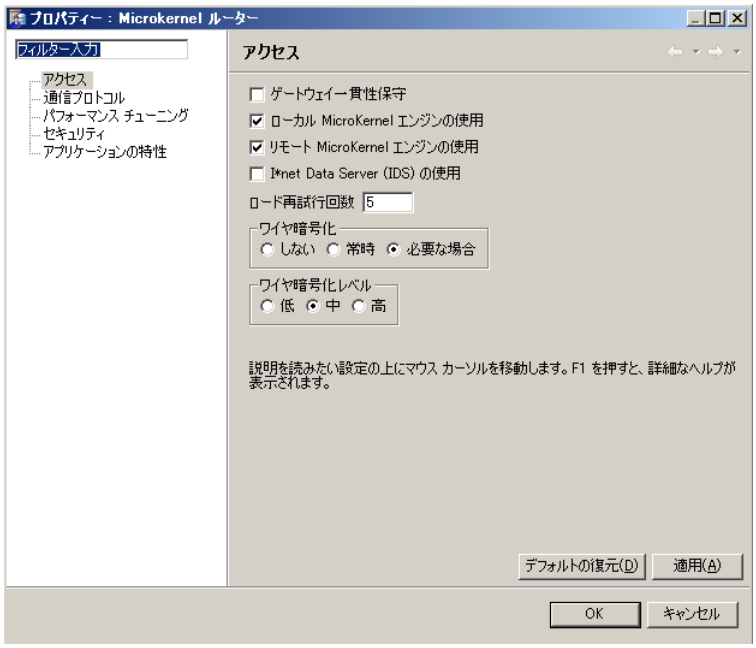
ここでトラブルシューティング時に良く使うのは、次の二つです:

- ローカルエンジンの構成: この PC 上で実行されている Pervasive の DBMS エンジンに関するパラメータの設定。

  「エンジン」の下にある、PC 名 (上の例では MyServer) のノードから、コンテキストメニューで「プロパティー」を選択するか、あるいは「ようこそ」画面の「ローカルエンジンの構成」をクリックすると、構成画面が表示されます。



- MicroKernel ルータの構成：他の PC にデータベースエンジンがある場合、データベースエンジンに接続する際の、クライアント側のパラメータの設定。「MicroKernel ルータ」ノードからコンテキストメニューで「プロパティー」を選択するか、あるいは「ようこそ」画面の「MicroKernel ルータの構成」をクリックすると、構成画面が表示されます。



8.1.2. Magic 実行時の標準的な設定

Pervasive では、uniPaaS の実行にあたって、安全性を高めてトラブルを極力回避するという意味で、一般に推奨している標準的な設定があります。Pervasive のテーブルのアクセスに問題があると思われる状況では、Pervasive の Control Center において以下のような設定をしてみてください。これは各サーバおよびクライアントにおいて設定することになります。

ローカルエンジンの構成

カテゴリ	パラメータ	標準設定	備考
通信プロトコル	サポートプロトコル	不要なものは削除	通常は TCP/IP のみでよい
	自動再接続の有効化	オン	
	自動再接続タイムアウト	1800 秒（30 分）などの大き目の値	
データ整合性	トランザクション一貫性保守	オン	Magic では必須の設定
メモリの使用	非アクティブ時、最小の状態に戻す	オン	長時間未使用時に応答なしとなる場合
パフォーマンスチューニング	セグメントサイズを 2GB に制限	オフ	ファイルを分割したくない場合

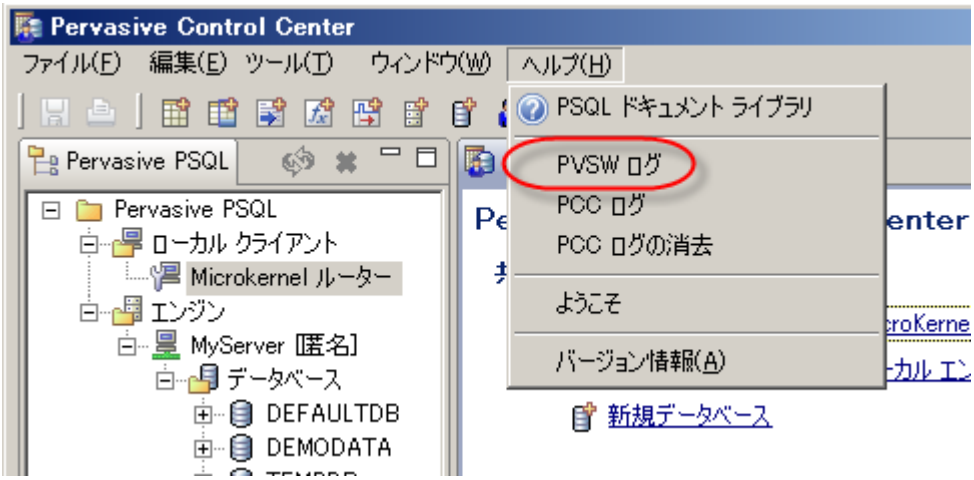
Microkernel ルーターの構成

カテゴリ	パラメータ	標準設定	備考
通信プロトコル	サポートプロトコル	不要なものは削除	通常は TCP/IP のみでよい
	自動再接続の有効化	オン	
パフォーマンスチューニング	キャッシュエンジンの使用	オフ	更新処理が多い場合

8.1.3. ログ

Pervasive のレベル (DBMS エンジンおよびクライアントモジュール)に重要なエラーが発生した場合には、ログファイルに記録されます。

このログファイルは pvsw.log という名前で、Pervasive のバージョンやエディション、Windows のバージョンなどにより格納されている場所が異なりますが、Pervasive Control Center のメニュー「ヘルプ(H) → PVSW ログ」を選んで表示させることができます。



8.1.4. トレース

uniPaaS の Pervasive ゲートウェイは、Pervasive のクライアントライブラリに対して、Pervasive が提供する API を介してアクセスしています。uniPaaS のプログラム実行中に、Pervasive テーブルへのアクセス時に問題が起こる現象が見られたら、詳細調査のために、この API レベルでのトレースを確認してみたくなることがあります。

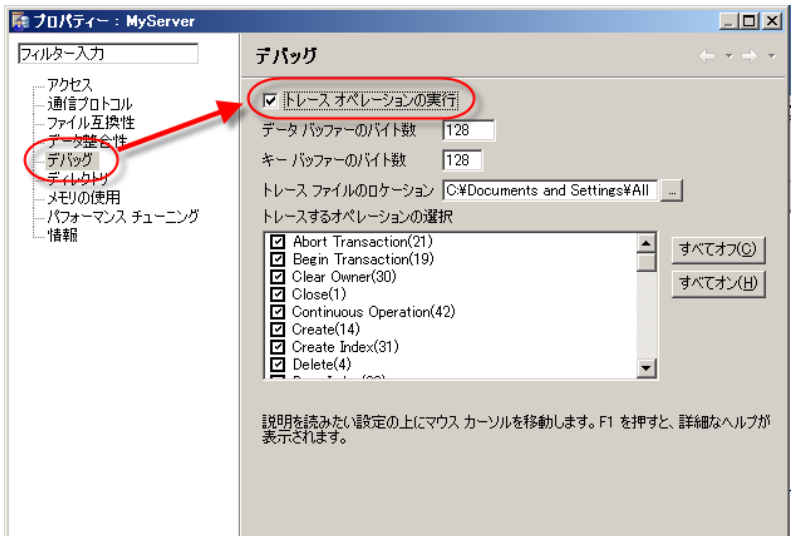
Pervasive では API レベルのトレース機能を提供しており、これを使って API のアクセスの様子を確認することができます。

Pervasive のトレースを有効にするには

2. Pervasive のデータベースエンジンが実行されている PC 上 (DBMS サーバ上)で、Pervasive Control Center を開きます。

3. 「ローカルエンジンの構成」画面を開きます。

4. 「デバッグ」画面で、「トレースオペレーションの実行」をオンにします。トレースするオペレーションはすべてオンにしておいてください。他の設定はデフォルトのままで構いません。



Pervasive の「開発者用ドキュメント → PSQL Programmer's Guide → Btrieve アプリケーションのデバッグ」に、「トレース ファイル」のサンプルがいくつかあり、その読み方も説明されています。
API のインターフェースはバイナリデータで行われており、またトレース結果もかなり詳細なものになるため、トレースをとってのデバッグ調査は、かなり現象が絞り込めてきてからのことになると思われます。

8.1.5. Butil

Butil は、Pervasive が提供するコマンドラインツールで、次のような幅広い機能を持っています。

- サーバーのバックアップに使用する Continuous オペレーションの開始と終了。
- 最後のバックアップからシステム エラーが発生するまでの間に行ったファイルへの変更の回復。
- ASCII 形式、シーケンシャル形式、および SDF 形式のデータのインポートとエクスポート。
- データ ファイル間のデータのコピー。
- MKDE バージョン情報の取得。

トラブルシューティング時には、次のような使い方があります。

Pervasive 製品バージョンの確認

引数として -ver とすると、Pervasive ソフトウェアモジュールのバージョンについての情報を表示します。

```
C:¥>butil -ver
Btrieve Maintenance ﾕｰﾃｨﾘﾃｨ 10.30.017.000
Copyright (C) Pervasive Software Inc. 2009
All Rights Reserved.

BUTIL-33: Btrieve ﾘｸｴｽﾀｰのﾊﾞｰｼﾞｮﾝは 10.30 です。

BUTIL-138: Btrieve のﾊﾞｰｼﾞｮﾝは 10.30 で、Workgroup Engine 版です。

ｺﾏﾝﾄﾞが完了しました。
```

ファイルのヘッダ情報の確認

Pervasive テーブルへのアクセス時に、問題が起こるファイルと起こらないファイルがある場合、Butil -stat においてヘッダの部分の情報に差異がないかを確認することで、原因を見つけられることがあります。以下に実行例を示します。

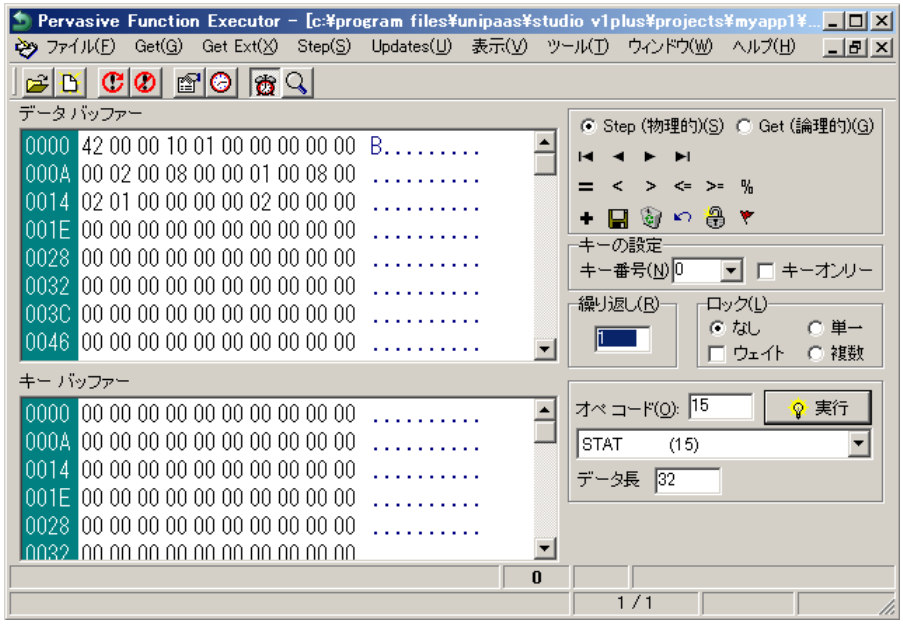
```
C:¥>butil -stat "C:¥Program Files¥uniPaaS¥Studio V1Plus¥Projects¥MyApp1¥TREECTLTAB1"
Btrieve Maintenance ﾕｰﾃｨﾘﾃｨ 10.30.017.000
Copyright (C) Pervasive Software Inc. 2009
All Rights Reserved.

File Statistics for C:¥Program Files¥uniPaaS¥Studio V1Plus¥Projects¥MyApp1¥TREECTLTAB1

File Version = 9.50
Page Size = 4096
Page Preallocation = No
Key Only = No
Extended = No
```

```
Total Number of Records = 10
Record Length = 66
Record Compression = No
Page Compression = No
Variable Records = No

Available Linked Duplicate Keys = 0
Balanced Key = No
Log Key = SYSKEY
System Data = Yes
   SYSKEY Status = Present
Total Number of Keys = 1
Total Number of Segments = 1

Key         Position        Type            Null Values*              ACS
    Segment          Length          Flags              Unique Values
  0    1         1       8  Float        M      --                 0    --
```

Pervasive のファイルを比較する場合、Butil -stat で比較する他に、エクスプローラのファイルのプロパティの情報（「詳細設定」ボタン押下時に表示される内容も含め）も比較してください。

8.1.6. Function Executor

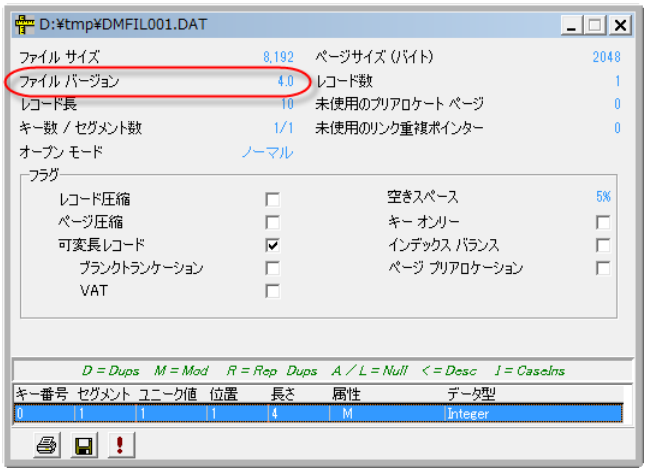
Pervasive テーブルに対して、レコードの更新・削除等を行っても、思った通りにならない場合、Pervasive が提供する Function Executor を使って、Pervasive の API レベルで直接オペレーションを行って見て、uniPaaS の側の問題なのか、Pervasive の側の問題なのかの、切り分けを行うことができます。



例えば、Pervasive のテーブルをオープンし、先頭のレコードを取得し、そのレコードを削除する、という一連のオペレーションがうまくいかない場合、Function Executor を使って、次のような操作を行ないます。

1. メニュー「ファイル → 開く」で該当ファイルをオープン
2. オペコードを 12 (GET FIRST)にして、実行。(最初のレコードに位置づけされる)
3. オペコードを 4 (DELETE) にして、実行(位置付けしているレコードが削除される)

Function Executor を使って、Pervasive ファイルのバージョンを調べることもできます。ファイルをオープンしてから、メニュー「表示 → ファイル統計情報」を選ぶと、ファイルについての情報が表示されますが、その中の「ファイル バージョン」にバージョンが表示されます。
右図の例では、ファイルバージョンは 4.0 です。

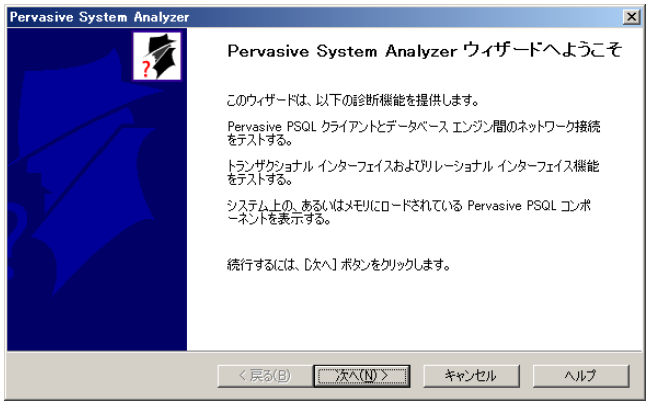


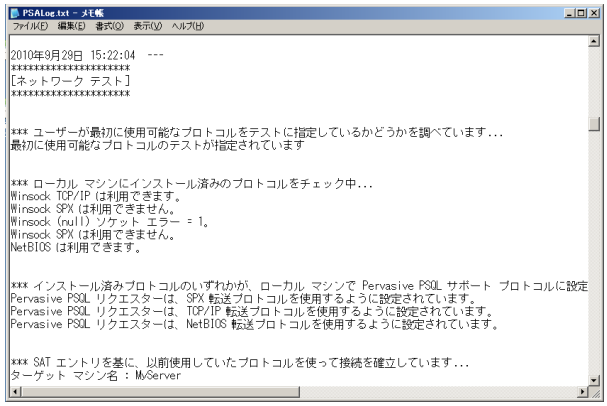
8.1.7. Pervasive System Analyzer

Pervasive System Analyzer とは、インストールされている Pervasive 製品について、以下のような一連のテストを行い、レポートするツールです。

- Pervasive のクライアントとデータベースエンジン間のネットワーク接続をテストする。
- Pervasive の API を通してデータベース処理を行い、正しく機能しているかをテストする。
- インストールされている Pervasive コンポーネントの情報を表示する。

Pervasive System Analyzer は、ウィザード形式でテスト項目を選択していき、最後にテキストファイルでレポートを保存します。



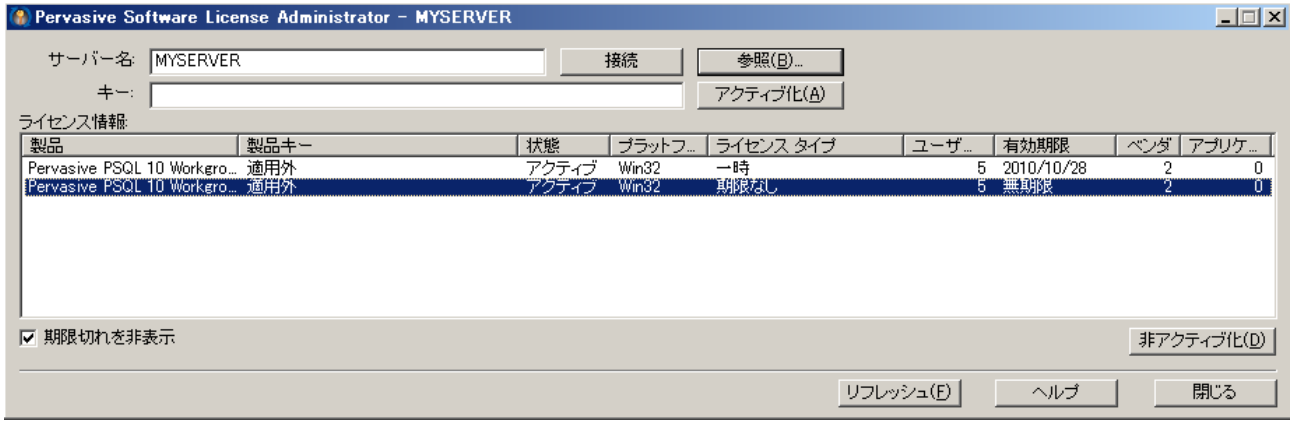


Pervasive の DBMS エンジンやネットワーク接続の動作異常が疑われる場合には、このテストを行ってエラー等が発生していないかをレポートで確認してください。

8.1.8. License Administrator

Pervasive のテーブルを使っているとき、「ファイルマネージャの異常 #161 エラー」が発生した場合には、Pervasive のライセンスが不足していたり期限切れになっていることを意味します。

このような場合、Pervasive に付属の「License Administrator」を使用すれば、有効なライセンスが存在するかをご確認できます。



8.1.9. Pervasive Software Monitor

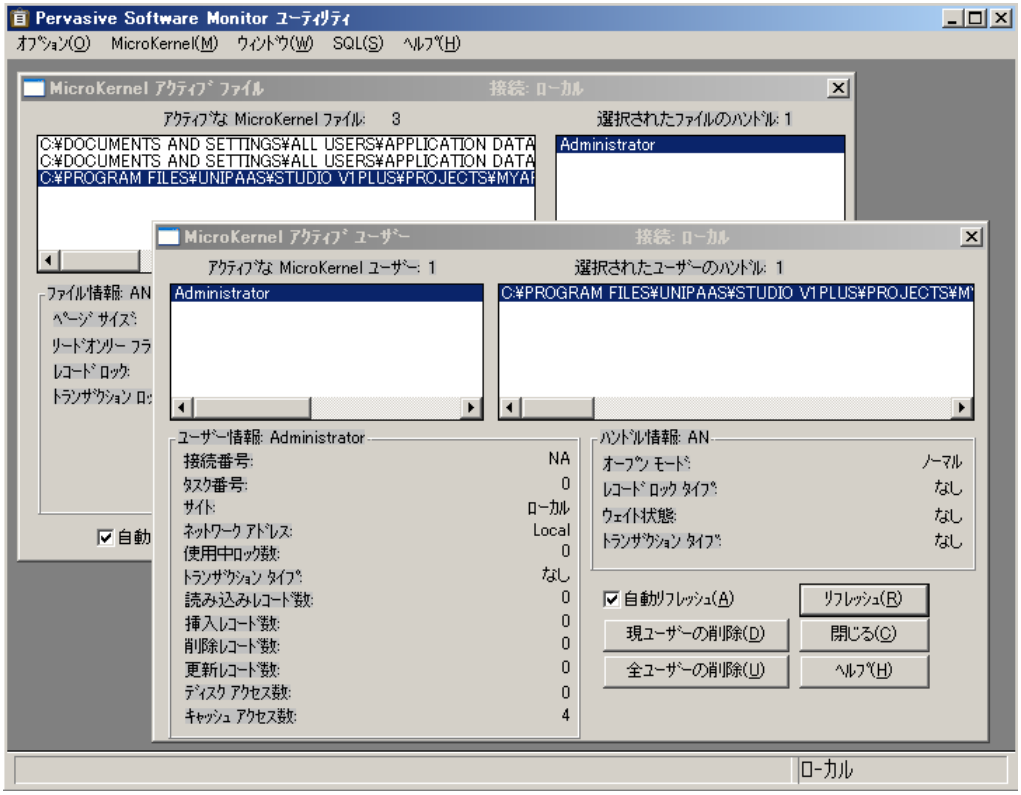
Pervasive Software Monitor は、Pervasive のデータベースエンジン (MicroKernel) の活動の状況を表示するユーティリティです。このユーティリティで次のような情報を確認することができます。

- 接続しているユーザ(クライアント)の一覧、および、各接続ごとの情報。
- 現在オープンされているファイルの一覧、および、各ファイルごとの情報。
- データベースエンジンのリソース使用状況の統計。
- クライアントとの接続の通信統計情報。

これらの情報は、例えば、次のような状況で確認のために使うことができます。

- レコードロックの解除待ちが発生する場合、どのユーザが使っているのか？
- ライセンスエラーが発生する場合、誰がどこから接続しているのか？
- 通信障害が起こるときに、通信状況がどうなっているのか？

次の図は、Pervasive Software Monitor ユーティリティの画面例です。ここでは、アクティブファイルと、アクティブユーザについての情報が表示されています。



8.1.10. 再インストール

Pervasive が正しく動作しない場合、いろいろと試しても正常に戻らない場合には、最後の手段として、Pervasive の再インストールを行うことがあります。

この場合、アンインストールしても、レジストリに情報が残っていて、設定がそのまま残っていると、再インストールした後にもそれが有効になって、本当の意味で「クリーンな環境に再インストール」とはなっていないこともあります。このため、Pervasive の再インストール時にはレジストリもクリアするようにしてみてください。

具体的には、次のような手順を行ってください。

Pervasive をクリーンに再インストールするには

4. Pervasive のアンインストール
5. マシンの再起動
6. PSA(Pervasive System Analyzer)を起動して「コンポーネントまたはアーカイブを削除する」を実行(Pervasive PSQL 10 では不要)。
7. マシンの再起動
8. PSA のアンインストール
9. マシンの再起動
10. 以下のフォルダの削除
    - C:¥PVSW
    - C:¥BTI
    - C:¥Program Files¥BTI
11. 以下のファイルの削除(特にパスの通っているフォルダにないかを確認)
    - wbtrv32.dll

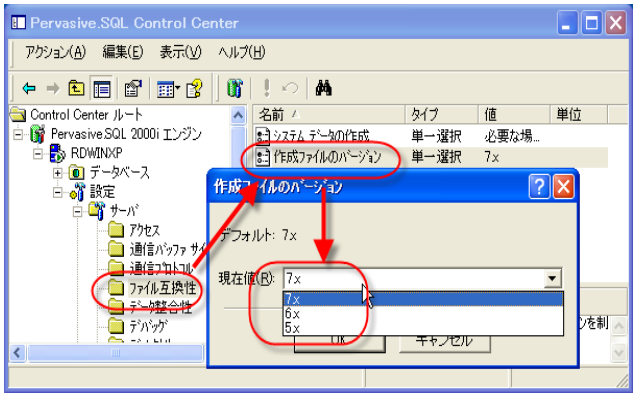
- w3btrv7.dll

12. Windows のインストールフォルダ¥pvsw.log
13. レジストリエディタの起動（コマンドプロンプトから「regedit」を実行）
14. レジストリエディタの「レジストリ/レジストリファイルの書き出し」を実行（バックアップ用）
15. 以下のレジストリの削除
    - HKEY_LOCAL_MACHINE/SOFTWARE/Pervasive Software
    - HKEY_LOCAL_MACHINE/SOFTWARE/Btrieve Technologies
16. Pervasive の再インストール

8.1.11. ファイルの互換性

Pervasive のファイルには、フォーマットにバージョンがあり、4.x ～ 10.x まで数種類があります。このうち 5.x およびそれ以前のバージョンはトランザクション処理などが異なり、最新の Pervasive エンジン (9.x 以降)では読み込み専用としてのみアクセスすることができます。もし、4.x、5.x のバージョンのファイルを、9.x 以降のエンジンを使ってアクセスし、書き込み (挿入、修正、削除)を行おうとすると、ステータス #46 のエラーとなります。
このため、古いバージョンのエンジンで作成したファイルを新しいバージョンのエンジンでアクセスするような状況では、次の点に注意してください。

1. システムの移行などで、古いエンジンを使って新たにファイルを作成することがない場合には、BUTIL などを使って、ファイルを新しいバージョンで再作成してください。
2. 古いエンジンを使うシステムと新しいエンジンを使うシステムとが共存し、ファイルコピーなどを行ってデータ共有を行うような構成の場合には、古いエンジンの「作成ファイルのバージョン」を 6.x 以上に設定してください。Control Center で 設定 → サーバ → ファイル互換性 →作成ファイルのバージョン で設定します。(右図は Pervasive.SQL 2000i での設定です)

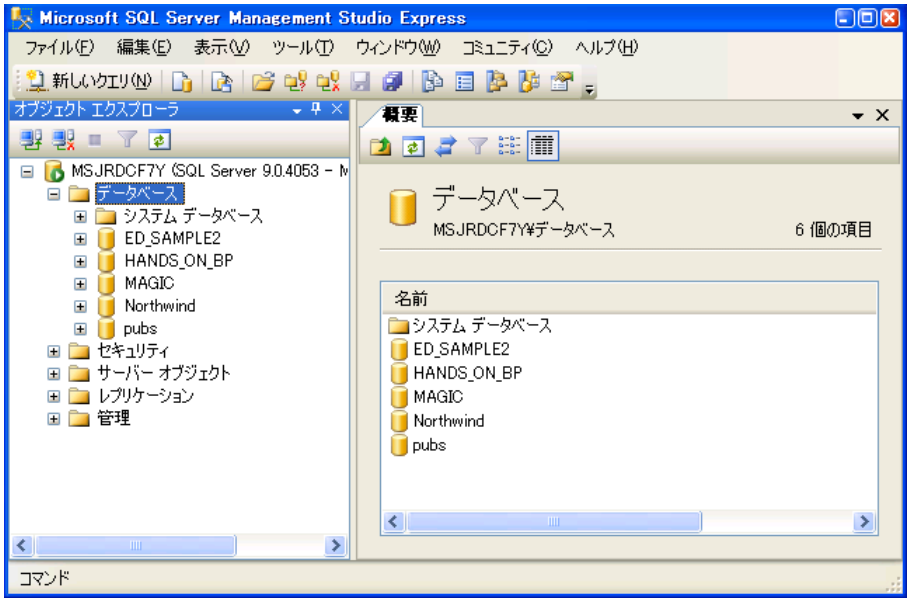


Pervasive ファイルのバージョンは、BUTIL や Function Executor を使って調べることができます。8.1.5 「Butil」 および 8.1.6 「Function Executor」 を参照してください。

8.2. MS SQL Server のツール

8.2.1. SQL Server Management Studio

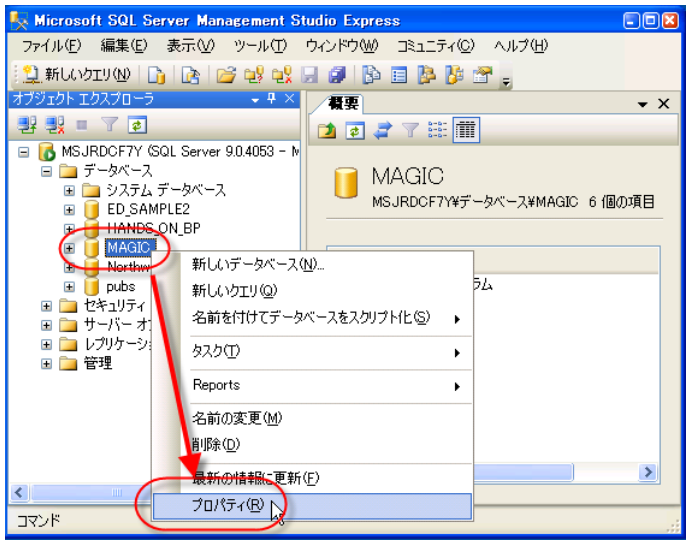
SQL Server Manageent Studio は、SQL Server を管理するための基本的なツールです。これ自身にはトラブルシューティングの機能はありませんが、データベースの構成や設定、テーブルに関する情報等を細かく確認・設定することができますので、運用時の管理目的だけでなく、テスト・デバッグ時にも必須のツールです。



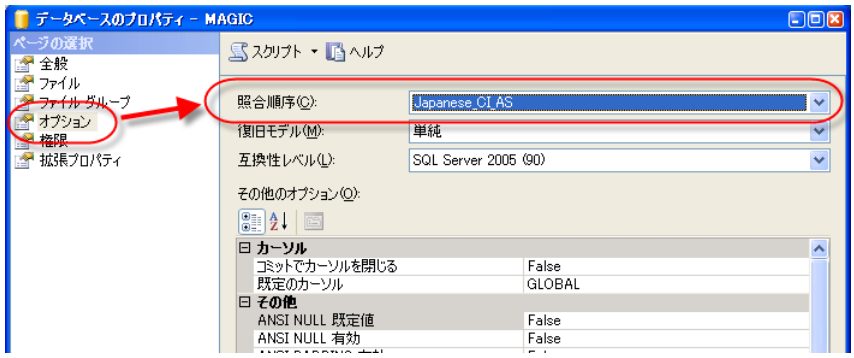
例 1：範囲付の結果が予想と異なる

UniPaaS で MS SQL Server のテーブルをアクセスする場合、範囲付や位置づけを行っているが、予期したのとは異なるレコードが検索されることがあります。このような場合には、データベースの「照合順序」が影響している可能性がありますので、SQL Server Management Studio から照合順序を確認してください。

1. データベースを選択して、ポップアップメニューから「プロパティ」を選択します。



2. 「オプション」を選択します。「照合順序」で、設定を確認できます。

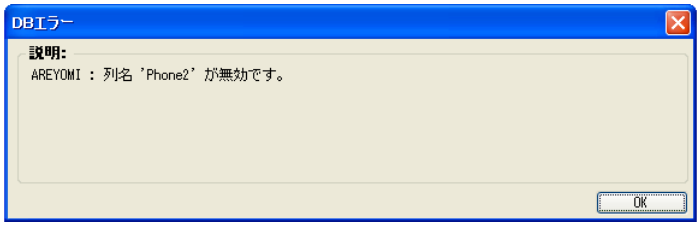


MS SQL Server の照合順序の意味については、Microsoft 社の Web サイト
http://msdn.microsoft.com/ja-jp/library/ms143515.aspx などを参照してください。

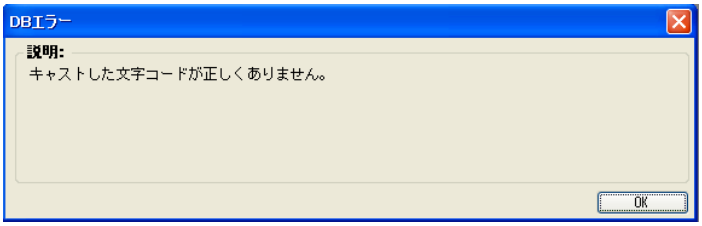
例 2：テーブルのプロパティを確認する

uniPaaS のデータベース特性の「開発モードでのテーブル変換」がオフの場合、uniPaaS のテーブルリポジトリを変更すると、uniPaaS での定義と SQL Server 上での実際のテーブルの定義とが異なることで問題が発生することもあります。

例えば、uniPaaS でカラムを追加した場合、新しいカラムは SQL Server のテーブルには存在していないので、「列名：（列名）が無効です」というエラーが出ます。
このような場合には、SQL Server Management Studio を使って、SQL Server 上でのテーブル定義（列やキーのプロパティ）を確認してください。



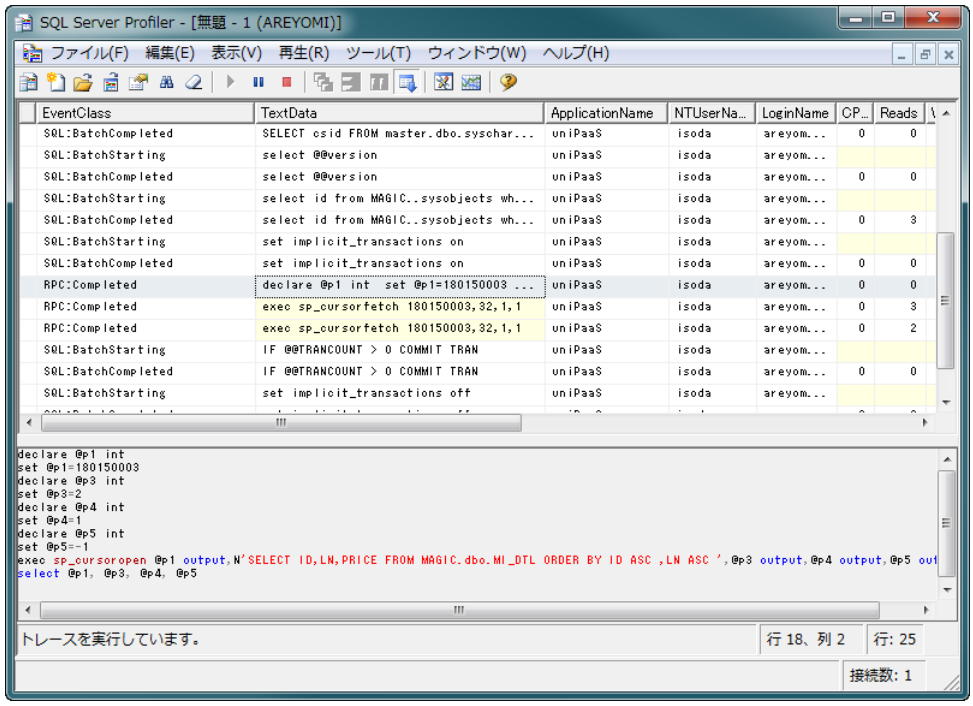
また、uniPaaS の日付型は SQL Server 上では datetime 型にマッピングされますが、SQL Server の datetime 型では「0000/00/00」という日付は不正であり、uniPaaS から入力しようとするとエラーとなります(右図)。このような場合にもカラムの特性を確認して、CHAR 型でテーブルを作成するか、あるいは 0000/00/00 という日付が入らないようにプログラム上改修を行ってください。



8.2.2. SQL Server Profiler

SQL Server Profiler は、SQL Server に付属しているユーティリティで、DBMS が実行するステートメント、実行の内部処理を経過時間などをトレースします。本来はパフォーマンスチューニングのためのツールですが、DBMS で実行される SQL 文をリアルタイムで観察することができるために、デバッグのツールとしても有効です。

下図は、デフォルトの設定で uniPaaS から APG でテーブル参照を行ったときのトレース結果の一部です。ここで見るように、SELECT 文の発行、レコードのフェッチ、トランザクションの開始、終了などが行われたことがわかります。



設定を変えることにより、トレースする情報を細かく指定することができます。詳しくは SQL Server Profiler のヘルプなどを参照してください。

uniPaaS ではサーバカーソルを利用しているために、Profiler 上ではストアドプロシージャへの呼び出しを行う、一見複雑な SQL 文が発行されています。例えば、次のようなものです。

```
declare @p1 int
set @p1=180150003
declare @p3 int
set @p3=2
declare @p4 int
set @p4=1
declare @p5 int
set @p5=-1
exec sp_cursoropen @p1 output,N'SELECT ID,LN,PRICE FROM MAGIC.dbo.MI_DTL ORDER BY ID ASC ,LN ASC ',@p3 output,@p4 output,@p5 output
select @p1, @p3, @p4, @p5
```

この SQL 文は、SQL Server のライブラリが作成しているもので、uniPaaS の SQL Server ゲーにも説明があります。トウェイが実際に発行している SQL 文は、最後から 2～3 行目にある以下の SELECT 文です。

```
SELECT ID,LN,PRICE FROM MAGIC.dbo.MI_DTL ORDER BY ID ASC ,LN ASC
```

Profiler を使って、uniPaaS が発行している SQL 文を確認したい場合には、このことに留意してください。

SQL Server Profiler は、無償の Express Edition には含まれていません。Developer Edition などをご利用ください。

第9章　uniPaaS 製品以外のツール

本章では、uniPaaS のデバッギングに役立つ、サードパーティ製品やフリーソフトなどのツールの使い方を説明します（下表）。

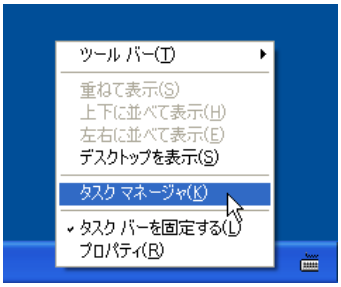
種類	機能	設定
プロセス状態	各プロセスの CPU/メモリ等の利用状況の表示	Windows タスクマネージャ、 リソースモニタ
システム イベント ログ	Windows レベルでのイベントの記録	Windows システムイベントログ
HTTP 通信	HTTP 通信データのキャプチャ	横取り丸
ネットワーク通信	ネットワーク通信キャプチャ	ネットワークモニタ
キーボードマクロ	キーボード/マウス操作の記録と再生	(各プログラム)

9.1. Windows タスクマネージャ

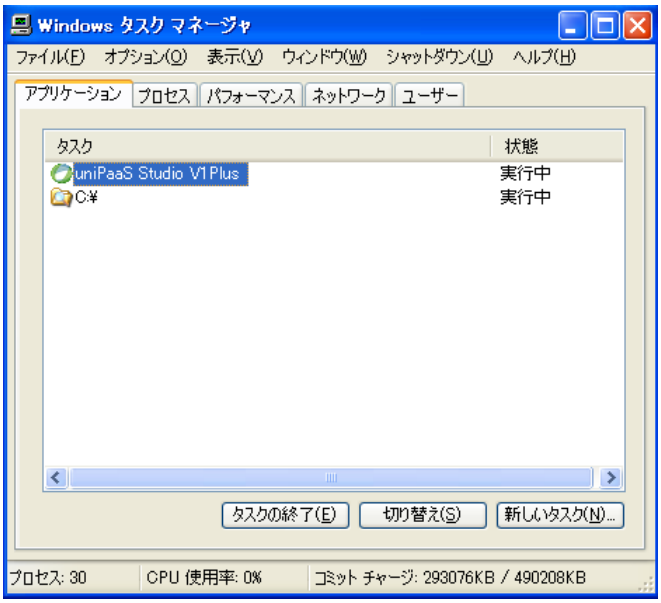
Windows のタスクマネージャは、Windows に標準に装備されているユーティリティで、現在実行中のプロセスに関する情報およびシステムの状況をリアルタイムで表示することができます。

9.1.1. STEP 1. タスクマネージャを起動する

タスクマネージャは、デスクトップのタスクバーで右クリックメニューから「タスクマネージャ」を選択することにより起動します。

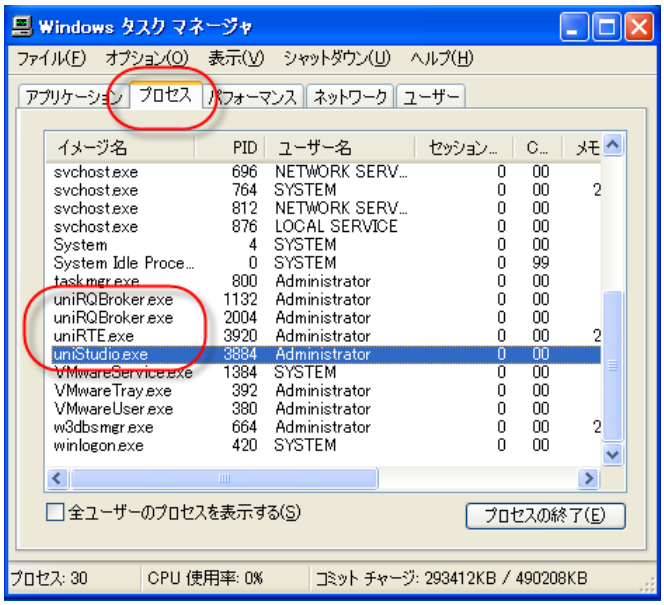


初期画面は次のように、現在デスクトップ上に出ているプログラムの一覧が表示されます。



9.1.2. STEP 2. プロセス一覧を見る

Windows で実行されているプロセスは、「アプリケーション」タブで一覧表示されているものばかりではありません。内部的にバックグラウンドで多くのプロセスが実行されています。それを見るには、「プロセス」タブを押します。一覧の「イメージ名」タイトルをマウスでクリックすると、イメージ名でソートされるので、目的とするタスクを探しやすくなります。



uniPaaS で使うプログラムの主なものは次の通りです。

uniPaaS 製品	モジュール名
開発版（uniPaaS Studio）	uniStudio.exe、uniRTE.exe
実行版（Magic Client）、および uniPaaS サーバ(Enterprise, RichClient)	uniRTE.exe
リッチクライアント クライアント側モジュール	uniRC.exe
Magic リクエストブローカ	uniRQBroker.exe

9.1.3. STEP 3. 表示項目を追加する

「プロセス」タブのデフォルトの設定では

- イメージ名
- ユーザ名
- CPU 利用%

が各プロセスごとに表示されますが、より詳細な情報を知るために、これに加えて

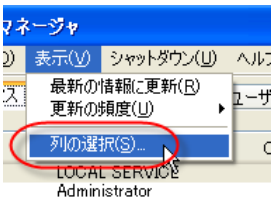
- プロセス ID
- メモリ使用量
- 仮想メモリサイズ

も表示させるようにします。
このほか、プロセスの活動状況をうかがうには、

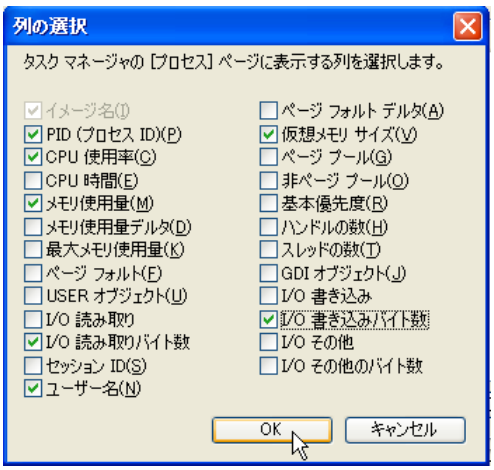
- I/O 読み取りバイト数
- I/O 書き込みバイト数

なども役に立ちます。

メニュー「表示」→「列の選択」メニューを選択します。

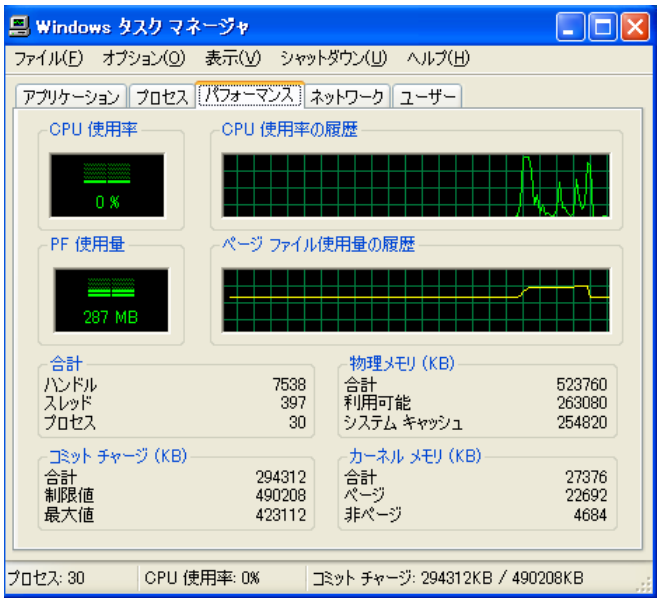


右図のような「列の選択」ダイアログが表示されますので、必要な項目を選択してください。
この設定はレジストリに記憶されるので、以後、タスクマネージャを再起動しても有効です。



9.1.4. システムの全体的な状況を見る

システムの全体的な状況は、「パフォーマンス」タブで見ることができます。ここでメモリや CPU の利用状況を確認することができます。

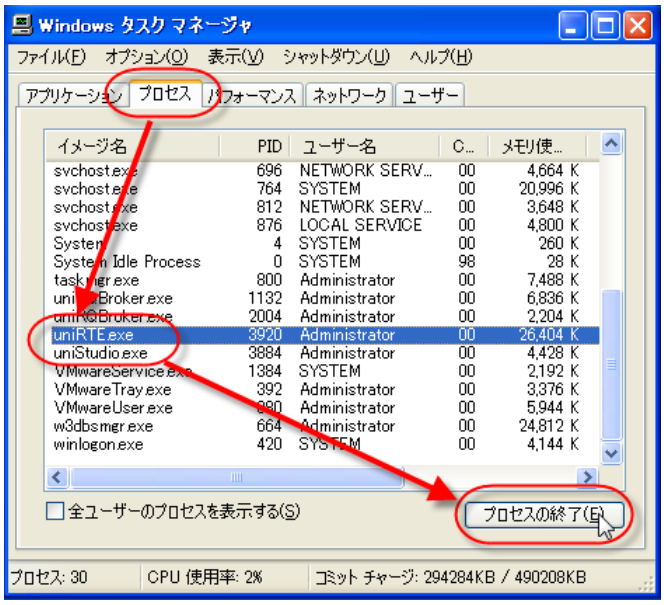


9.1.5. プロセスを強制終了させる

プログラムがハングアップしたり無限ループに陥ったりしてしまったときに、どうしても停止させることができない場合には、最後の手段として、タスクマネージャからプログラムを強制終了させることができます。

この方法を使うと、必要な終了処理などが一切行われずにプログラムが強制的に終了させられるので、データの破損や他のプログラムの動作への影響などが起こる可能性があります。したがって、プログラムの強制終了はあくまで、やむを得ず行う最後の手段としてください。

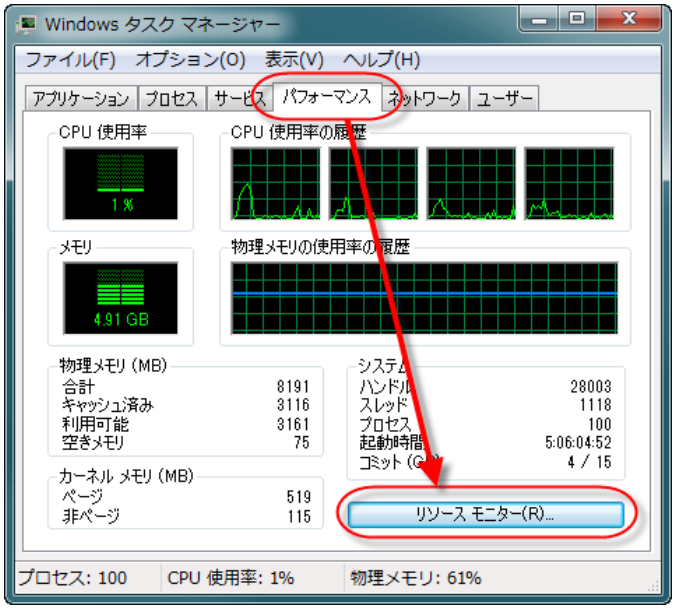
プロセスを強制終了させるには、タスクマネージャの「プロセス」タブを開き、強制しようとするプログラムにカーソルを合わせ、「プロセスの終了」ボタンを押します。



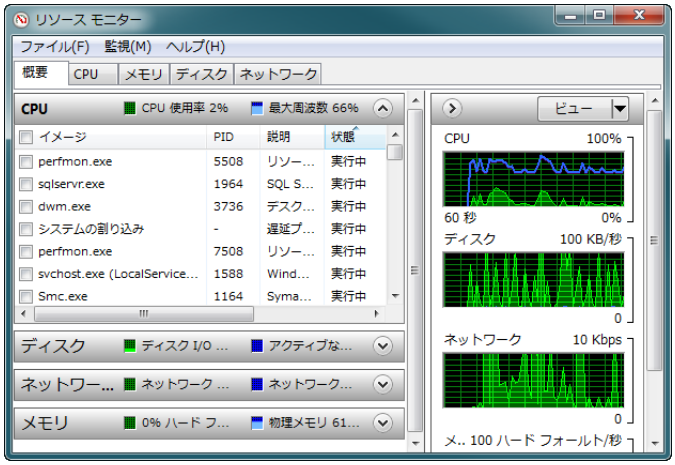
9.2. リソースモニタ

リソースモニタはタスクマネージャと同じく、Windows に備わっている Microsoft 社のユーティリティで、タスクマネージャよりもより細かく、プロセス単位で CPU 利用率、メモリ利用量、ディスク IO、ネットワーク活動などをリアルタイムで監視することのできるユーティリティです。

Windows Vista 以降の Windows (Windows Vista、Windows Server 2008、Windows 7) では、タスクマネージャから「リソースモニタ」を起動することができます。
タスクマネージャを起動し、「パフォーマンス」タブを開き、「リソースモニタ」ボタンを押します。
コマンドラインから、「resmon」 を実行しても、起動させることができます。

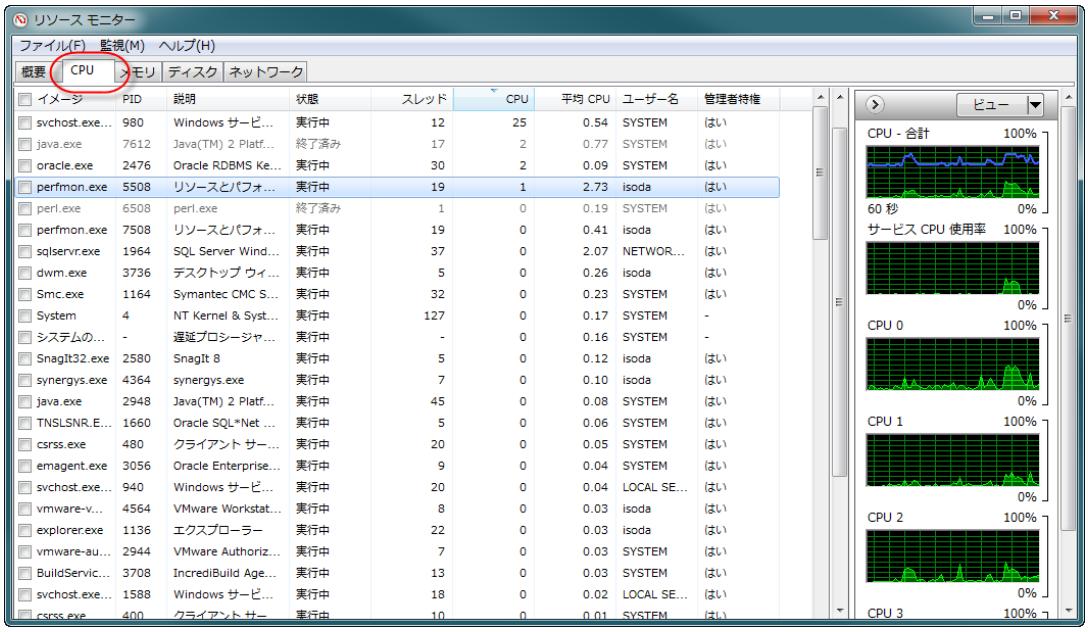


リソースモニタが開きます。



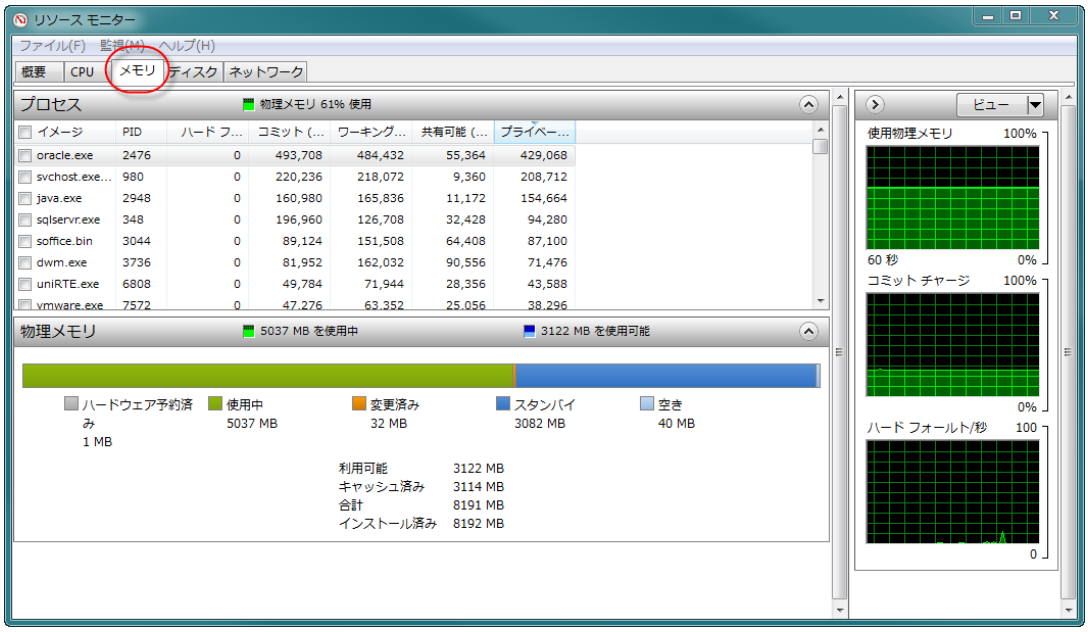
9.2.1. 「CPU」タブ

「CPU」タブを開くと、各プロセスごとの CPU 利用率、スレッド数等が表示されます。



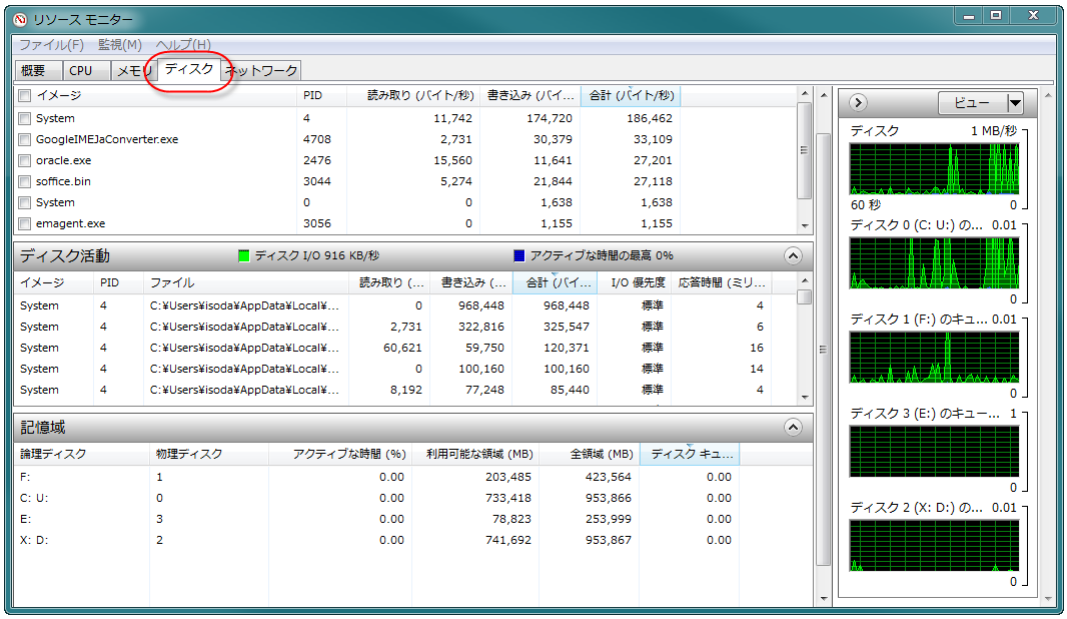
9.2.2. 「メモリ」タブ

「メモリ」タブを開くと、システム全体のメモリ使用量とその内容、および各プロセスごとのメモリ使用量の内訳を見ることができます。



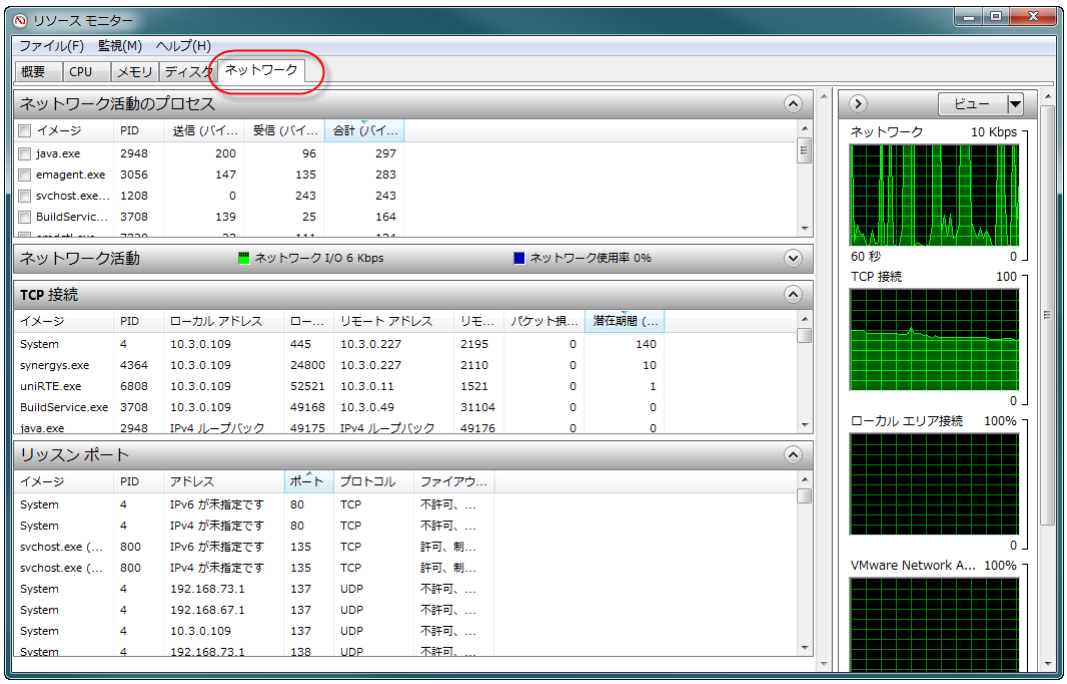
9.2.3. 「ディスク」タブ

「ディスク」タブを開くと、各プロセスごとのディスク IO の活動状況、およびシステムの各論理ディスク(ドライブ)ごとの活動状況が表示されます。



9.2.4. 「ネットワーク」タブ

「ネットワーク」タブを開くと、各プロセスごとの、TCP/IP による通信活動状況、利用しているポート番号、接続の確立したコネクションの情報が表示されます。



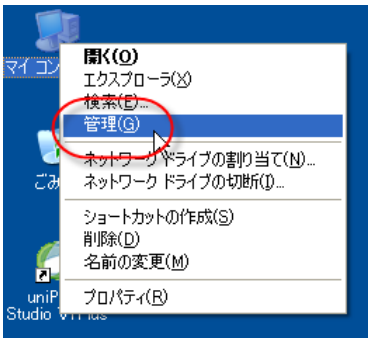
9.3. システムイベントログ

9.3.1. システムイベントログとは？

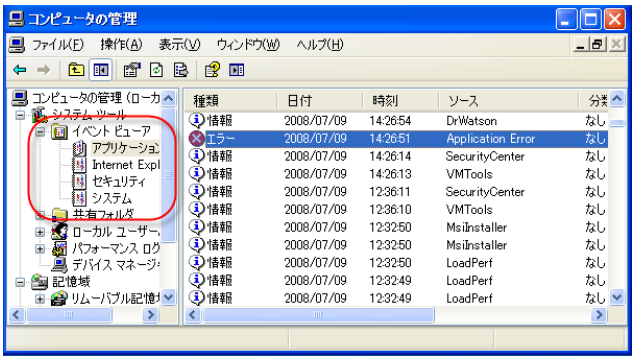
システムイベントログとは、Windows オペレーティングシステムが備えるログ機構で、「コンピュータの管理」ユーティリティの一部として統合されています。ここには、Windows がさまざまなアプリケーションを実行していくうちに遭遇するイベントを記録していきます。
uniPaaS を実行しているうちに、プログラムの実行に異常があった場合にも、異常についての簡単な情報がシステムイベントログに記録されます。

9.3.2. システムイベントログの使い方

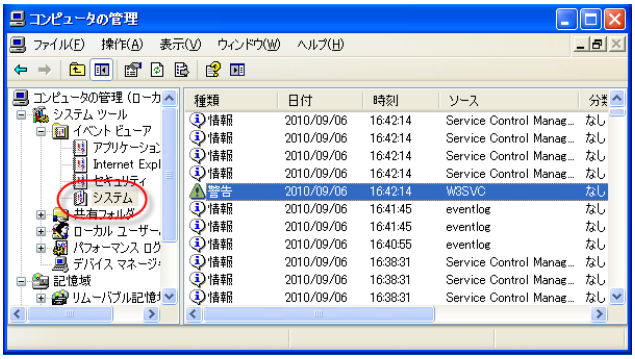
システムイベントログは、特に設定をしなくとも、デフォルトで有効になっています。
システムイベントログを見るには、まず、「コンピュータの管理」ツールを開きます。これは、デスクトップの「マイコンピュータ」アイコンを右クリックして開くメニューから「管理」を選択します。



「システムツール」→「イベントビューア」→「アプリケーション」を開くと、右図のようなイベント一覧が表示されます。



uniPaaS でエラーが起こった場合には、「アプリケーション」に、IIS でエラーが起こった場合には「システム」のイベントに記録されることが多いようです。
IIS のエラーは、「ソース」として「WAM」あるいは「W3SVC」という名前となります。

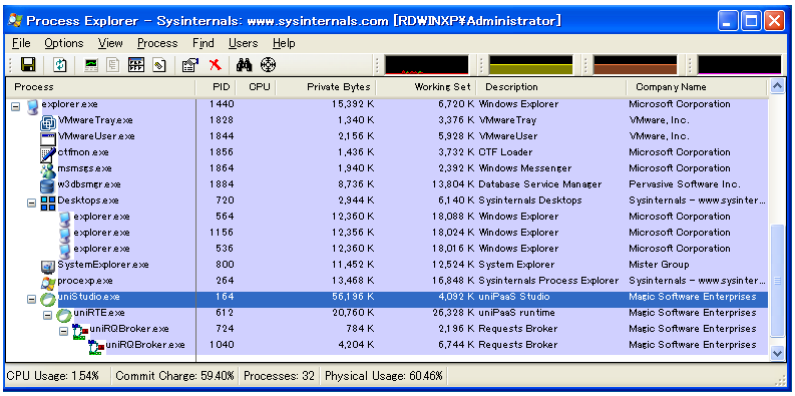


9.4. その他のシステムモニタツール

9.4.1. Process Explorer

Process Explorer は、Sysinternals (http://technet.microsoft.com/ja-jp/sysinternals/default.aspx) からダウンロードできるユーティリティの一つです。

Process Explorer は、Windows のタスクマネージャより細かな情報を見ることができます。各プロセスごとに、プロセスツリーや、イメージ・メモリ・ネットワーク・ディスク等の活動状況の詳細をリアルタイムで見ることができます。

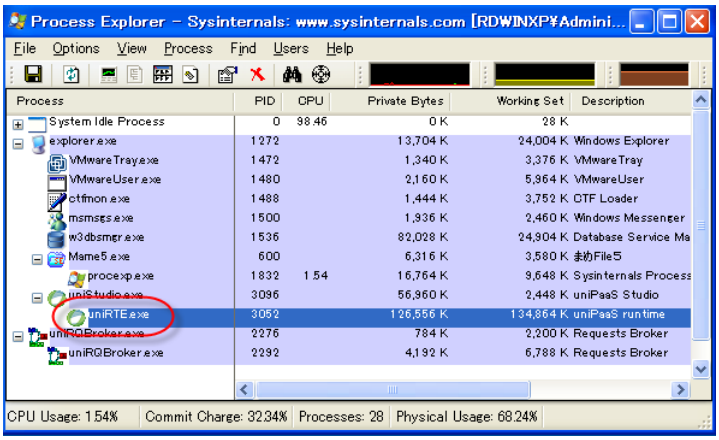


例1：uniPaaS 実行エンジンのメモリ消費の推移を見る

実行中の uniPaaS 実行エンジン (uniRTE.exe) のメモリ消費量は、システム安定稼働に影響を与えます。タスクマネージャなどでも各プロセスのメモリ使用量の瞬間値、およびシステム全体の使用量の推移は見ることができますが、各プロセスごとのメモリ使用量の推移は見ることができません。Process Explorer を使えば、各プロセスごとにメモリ消費量の推移を見ることができます。

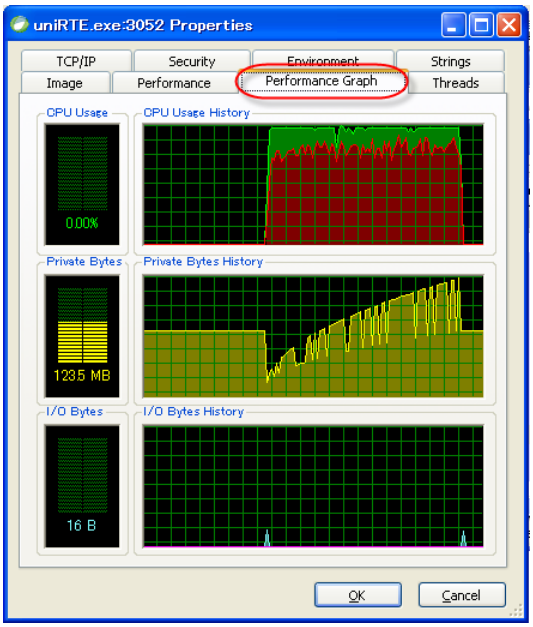
例えば、uniRTE.exe のメモリ消費量を見るには、次のようにします。

Process Explorer を開いて、uniRTE.exe を探します。



ダブルクリックしてプロセスのプロパティダイアログを開き、Performance Graph タブを開きます。

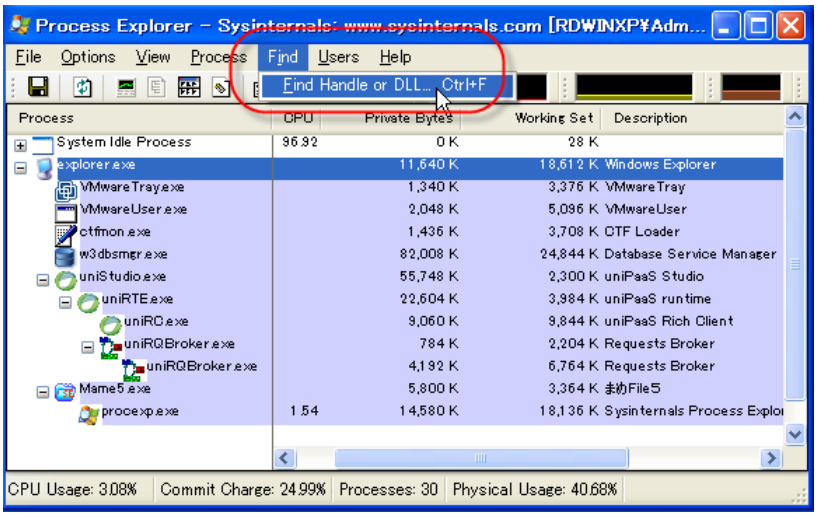
右の図は、メモリテーブルに大量のレコードを作成していったときの推移です。上から、CPU 利用率、メモリ使用量、ディスクへの入出力バイト数の推移を表したグラフです。



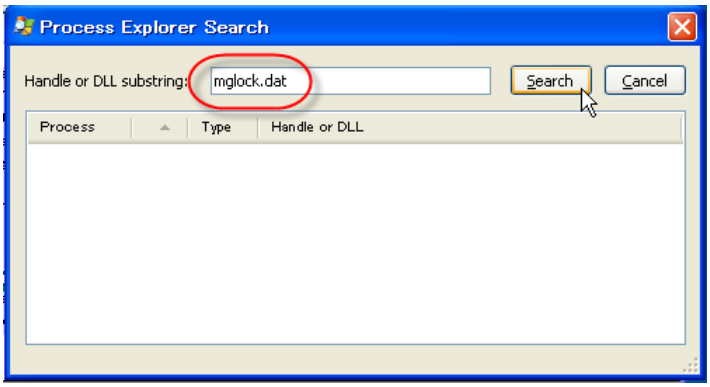
例2：mglock.dat を掴んでいるプロセスを探す

uniPaaS の実行中に、自分以外は誰も使っていないはずなのにファイルやレコードロックが発生する、ということがあります。原因はいろいろ考えられますが、uniPaaS 独自のロック機能を実現している mglock.dat ファイルを、他のプロセスがつかんだままになっている、という可能性があります。このような場合に、どのプロセスが mglock.dat をオープンしているかを調べられると便利です。
Process Explorer を使うと、特定のファイル（ファイル名の一部だけでも良い）をオープンしているプロセスを探し出すことができます。

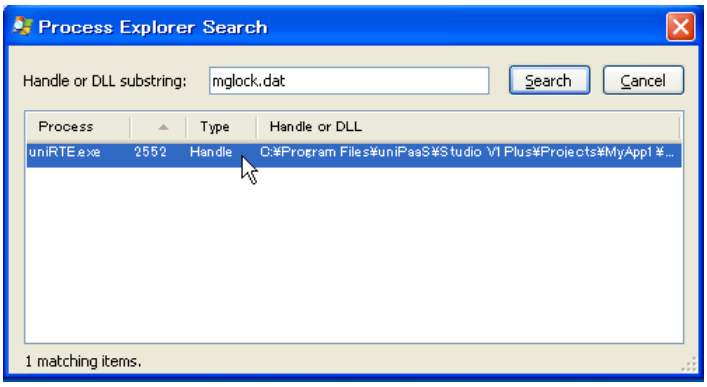
3. Process Explorer を開きます。
4. メニュー「Find → Find Handler or DLL」を選びます。Process Explorer Search ダイアログが開きます。

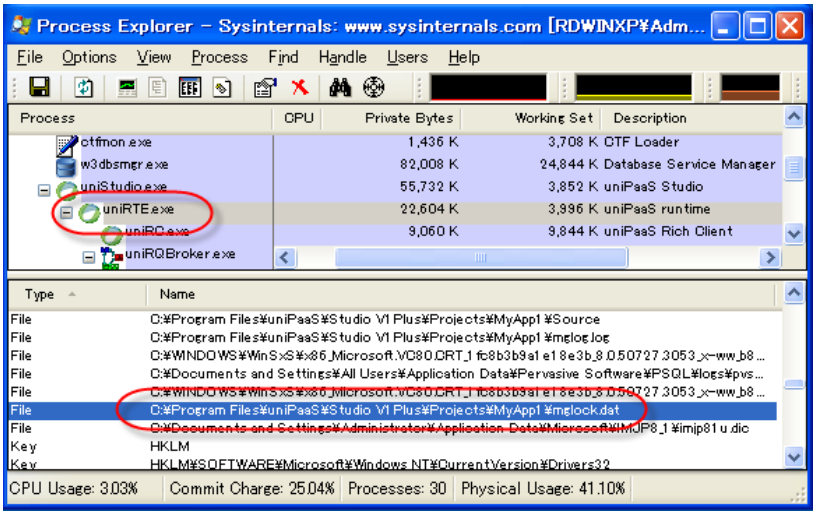


5. 「Handle or DLL substring」欄に、探したいファイル名（今の場合 mglock.dat）を指定し、「Search」ボタンを押します → 結果が表示されます。



6. ダブルクリックします → ダイアログが閉じ、そのファイルをオープンしているプロセスに位置づけられます。

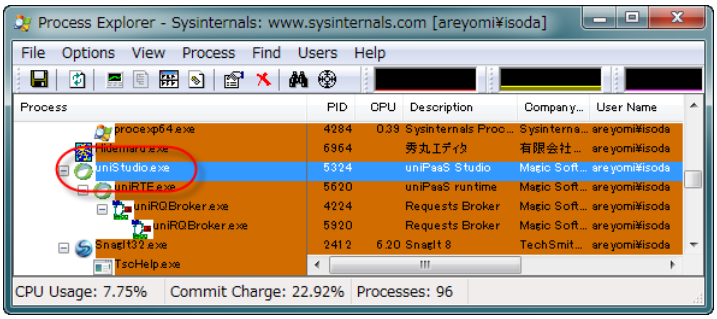




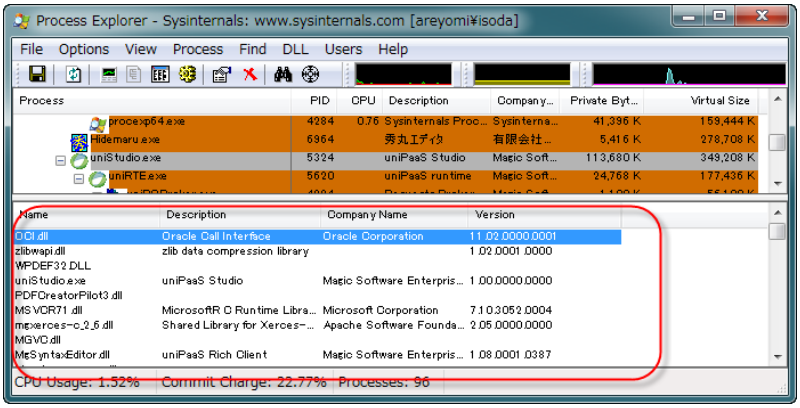
例3：ロードされている DLL を確認する。

システム上に同一名の DLL がある場合、設定などの間違いにより、名前は同じだが、意図していたものと異なる DLL がロードされて、動作が意図したとおりにならないことがあります。
たとえば、異なったバージョンの uniPaaS をインストールしていると、同一名のゲートウェイ DLL が異なったバージョンのインストールディレクトリに存在します。異なるバージョンのゲートウェイがロードされると、動作が微妙に異なったり異常終了したりしま。
また、「外部コール U=UDP」コマンドで呼び出す、ユーザ定義プロシージャの DLL として、同名だが異なったものがロードされることもあります。いずれもこのような状況は、なかなか原因をみつけにくい問題になります。
このような問題は、ProcessExplorer を使うと、特定のプロセスでロードしている DLL の一覧を見て確認することができます。

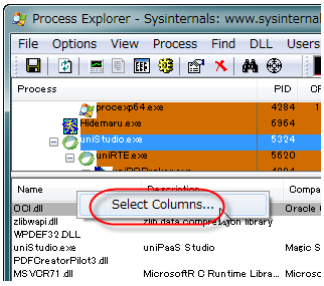
ProcessExplorer を起動し、確認したいプロセスを見つけます。
右図では、Studio のプロセス uniStudio.exe を選択しています。
実行時の動作の不正に関する場合には、実行モジュール uniRTE.exe を選択します。

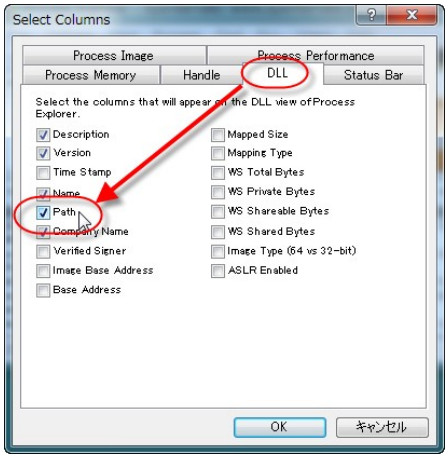


メニューから、
View → Show Lower Pane、
を選択した後、
View → Lower Pane View → DLLs
を選択します。右図のように、プロセスがロードしている DLL の一覧が表示されます。

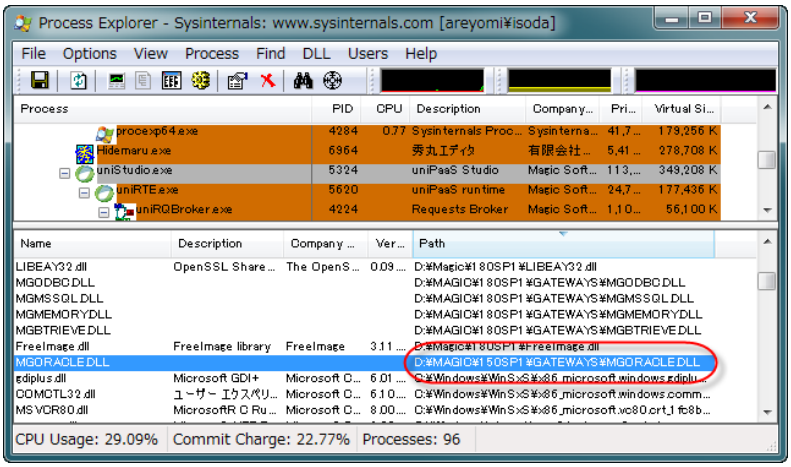


DLL のパスが表示されていない場合には、タイトル行（「Name」など）で右クリックしてポップアップメニューから「Select Columns」を選択し、ダイアログで「Path」にチェックを入れます。
（この設定は、次回以降にも記憶されます）





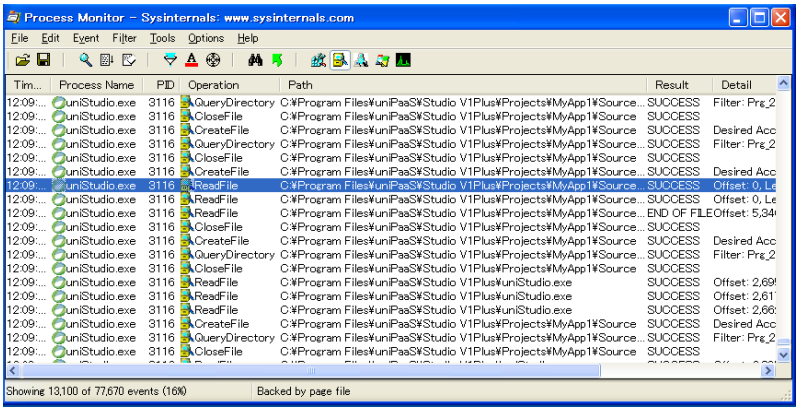
Name または Path でソートして、間違った DLL がロードされていないかを確認してください。
右図の例では、MGORACLE.DLL が異なったインストールディレクトリ D:¥MAGIC¥150SP1 からロードされていることがわかります。
これは、MAGIC.INI の [MAGIC_GATEWAYS] の設定で誤った DLL が指定されているときに起こりますので、MAGIC.INI を再確認することで対応する、ということになります。



Process Explorer、および下記の Process Monitor, VmMap を含め、Sysinternals から提供されるユーティリティをすべてまとめたものとして、「Sysinternals Suite」が無償でダウンロードできます。
(http://technet.microsoft.com/ja-jp/sysinternals/bb842062.aspx)

9.4.2. Process Monitor

Process Monitor も Sysinternals から提供されるユーティリティで、各プロセスについて、ファイルアクセス、レジストリ操作、ネットワーク活動などを、オペレーション単位で監視することができます。



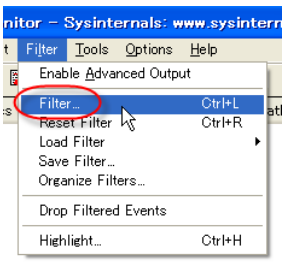
利用例1：MAGIC.INI の確認

uniPaaS 起動時に、標準の uniPaaS ディレクトリ上にある MAGIC.INI を使うのではなく、別の箇所にある MAGIC.INI を指定して起動することがあります。また、コマンドラインに @(ファイル名) の形式で、追加設定を指定することもできます。
このときに、ちゃんと指定しているはずであるにもかかわらず、思ったように設定がされていない（論理名が定義されていない、など）という場合があり、この場合には「どこの MAGIC.INI を参照しているのか？」あるいは「@(ファイル名) で指定したファイルがちゃんと読み込まれているのか？」ということを確認したくなります。
このような場合に、Process Monitor のファイルアクセス活動を監視すると、チェックすることができます。
例えば、ショートカットを作成し、「リンク先」として、

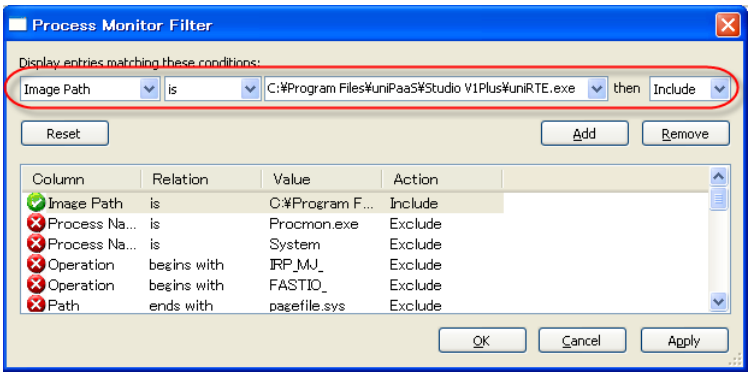
```
"C:¥Program Files¥uniPaaS¥Studio V1Plus¥uniRTE.exe" /ini=MYMAGIC.INI  @ENV¥MGADD.INI
```

と指定した uniRTE.exe を起動したときに、設定ファイル MYMAGIC.INI および ENV¥MGADD.INI が正しく読み込まれているかを確認するには、次のようにします。

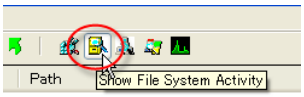
Process Monitor を起動し、メニュー「Filter → Filter」を選択します。



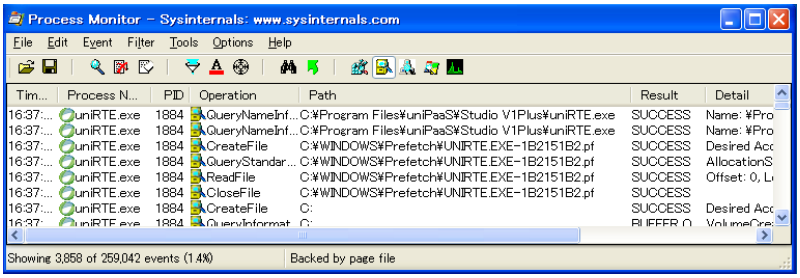
「Image Path」「is」「C:¥Program Files¥uniPaaS¥Studio V1Plus¥uniRTE.exe」 then「Include」として、「Add」ボタンを押します。
この設定は、記憶されます。



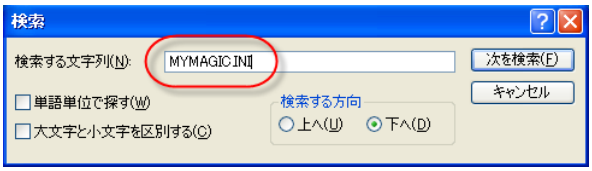
ツールバーで、「Show File System Activity」（右図のアイコン）のみを有効にしておきます。



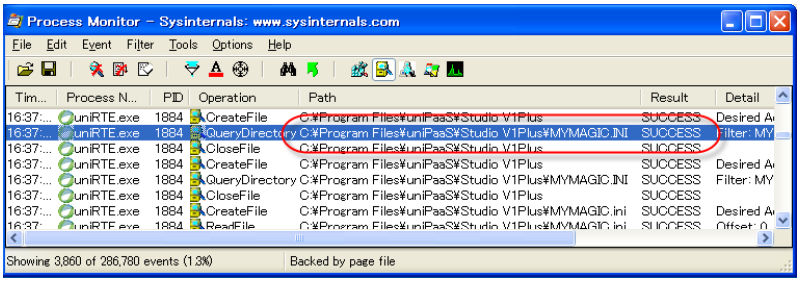
作成した uniRTE.exe のショートカットを起動します。uniRTE.exe が起動して、Process Monitor にファイル IO 活動のログが記録されます。



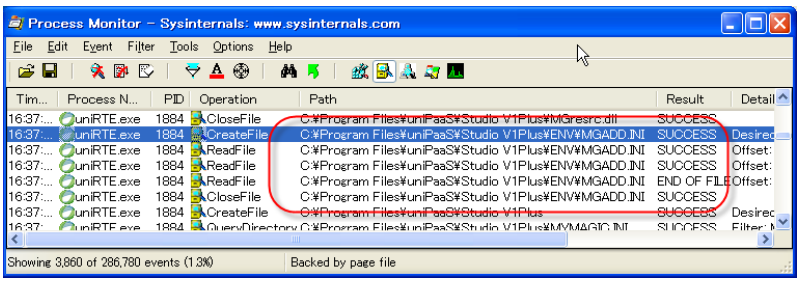
最初に、MYMAGIC.INI が正しく読まれているかを調べます。
メニュー「Edit → Find」で、MYMAGIC.INI を指定します。



Result が SUCCESS となっているので、正しく読み取れていることがわかります。その後のログを見ても、CreateFile（ファイルオープン）、ReadFile（ファイル読み込み）など SUCCESS（最後は END OF FILE）になっていることを確認します。



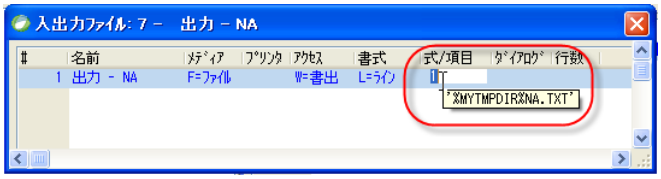
ENV¥MGADD.INI についても、同様に調べます。



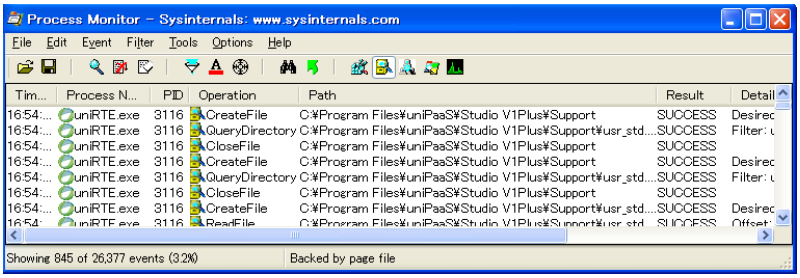
利用例2：書き込んだはずのファイルが作成されていない

バッチタスクでテキストファイルを出力したはずなのだが、ファイルが作成されていない、というような場合に、原因を追求するために、ProcessMonitor を使うことができます。

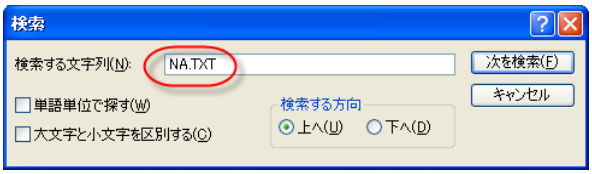
例えば、「%MYTMPDIR%NA.TXT」というファイルにデータを出力するバッチタスクを実行するとします。



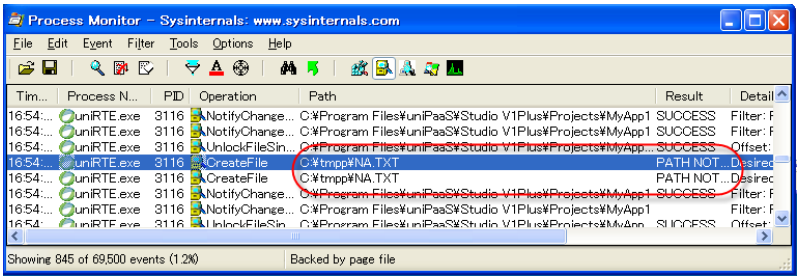
前記の例と同様にして、Process Monitor で、uniRTE.exe の ファイル IO 活動をモニタさせます。
この状態で、uniPaaS のバッチタスクを実行します。
Process Monitor に、IO 活動のログが記録されます。



「NA.TXT」をキーワードをして検索をします。



この例では、Result が PATH NOT FOUND となっていました。パス名を見てみると、C:¥tmpp¥NA.TXT となっていました。
C:¥tmp ディレクトリに書き込むつもりだったのであれば、ディレクトリ名が間違っていた、ということになります。結局、ファイルが作成されなかった原因は、論理名 MYTMPDIR の定義の間違い、ということになります。



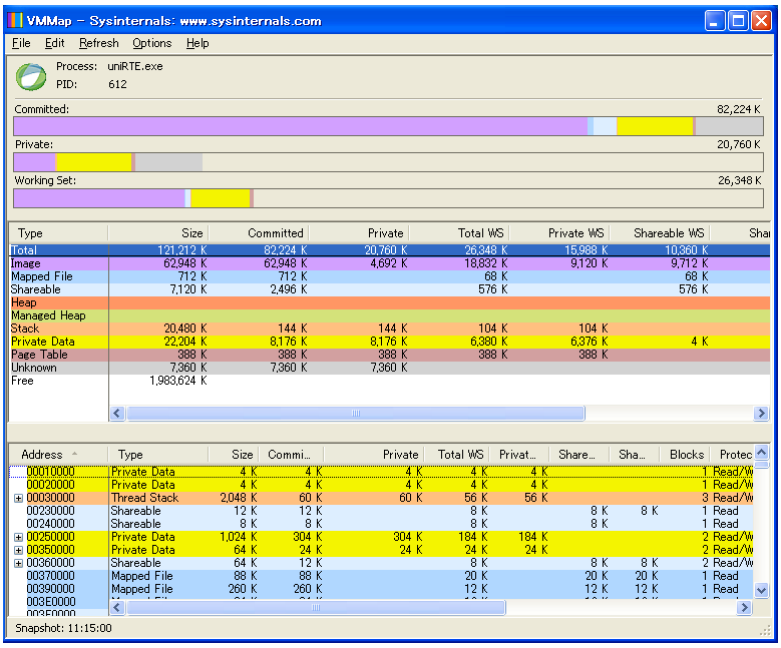
フィルタとしては、次のような設定が便利と思います。

- UniPaaS 関連のプロセスを監視するには、
  - 「Company」「contains」「Magic Software Enterprises」
- 特定のモジュール (例えば uniRTE.exe) を監視するには、
  - 「Image Path」「is」「(uniRTE.exe のフルパス名)」、あるいは
  - 「Process Name」「is」「uniRTE.exe」
- 実行中の特定のインスタンスを監視するには、
  - 「PID」「is」「(プロセス ID)」

ProcessMonitor は Windows の API レベルのログをとるので、簡単な操作でも大量のログが作成され、実行速度にも影響を与えます。従って、ProcessMonitor を使った調査は、実運用に影響を与えないテスト環境で、十分に問題が絞り込まれてから行うようにしてください。

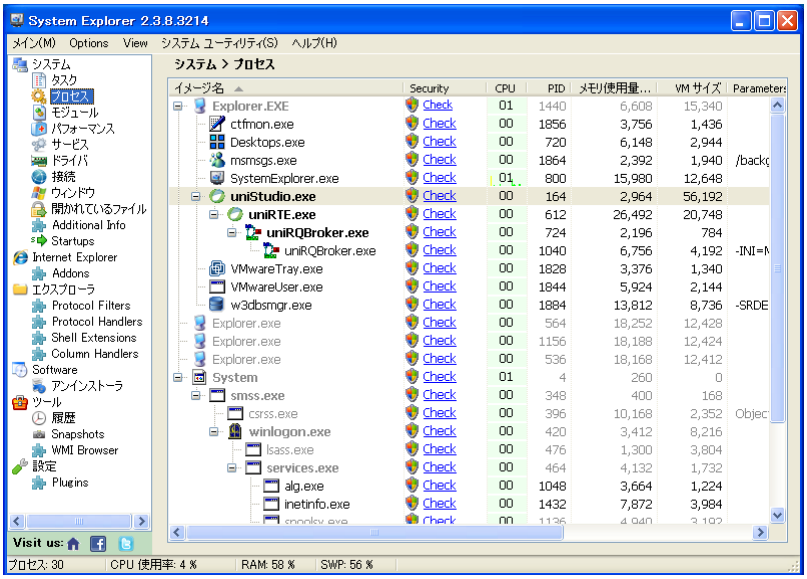
9.4.3. VmMap

vmmap は、プロセスが使っているメモリ内容をかなり細かく見ることができます。メモリリークが疑われる場合に詳細に調査することができます。



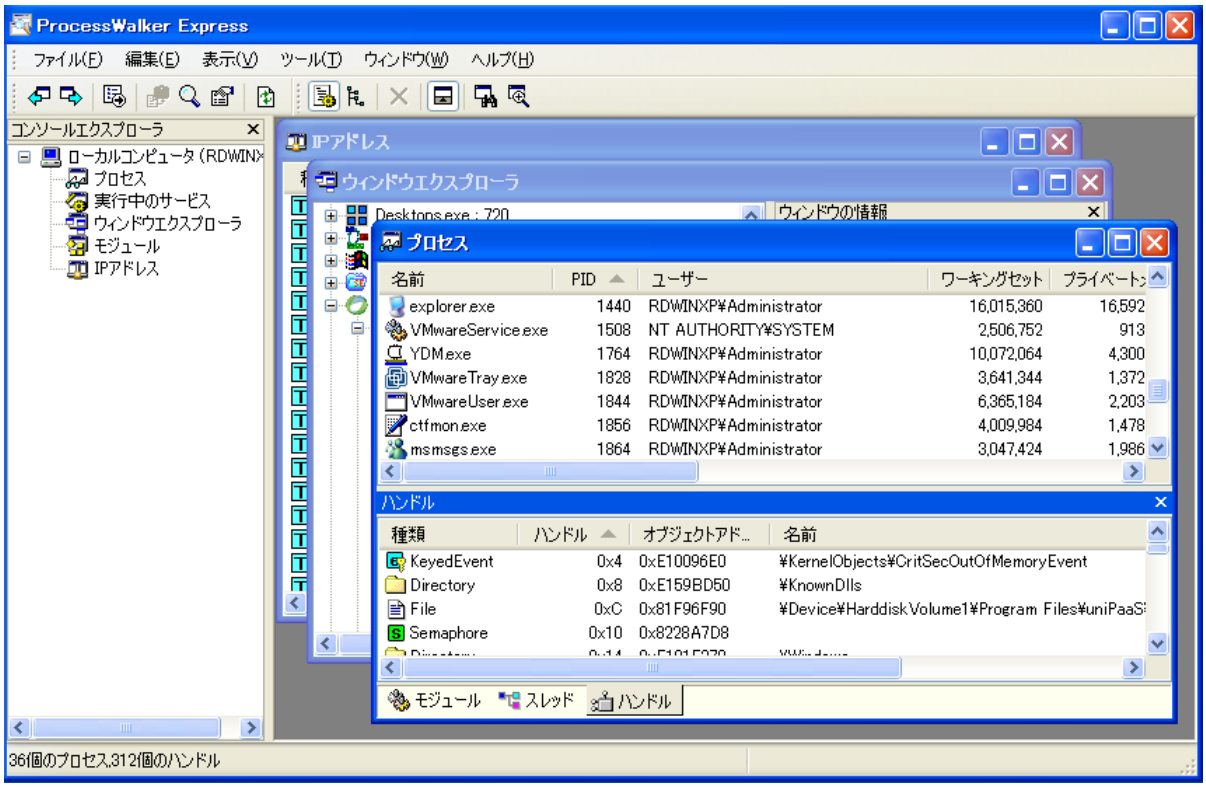
9.4.4. SystemExplorer

SystemExplorer も、システム診断のためのユーティリティで、タスクマネージャよりも詳細な情報を表示することができます。非常に多くの機能があります。
http://www.systemexplorer.net/ からダウンロードすることができます。



9.4.5. Process Walker

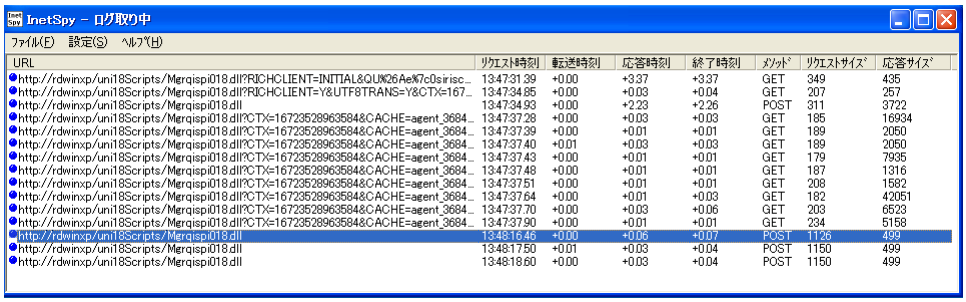
やました工房 (http://www001.upp.so-net.ne.jp/yamashita/index.htm) からも、システムやプロセスの状況監視ツールが公開されていますが、ProcessWalker Express が多くの情報を収集することができ、便利と思います。このユーティリティでも、各プロセスに関して非常に多くの情報を表示させることができます。


ProcessWalker Express
ファイル(F) 編集(E) 表示(V) ツール(T) ウィンドウ(W) ヘルプ(H)
コンソールエクスプローラ
ローカルコンピュータ (RDWIN
プロセス
実行中のサービス
ウィンドウエクスプローラ
モジュール
IPアドレス
IPアドレス
ウィンドウエクスプローラ
Desktops.exe : 720
ウィンドウの情報
プロセス
名前 PID ユーザー ワーキングセット プライベート
explorer.exe 1440 RDWINXP¥Administrator 16,015,360 16,592
VMwareService.exe 1508 NT AUTHORITY¥SYSTEM 2,506,752 913
YDM.exe 1764 RDWINXP¥Administrator 10,072,064 4,300
VMwareTray.exe 1828 RDWINXP¥Administrator 3,641,344 1,372
VMwareUser.exe 1844 RDWINXP¥Administrator 6,365,184 2,203
ctfmon.exe 1856 RDWINXP¥Administrator 4,009,984 1,478
msmsgs.exe 1864 RDWINXP¥Administrator 3,047,424 1,986
ハンドル
種類 ハンドル オブジェクトアド... 名前
KeyedEvent 0x4 0xE10096E0 ¥KernelObjects¥CritSecOutOfMemoryEvent
Directory 0x8 0xE159BD50 ¥KnownDlls
File 0xC 0x81F96F90 ¥Device¥HarddiskVolume1¥Program Files¥uniPaaS
Semaphore 0x10 0x8228A7D8
モジュール スレッド ハンドル
36個のプロセス,312個のハンドル

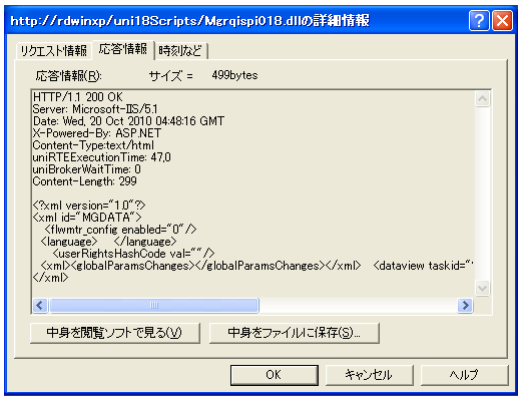
9.5. パケットキャプチャプログラム

9.5.1. 横取り丸

横取り丸というのは、HTTP プロトコルに基づく通信の要求と応答の内容を記録しておくフリーウェアのユーティリティです。Web 対応アプリケーションや、uniPaaS リッチクライアントシステムの動作に問題がある場合、原因調査に役立ちます。
横取り丸は、ベクター（http://www.vector.co.jp/）などのダウンロードサイトや、開発者のサイト（http://hidemaru.xaxon.co.jp/index.html） から無料でダウンロードすることができます。
横取り丸は、プロキシサーバとして動作し、パケットのデータを保存します。一覧には、リクエスト時刻、応答時刻、メソッド、リクエスト・応答サイズなどが表示されます。

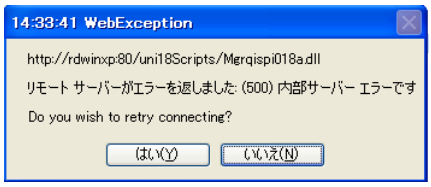


各リクエストをダブルクリックすると、その内容が表示されます。また、キャプチャしたデータはファイルに保存し、後からそのファイルを開いて調査することもできます。

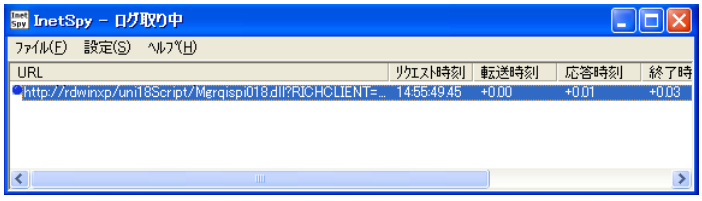


例1：リッチクライアント起動時のエラー

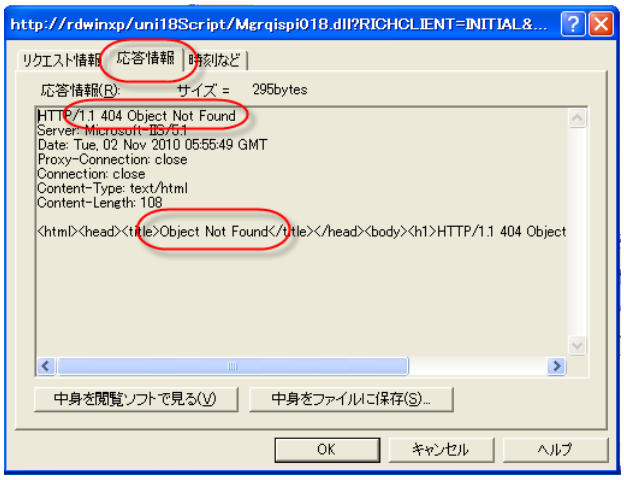
リッチクライアント関係の設定が間違っていると、リッチクライアントプログラムを起動する際にエラーが出ます。
例えば、「リモートサーバがエラーを返しました」が出る場合を見てみます。



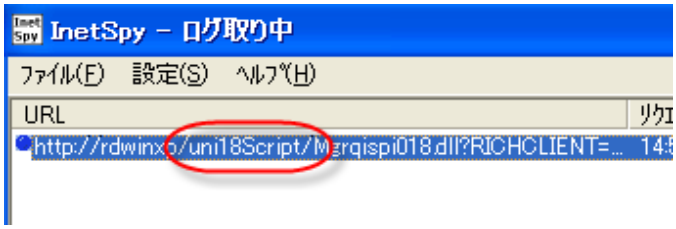
横取り丸で通信を監視すると、右図のようなパケットでした。



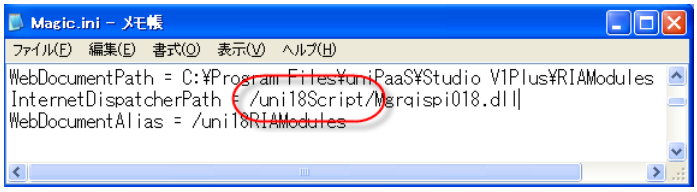
中身を見てみると、応答の結果が、「404 Object Not Found」でした。
これは、指定された URL を、Web サーバが認識できなかったことを意味します。



そこで URL を再度良く見てみると、仮想ディレクトリが uni18Script になっていました。本当は uni18Scripts（最後の「s」がある)なので、これが誤りであることがわかります。



この仮想ディレクトリは、MAGIC.INI ファイルの InternetDispatcherPath から取られるものなので、MAGIC.INI を確認し、修正します。



例2：ブラウザコントロールやイメージが表示されない

リッチクライアントプログラムの起動は OK だが、ブラウザコントロールに PDF ファイルや HTML ファイルを表示する際にエラーになる場合や、イメージファイル(背景、イメージコントロール、イメージボタン）が表示されないなどの問題がある場合を見てみます。
このようなエラーが出る場合には、しばらく(1～2分)だんまりになって、その後にエラーとなる、という場合もしばしばあります。

横取り丸で見てみると、次のような応答が見られます。
「応答サイズ」に表示がなく、ダブルクリックして「応答情報」を見ても空白でした。

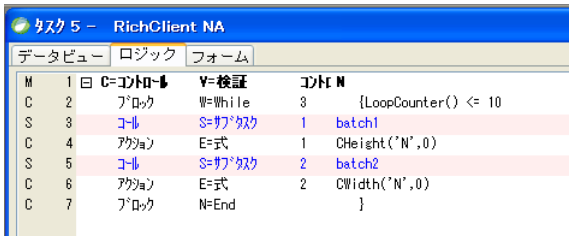
このような状況は、URL に指定されたサーバ名が認識できなかったか、指定の Web サーバが起動していなかった場合に起こります。今の例では、URL は http://myserver01/… でしたが、この myserver01 の Web サーバが認識されなかったということです。
ネットワークの設定の問題になるので、(1) myserver01 が起動しているか、(2) ping myserver01 で返答が返っ

てくるか、(3) Web サーバが起動しているか？などを確認してください。サーバの移行や、システムの移行などで、設定と実際のホスト名とが異なった可能性も考えられます。

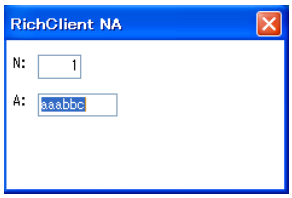
例3：起動・動作が遅い

リッチクライアントプログラムが動くは動くのだが、起動・動作が異常に遅いので、原因を突き止めたいということがあります。リッチクライアントプログラムのパフォーマンスに関しては、多くの要因がありますが、その一つとして、クライアントとサーバ間の通信量や回数が多い、という可能性があります。これについて確認するときにも、横取り丸が使えます。

ここの例では、2項目だけの簡単なリッチクライアントプログラムですが、コントロール検証でわざと多数のサーバアクセスが発生するようなロジックを入れてみました。



起動して、項目間を移動すると、もたつき感があります。



横取り丸を見てみると、カーソル項目 N から A に移動するたびに、多くの通信が起こっている状況が見て取れます。
カーソルが項目 A から項目 N に移動する際にはこのような通信はありませんでした。

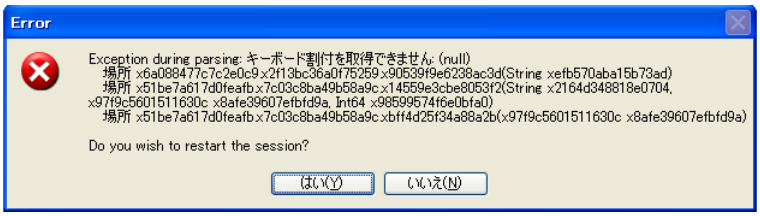
このように、横取り丸で通信の状況を観察することにより、「遅い」と感じられる場所で、裏でどのような処理が起こっているのかを、推測することができます。確認すべきは、次のようなところでしょう。

- 通信の回数：回数が多いと動作が遅くなります。この場合不要なサーバアクセスが発生しないよう、プログラムロジックを見直す必要性があります。
- 「応答時刻」「終了時刻」：1回のリクエストの応答時間に時間がかかると、全体の時間も遅くなります。応答時間のかかっているリクエストに対応するコマンドを特定し、ロジックを見なおしてください。
- 「リクエストサイズ」および「応答サイズ」：リクエストのデータ量が多いと、特に細い回線を使っている場合には、処理が遅くなります。データサイズを少なくするようにプログラム上の工夫をする必要があります。（例：チャンクサイズの調整など）

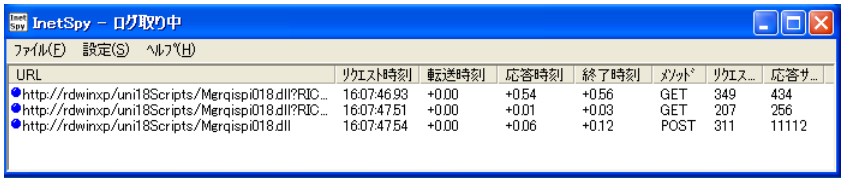
また、「例2：ブラウザコントロールやイメージが表示されない」であったように、PDF ファイルやイメージファイルの URL が間違っていると、サーバからのエラーの検出に時間がかかるので、リッチクライアントプログラムの動作が非常に遅くなってしまう場合があります。例えば、指定した URL のサーバ名が間違っていた場合、TCP/IP レベルでの名前解決のための1～2分かかり、その間、リッチクライアントプログラムはハングアップしたような状態になります。
このような場合でも、「例2：ブラウザコントロールやイメージが表示されない」で説明したように、横取り丸で通信状況を観察すれば、問題の起こる URL を突き止めることができますので、URL を修正することで、応答速度の問題を解決できます。

例4：キーボード割付を取得できません

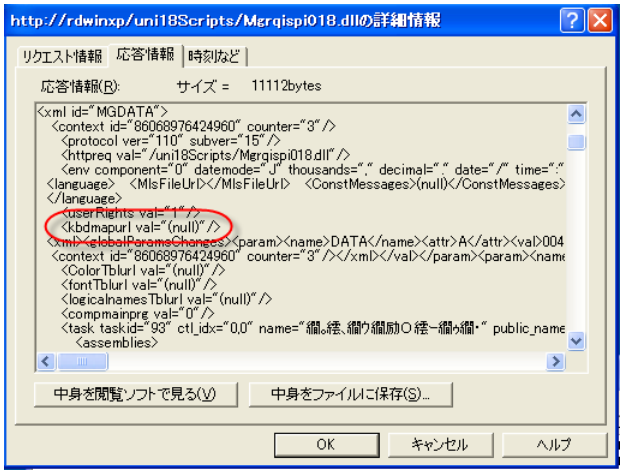
リッチクライアントの起動時に、「キーボード割付を取得できません」というエラーが出て、起動できないことがあります。



このエラーは横取り丸を漠然と観察してもわかりにくいエラーとなりますが、とにかく、横取り丸で観察すると、次のような内容になっていました。



これだけでは何が悪いのかよくわかりませんが、3 番目のパケットの内容を見てみました。すると、随所に「(null)」という設定が見つかりました。
その中に、
<kbdmapurl val="(null)"/>
という、それらしい記述もありました。
その他に、
<ColorTblurl val="(null)"/>
<fontTblurl val="(null)"/>
<logicalnamesTblurl val="(null)"/>
なども見られます。



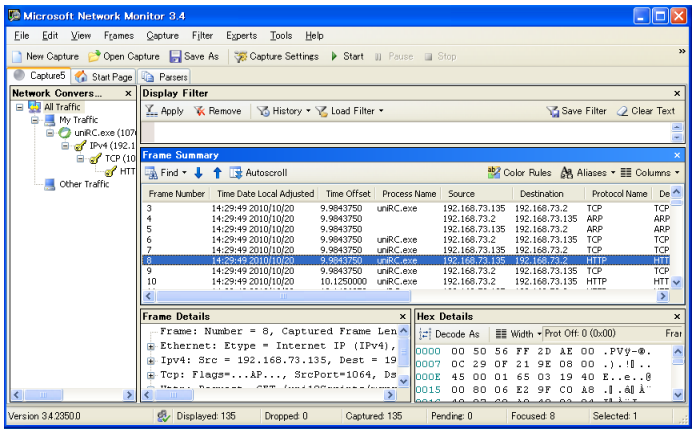
このことから、「キーボードマップテーブルなどの URL に何か問題がある」ということが推測されます。
横取り丸からは、これ以上のことを読み取ることはできませんが、キーボードマップテーブルは RIA キャッシュに作られるものなので、RIA キャッシュに何か問題があるのではないかと思われます。
今回の例では、MAGIC.INI の RIACacheFilesPath が間違っていた(存在しない場所をさしていた) のが問題の原因で、正しいディレクトリを指定することにより解決しました。

9.5.2. ネットワークモニタ

ネットワークモニタと は、ネットワーク上を流れるパケットをキャプチャするソフトウェアです。
横取り丸との大きな違いは、横取り丸は HTTP プロトコル上を流れる通信パケットに特化したユーティリティであったのに対し、ネットワークモニタは PC 上の特定のネットワークカードを監視して、そこを通過する HTTP 以外のプロトコルのデータも含むすべてのデータをキャプチャできるところにあります。
キャプチャするデータ量を減らすために、フィルタを定義して特定の条件が成立するパケットのみをキャプチャするということもできます。また、キャプチャしたデータに対して、表示のフィルタをかけて、表示内容を絞り込むこともできます。

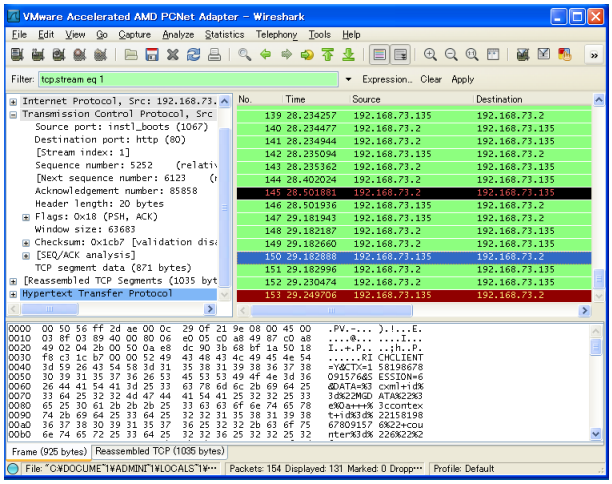
ネットワークモニタの代表的なものとしては、次のようなものがあります。

Microsoft Network Monitor は、Microsoft の Web サイト から無償でダウンロードできるネットワークモニタです。



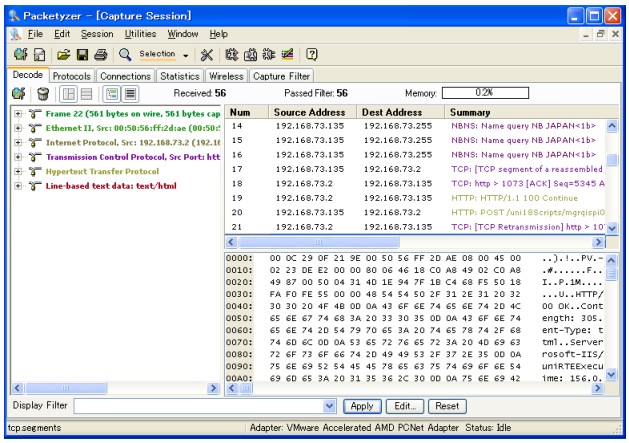
WireShark は、もと Ethereal と呼ばれていたネットワークキャプチャツールで、オープンソースソフトウェアです。機能としては Microsoft Network Monitor と同様で、ネットワーク インターフェース カードを特定して、そこを通るデータをすべてキャプチャします。

Web サイト http://sourceforge.net/projects/wireshark/ からダウンロードすることができます。



Packetyzer は、Ethereal (現 WireShark) の流れをくむネットワークキャプチャツールで、操作法も似ています。WireShark に比べて、パケットの分析機能が多くあります。

現在はアップデートが中止され、商用のソフトウェアに吸収されたようですが、Web サイト http://sourceforge.net/projects/packetyzer/ からダウンロードすることができます。



いずれも機能や操作法は似ているので、いずれか一つを選んで使えばよいでしょう。

9.6. キーボードマクロ

キーボードマクロというのは、PC ユーザのキー操作やマウス操作をキャプチャ・記録して、後で同じ操作を自動で再生することができるユーティリティです。
一般には、決まった操作手順を自動化するために使われるものですが、テスト段階において、同じ操作を繰り返し行ないストレスをかけるようなテスト(リグレッションテスト)に応用することができます。
キー入力のシミュレーションを行うものなので、オンライン、リッチクライアントいずれのプログラムにも利用することができます。

9.6.1. Reckey

Reckey はキーボード入力のみをキャプチャ・再生することができるユーティリティです。マウス操作は利用できませんので、カーソルの移動には TAB キーあるいはショートカットを使い、メニューも Alt+X のショートカットキーを用いることになります。

キャプチャした内容はテキストファイルに格納されるので、適当に手で編集して、ループ構造を持つような独自のスクリプトを作成することも可能です。

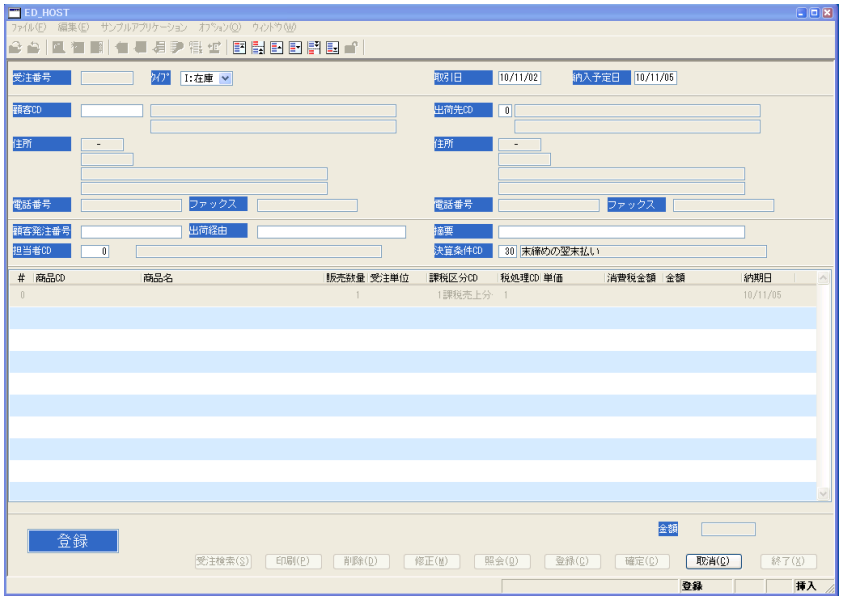
機能的には比較的単純ですが、それだけに覚えるのも簡単で、単純なリグレッションテストに活用することができると思います。

RecKey は、Web ページ http://www.hi-ho.ne.jp/kyagi/ からダウンロードすることができます。

利用例：アプリケーションの定形入力の繰り返し

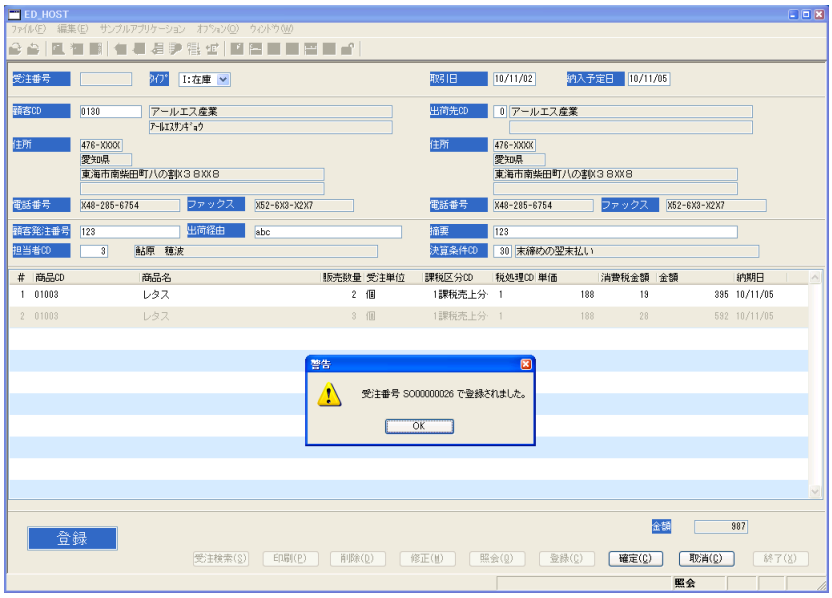
RecKey の利用は、(1) 記録、(2) スクリプトの修正、(3) 再生(実行) という 3 段階で行ないます。

最初に記録を行ないます。
アプリケーションを起動し、テストしたい画面を表示します。



RECKEY を起動し、「記録」ボタンを押します。

キーボードを使って、一連の処理を行い、RecKey に記録させます。
操作では、文字データの入力の他、ズームからの一覧選択やファンクションキー(F3、F4)、Enter キーなども使うことができます。
終わったら、RecKey の記録を停止し、適当な名前でファイルに保存しておきます。



記録の段階で重要なことは、次のことです。

- キーボードのみで操作すること。RecKey はマウス動作を記録できません。従って、ボタンなどは Tab でパークできるようになっているか、あるいはショートカットキーでクリックすることができるようになっている必要があります。
- 最初と最後とを合わせること。例えば、データ入力プログラムであれば、まっさらの初期状態から始まり、一連の入力と確定とを行い、次の入力のためのまっさらの状態で終わる、というようにします。これが合っていないと、繰り返しを正しく行えなくなります。

記録が終わったら、スクリプトの編集を行います。スクリプトの編集は、「編集」メニューを選ぶと、ノートパッドが開きます。

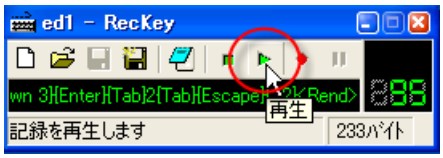
記録直後のスクリプトは、右図のようなものです。
スクリプトの意味については、直観的にも理解しやすいですが、正確には RecKey のマニュアルを参照してください。

```
//*** 修正したら上書き保存してください。<< ed1-0 >>
//*** この行の次から修正してください。
<Active "uniPaaS RunTime","ED_HOST">
{Tab 3}{F5}{Down 2}{Enter}{Tab 2}123{Tab}abc{Tab}qwe{Tab}{F5}{Down 2}
{Enter}{Tab 2}{F5}{Down 2}{Enter}{Tab}2{Down}{F5}{Down 3}{Enter}
{Tab}2{Tab}{Escape}{  2}
```

100 回の繰り返しを行わせるために <REPEAT 100> … <REND> で囲みます。また、キー入力に適当な間隔を置くため、<Interval 40> を入れています。3 秒のウェイトを置きたいところに<Wait 300>を入れました。
編集が終わったら、ノートパッドを閉じます。これで変更が RecKey に反映されます。
ここで、スクリプトをまたファイルに保存してください。

```
//*** 修正したら上書き保存してください。<< ed1 >>
//*** この行の次から修正してください。
<Active "uniPaaS RunTime","ED_HOST">
<Interval 40>
<Wait 300>
<Repeat 10>
{Tab 3}{F5}{Down 2}{Enter}{Tab 2}123{Tab}abc{Tab}qwe{Tab}{F5}{Down 2}
{Enter}{Tab 2}{F5}{Down 2}{Enter}{Tab}2{Down}{F5}{Down 3}{Enter}
{Tab}2{Tab}{Escape}{  2}
<Rend>
```

以上で、スクリプトができあがりです。あとは、「再生」ボタンを押せば、スクリプトに従って、自動的にキー入力が行われるようになります。



RecKey 再生時の注意事項として、次のようなことがあります。

- 再生中はマウスやキーボード等の操作を一切行ってはいけません。RecKey はスクリプトに従って、馬車馬のようにキー入力をどんどんシミュレートして行くだけであり、フォーカスがどのウィンドウにあるかとか、uniPaaS の状態がどうなっているかとかは一切わかりません。このため、マウス操作によりフォーカスのあるウィンドウが変わったりすると、意図していたようなストレステストを行うことができなくなってしまいます。
- 複雑な操作だと、スクリプトが思ったように再生されないことがあります。デバッグ機能はありませんので、再生しながら動作を確認する試行錯誤が必要になります。タイミング的に厳しい場合には、Interval でキー入力間隔に余裕を持たせるとか、Wait で待ちを入れるなどの調整が必要になります。

RecKey の再生は中断することができますが、中断するとキーボードが中途半端な状態になってしまい、Alt キーを押していないのに Alt キーを押したのと同じような状態になってしまうことなどがあります。このような場合には、キーボードの Alt キー、Ctrl キー、Shift キーなどを押して離す、ということを行うと直ることがあります。これらのキーはキーボードの左右にある場合もあるので、両方のキーで行ってみてください。

9.6.2. UWSC

UWSC はシェアウェア（Free バージョンもあり）のキーボードマクロユーティリティで、キー入力とマウス入力とをキャプチャ・再生することができます。http://www.uwsc.info/ からダウンロードすることができます。

BASIC に似たスクリプト言語を持ち、多くの組み込み関数や制御構造があり、かなり機能の高いもので、単純な繰り返しだけでなく、条件分岐やデータを毎回変えることなどもできます。

RecKey と比べると、RecKey は単純な機能しかありませんが、理解しやすく手軽に使うことができます。UWSC は機能が高いので応用範囲が広いですが、理解して使いこなすのにやや壁が高いように思われます。

9.7. その他

9.7.1. Dependency Walker

Dependency Walker は、EXE や DLL モジュールの依存関係をチェックするユーティリティです。Windows の実行可能モジュール (EXE)は、通常、多くの共有 DLL モジュールを呼び出して実行を行うので、もし利用している DLL モジュールがシステムにインストールされていなければ、起動時にエラーとなって、実行することができません。

uniPaaS の場合には、次のような時にこのようなエラーが起こります。

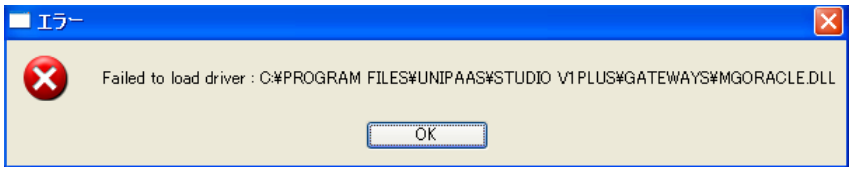
- uniPaaS モジュール (uniStudio.exe や uniRTE.exe など)が必要とする DLL が見つからないとき (誤って削除してしまったとか、必要なサブシステムがインストールされていなかった、など)
- ゲートウェイをロードしようとしたときに、必要な DBMS クライアントモジュールがインストールされていなかったとき。

Dependency Walker は、依存関係のある EXE/DLL ファイルをシステム内で確認し、欠損している場合にはエラーを表示します。

http://www.dependencywalker.com/ からダウンロードすることができます。

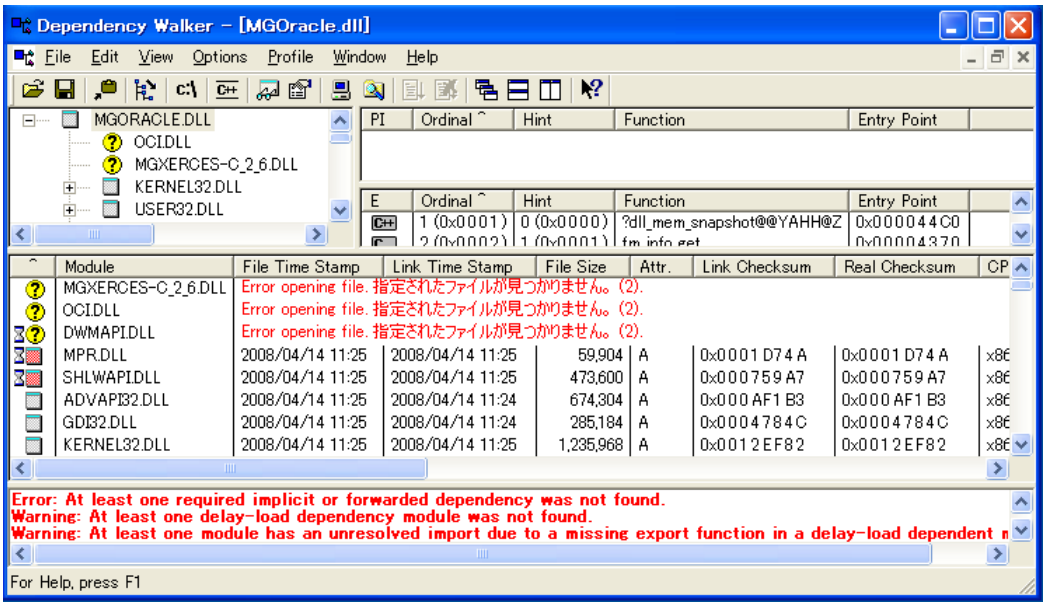
例: ゲートウェイのロードに失敗する場合

MAGIC.INI ファイルで指定されているゲートウェイのロードに失敗すると、下記のようなエラーが uniPaaS 起動時に表示されます。これは Oracle のゲートウェイ (mgoracle.dll) がロードできなかった例です。



このような場合に、ゲートウェイ MGOracle.dll がロードされなかった原因を調べるために、Dependency Walker を使います。

1. ゲートウェイ (エラーメッセージにフルパスが記述されています。通常、uniPaaS のインストールディレクトリの下の Gateways サブディレクトリにあるはずです) が存在するかどうかを確認します。もしなければ、uniPaaS のインストーラを起動して、追加インストールします。
2. あるのにエラーが出ている場合には、Dependency Walker を起動します。
3. Dependency Walker で、ゲートウェイの MGOracle.dll を開きます。→ 次の図のような画面が表示されます。



4. これを見ると、「Error opening file：…」として、3 つの DLL が見つからなかったことがわかります。
   - MGXERCES-C_2_6.DLL というのは、uniPaaS インストールディレクトリにあり、uniStudio.exe/uniRTE.exe も利用するものです。このファイルは実行時には uniStudio/uniRTE でロードされているので、Gateway ディレクトリで見えなくても大丈夫です。
   - OCI.DLL というのは、Oracle のクライアントモジュールです。これが欠損しているということは、Oracle のクライアントモジュールがインストールされていないか、インストールされていても PATH が通っていないために見つけることができなかったことを意味します。Oracle のクライアントモジュールを再インストールすることで解決できるかもしれません。
   - DWMAPI.DLL というものも欠損していますが、これはシステム内部で使われているものであり、また、「delay-load」という特性を持っているものなので、見つからなくても実際上は問題になりません。

第10章　プログラミングの小技

uniPaaS アプリケーション実行時の問題調査やデバッグは、今まで説明してきたようなツールを使うことにより効率化することもできますが、uniPaaS アプリケーションにちょっとした工夫をすることにより更に効率化することもできます。
本章では、そのような uniPaaS プログラムでの小技をいくつか紹介します。

ここで示したプログラムは、弊社 HP スキルアップセンターよりダウンロードできます。

10.1. Logging 関数

第 2 章「uniPaaS 実行エンジンの出力ログ」で、アクティビティモニタによる問題追跡の方法について説明しました。アクティビティモニタに出力する内容は、「ロギングダイアログ」(2.5「ロギングダイアログでフィルタを設定するには？」参照) で設定しますが、ここで設定するとアプリケーション実行の最初から最後まで、この設定に従ってログが出力され、不要な部分のログが大量に出てきます。そうなると本当に見たいところを大量のログの中から見つけるのが難しい、実行速度が遅くなり調査に支障を来す、などが問題になることもあるでしょう。

Magic uniPaaS では、Logging 関数を使って、記録するログの内容を、プログラムから動的に変更することができるようになりました。これにより、ログ取りの不要な部分の実行中はログをオフにしておき、本当に必要な部分に入ったときに有効にする、というようなログレベルのコントロールが可能になります。
この関数を実行すると、即時に変更内容が有効になります。MAGIC.INI やロギング ダイアログでの設定には影響を与えません。

構文: Logging (開始/停止, フィルタ)

パラメータ:

- 開始/停止 … フィルタの開始／停止を指定する論理値
  - True…フィルタを開始します。
  - False … フィルタを停止します。
- フィルタ(文字) … フィルタオプションを表す文字列。以下のオプションが指定できます。

Task	Levels	DataView
Recompute	Flow	Events
LogBrowser	Gateway	TransCache
BackgroundMsg	BeginEndMsg	LogSynch
ALL	RESET	

ExecutionLogFileName=(ファイル名)

Btrieve= (N,D,S,C のどれか)

DB2400= (N,D,S,C のどれか)

Oracle= (N,D,S,C のどれか)

AS400= (N,D,S,C のどれか)

ODBC= (N,D,S,C のどれか)

MicrosoftSQLServer = (N,D,S,C のどれか)

Memory= (N,D,S,C のどれか)

戻り値: 論理値 … 処理が成功した場合「True」が返ります。

例: Logging ('FALSE' LOG,' Oracle')

注意事項:

- [フィルタ]の指定が正しくない場合、「False」が返ります。
- 「ExecutionLogFileName」で外部ログファイル名が設定され、ファイルが作成できなかった場合、「False」が返ります。メッセージは mgerror.log に書き込まれます。(ログファイル名は、MAGIC.INI ファイルの GeneralErrorLog パラメータで変更できます。)
- 外部ログが設定されず、デバッグモードも設定されていない場合、「False」が返ります。
- [開始/停止]パラメータが「True」で DBMS パラメータが「N」と評価された場合 (例:' True' ,' ODBC=N')、ゲートウェイオプションは無視され、「False」が返ります。

- 開発エンジンがデバッグモードでない場合、(指定されていれば)外部ログファイルのみに反映されます。
- この関数は、MAGIC.INI を更新しません。設定された値は、現在のコンテキストでのみ有効です。開発エンジンに戻ると、値は MAGIC.INI に設定されているデフォルト値に戻ります。
- [フィルタ]パラメータの「ALL」が停止(False)とともに使用された場合、全ての値は False に設定されます。
- [フィルタ]パラメータの「ALL」が開始(True)とともに使用された場合、全ての非ゲートウェイ値は True に設定されます。
- 「RESET」フィルタは、INI の値を全てリセットします。この場合、第一パラメータは、True とします。False が設定された場合、無視され「False」が返ります。

この関数を使うと、アクティビティモニタの出力内容をコントロールできるので、欲しい情報だけを的確に採取することができるようになります。

例: ゲートウェイログ出力を特定のタスク実行中にだけ限定する

例として、あるタスクに関してだけ、ゲートウェイのログを調査したいときがあるとします。「ロギング」ダイアログや MAGIC.INI の設定でゲートウェイログを出力するようにしておくと、アプリケーションの最初から最後まですべてゲートウェイのログがアクティビティモニタやログファイルに記録されるようになり、複雑なロジックのアプリケーションでは、膨大なログ出力となることがあります。
しかし、本当にゲートウェイログを採取したいタスクが特定されている場合には、次のようにして、そのタスクの実行期間中だけログを記録するようにできます。

- 「メインプログラム」のタスク前処理で、Logging ('FALSE'LOG, 'ALL') を実行し、すべてのログ出力を抑制します。
- ログを記録したいタスクのタスク前処理で、Logging ('TRUE'LOG, 'Gateway')、および Logging('TRUE'LOG, 'MicrosoftSQLServer=D') を「アクション」コマンドで実行します。(DBMS が SQL Server で、ログのレベルが D (開発者) の場合)
- タスク後処理で、再度 Logging ('FALSE'LOG, 'ALL') を実行し、ログ出力を抑制します。

10.2. FlwMtr 関数

実行の記録のために、アクティビティモニタに任意のメッセージを記録しておくことができます。これには FltMtr 関数を使います。

構文: FlwMtr (文字列, 論理ブレイク)

パラメータ:
- 文字列 (文字型/Unicode 型) … アクティビティモニタに表示させる文字列
- 論理ブレイク (論理型) … V1Plus では無効です。

戻り値: 常に False が返ります。

FlwMtr 関数を実行すると、指定した「文字列」がアクティビティモニタに記録されます。アクティビティモニタには、他のログ (タスク起動・終了、フロー、DBMS 活動) なども記録されるので、例えば「このコマンドはどのタイミングで実行されるのか？その時の項目の値は何か？」などを調べたい場合に、デバッグ用に FlwMtr 関数を使って記録をすることができます。

リファレンスマニュアルでは、第 2 パラメータ「論理ブレイク」として、'TRUE'LOG の場合にデバッグブレークがかかるとありますが、V1Plus では無効になっています。

10.3. 項目の表示プログラム

uniPaaS のプログラムを実行している時、特定の時点での変数の値を見てみたい、と思うことがあります。
デバッガが利用できる場合には、ブレークを入れて、「項目」画面を表示させることにより、その時点で有効な項目の値を見ることができます。しかし、(1) ブレークを入れたくない場合、(2) デバッガが利用できない場合、(3) ファイルなどに記録をとっておきたい場合、などでは別の方法を考える必要があります。
このような場合には、ちょっとした uniPaaS プログラムの作成が必要になりますが、Var 関数を使って実現することができます。

プログラム「項目出力」は次のように作ります。

タスク特性	タスク名: 項目出力 タスクタイプ: バッチ 初期モード: 修正 タスク終了条件: Counter(0) >= 'A'VAR
入出力ファイル	名前: Out メディア: F=ファイル アクセス: A=追加 書式: L=ライン 式: 'var.log'
データビュー	変数 A「this counter」数値型、書式 8
ロジック/レコード後処理	項目更新 A = Counter(0) フォーム出力 O=出力、フォーム: 4 (out line)、ファイル: 1 (Out)
フォーム/4 out line	クラス: 1 インターフェースタイプ: T=テキスト形式 幅: 230、高さ: 1 項目1: (0,0) - (30,1)、型: U=Unicode、書式: 30、データ= VarName(a) 項目2: (31,0) - (200,1)、型: U=Unicode、書式: 200、 データ: CASE (VarAttr (A), 'A', VarCurr(A), 'N', Str(VarCurr(A),'N16.2'), 'L', IF(VarCurr(A),'TRUE','FALSE'), 'D', DStr(VarCurr(A), 'YY/MM/DD'), 'T', TStr(VarCurr(A), 'HH:MM:SS'), 'U', VarCurr(A), '(type:' & VarAttr(A) & ')')

Var 関数を簡単に説明すると、VarAttr は項目の型を 1 文字で返します (文字型 → 'A'、数値型 → 'N' など)。
また、VarCurr は、項目の現在の値を返します。値の型は、パラメータで指定した項目の型そのものです。

「項目出力」プログラムを簡単に説明すると、このプログラムでは、VarCurr(A) の値をファイルに出力します。
フォームの項目2がその値ですが、この式が CASE を使った複雑なものになっているのは、VarAttr (A) のタイプが文字型とは限らないので、文字型に変換するために、それぞれの型に合った組み込み関数と書式とを使っているためで、やりたいことは、VarCurr(A)の値を取得することです。

このバッチタスクはループを作りますが、その終了条件は Counter(0) >= 'A'VAR となっています。

これだと、「このプログラムは、自分のタスクの変数 A の値を出力するだけではないか？変数 A の値は数値型

で Counter(0)に代入されているが、終了条件が　Counter(0) >= 'A'VAR とはどういうことだ？」と思われるかもしれません。この謎を解く鍵は、次の二つの Magic ルールにあります。

1. VAR リテラルの値は、実際には数値です。例えば、'A'VAR = 1、'B'VAR = 2、… というように、'A' を起点とする連続整数値となります。
2. 実行時には、VAR リテラルの値は、メインプログラムから現在実行中のタスクまでのすべてのタスクで定義されているデータ項目を積み重ねて、最初(メインプログラムの最初の項目)を1と数えた連続数値となっている。

1は簡単に理解できると思いますが、2 は少し説明が必要です。

例として、タスク A、B、C があり、それぞれ、2 個、2 個、4 個の項目がデータビューに定義されているとします。

そして、タスク A が B を、B が C を呼び出したとします。

この状態で、上記の「項目出力」タスクを呼び出します。

このとき、VAR リテラルの値は、右図のようになっています。

これからわかるように、「項目出力」タスクの 'A'VAR は、この状態では 9 になっています。

タスク	変数名	VAR リテラル値
A	今日の日付 今の時刻	1 2
B	N A	3 4
C	ProductID ProductName CategoryID UnitPrice	5 6 7 8
項目出力	this counter	9

VAR リテラルの値についてのルールを理解すれば、終了条件 Counter(0) >= 'A'VAR は、この場合 Counter(0) >= 9 と書いたのと同じで、Counter(0) が 1 ～ 8 まで、8 回のループが走ることがわかります。

レコード後処理では、Counter(0) の値を A に代入して、VarCurr(A) を出力しますから、ループで VarCurr(1) ～ VarCurr(8) までについて出力がされます。このようにして、「項目出力」タスクは、その時点で有効な項目をすべて出力できるようになります。

ファイルに出力された結果は、右図のようになります。

変数項目.今日の日付	10/11/02
変数項目.今の時刻	18:37:36
NA.N	1.00
NA.A	aaabbc
Products.ProductID	6.00
Products.ProductName	Grandma's Boysenberry Spread
Products.CategoryID	2.00
Products.UnitPrice	25.00

この項目表示プログラムは、コンポーネント化しておけば、デバッグしているアプリケーションから簡単に呼び出すことができるようになります。

10.4. コマンドラインリクエスタのログ読み込みプログラム

uniPaaS のコマンドラインリクエスタ (Mgrqcmdl.exe) のログ出力機能により、実行されたリクエストについての情報を一覧で取得することができます(4.1.4「個々のリクエストについて表示させるには？」参照)。
その出力フォーマットは、次のようなものです。

```
C:¥Program Files¥uniPaaS¥Enterprise Server V1Plus>mgrqcmdl -query=log

                Log of requests  (MyServer/5215, 10:11:21)
  ------------------------------------------------------------------------------

#   Request  Status          Start      Elapsed     Completion Codes
      Id                     Time       (sec.)    (Mri, Runtime, Dbms)
================================================================================
1  | 22   |DONE        | 16/11 09:39:42 | 0    | -OK (0) 0 47
Program  : "NA_MAIN in sub (update fail)" ("subtask-link")
Priority : 0
Client : 00000000EF42B255_D (pid 5656),   EnterpriseServer : MyServer/1501
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
2  | 21   |DONE        | 16/11 09:39:41 | 1    | -OK (0) 0 47
Program  : "NA_MAIN in sub (update fail)" ("subtask-link")
Priority : 0
Client : 00000000EF42B255_D (pid 5656),   EnterpriseServer : MyServer/1501
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
・・・  (以下省略)
```

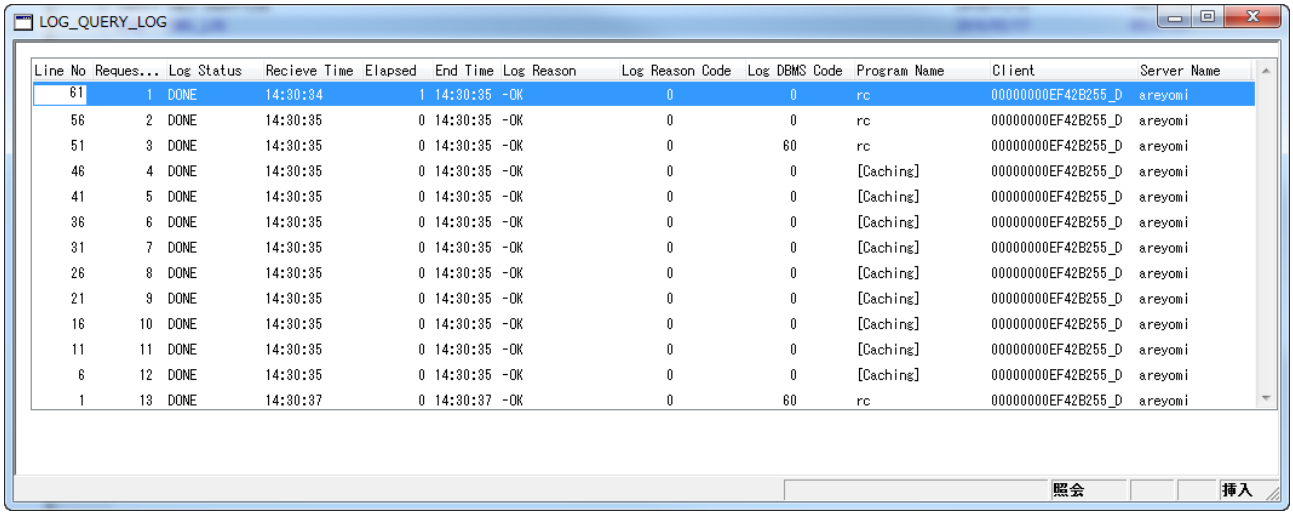
この出力はテキストファイルに格納されますが、データベース化することにより、解析が容易になります。例えば、実行時間 (Elapsed sec.) が長い順にリクエストを表示させたいとか、あるいはクライアント(Client) 別にリクエストをグループ化して調査したい、というような場合、テキストファイルのままでは難しいですが、データベース化することで簡単になります。
このようなデータベース化プログラムは、uniPaaS で簡単に作成することができ、基本的にはテキストファイル読み込み → 行の解析 → レコード作成、という処理になります。
ログファイルは、次の5行が1リクエストに対応していて、このパターンの繰り返しとなっています。

```
================================================================================
1  | 22   |DONE        | 16/11 09:39:42 | 0    | -OK (0) 0 47
Program  : "NA_MAIN in sub (update fail)" ("subtask-link")
Priority : 0
Client : 00000000EF42B255_D (pid 5656),   EnterpriseServer : MyServer/1501
```

それぞれの項目の意味は次の通りです。

```
================================================================================
<#> | <リクエスト ID>   |<ステータス> | <日付> <時刻> | <処理時間> | <結果ステータス>
Program  : "<プログラム名>" ("<アプリケーション名>")
Priority : <優先度>
Client : <クライアント ID> (pid <プロセス ID>), EnterpriseServer : <サーバホスト名>/<サーバポート番号>
```

各項目は固定長ではないので、各行から InStr、StrTok などの関数を使って項目値を切り出すようになります。uniPaaS プログラムとしては簡単なプログラムですので、詳しいことは実際のプログラムを見てください。
次の図は、このログ読み込みプログラムを使って読み込んだ例です。このように uniPaaS でデータベース化してしまえば、簡単なものはオンラインプログラムの実行時オプション（位置づけ、範囲付、ソート）で並び替えができるし、少し複雑なロジックを必要とするようなものでも、uniPaaS プログラムを作って対応することができるようになります。また、DataViewToText/DataViewToXML 関数などを使って、CVS ファイルや XML ファイルに出力して、Excel などで分析することもできます。



Line No	Reques...	Log Status	Recieve Time	Elapsed	End Time	Log Reason	Log Reason Code	Log DBMS Code	Program Name	Client	Server Name
61	1	DONE	14:30:34	1	14:30:35	-OK	0	0	rc	00000000EF42B255_D	areyomi
56	2	DONE	14:30:35	0	14:30:35	-OK	0	0	rc	00000000EF42B255_D	areyomi
51	3	DONE	14:30:35	0	14:30:35	-OK	0	60	rc	00000000EF42B255_D	areyomi
46	4	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
41	5	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
36	6	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
31	7	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
26	8	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
21	9	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
16	10	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
11	11	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
6	12	DONE	14:30:35	0	14:30:35	-OK	0	0	[Caching]	00000000EF42B255_D	areyomi
1	13	DONE	14:30:37	0	14:30:37	-OK	0	60	rc	00000000EF42B255_D	areyomi

10.5. アクティビティモニタ出力の読み込みプログラム

アクティビティモニタの出力 (2.1「アクティビティ モニタとは何ですか？」以下を参照)も、テキストファイルからデータベース化することにより、分析が容易になります。このような読み込みプログラムも、uniPaaS で簡単に作成することができます。
アクティビティモニタの出力例を以下に示します。

```
<7819595268872 > 09:39:27.422    – >> 情報 >>  IO file 'SUPPORT¥html_std.jpn' cannot be OPENED
<15639190537744> 09:39:33.575    – >> Calling task 'rc1' ('subtask-link') : Session Counter #4 (Request ID #3)
<15639190537744> 09:39:33.579    – >> 情報 >>  IO file 'SUPPORT¥html_std.jpn' cannot be OPENED
<15639190537744> 09:39:33.610    –  >>開始 ﾛｰﾄﾞ ﾊﾞｯﾁ ﾀｽｸ – 'メインプログラム (subtask-link)'  照会 ﾓｰﾄﾞ
<15639190537744> 09:39:33.612    –  終了 ﾀｽｸのﾛｰﾄﾞ
<15639190537744> 09:39:33.614    –  ﾚｺｰﾄﾞ読込
<15639190537744> 09:39:33.675    –   >>開始 ﾛｰﾄﾞ ﾘｯﾁｸﾗｲｱﾝﾄ ﾀｽｸ – 'NA_MAIN in sub (update fail)'  修正 ﾓｰﾄﾞ
<15639190537744> 09:39:33.676    –   終了 ﾀｽｸのﾛｰﾄﾞ
<15639190537744> 09:39:33.679    –   ﾚｺｰﾄﾞ読込
<15639190537744> 09:39:33.682    –   新規の遅延ﾄﾗﾝｻﾞｸｼｮﾝ書込 2
<15639190537744> 09:39:33.683    –   開始 ﾚｺｰﾄﾞ前
<15639190537744> 09:39:33.684    –   終了 ﾚｺｰﾄﾞ前
<15639190537744> 09:39:33.730    –   >> .....  Completed Request ID #3 (0.16 seconds) .....
<15639190537744> 09:39:33.731    –   >> .....
<15639190537744> 09:39:33.789    –   >> 情報 >> Request's Session Counter #0 <> Server's Session Counter #4
<15639190537744> 09:39:34.447    –   >>  ｲﾍﾞﾝﾄ処理中  [ｺﾝﾃｷｽﾄがﾌｫｰｶｽ喪失(L)]  ｺﾝﾄﾛｰﾙ名 :  [ 無効 ]
<15639190537744> 09:39:34.462    –   >> Calling task 'rc1' ('subtask-link') : Session Counter #5 (Request ID #13)
<15639190537744> 09:39:34.001(c) –    Starts Record Prefix
<15639190537744> 09:39:34.004(c) –      Flow – (Step Forward)
<15639190537744> 09:39:34.475    –      ﾌﾛｰ – ｺｰﾙ S=ｻﾌﾞﾀｽｸ :  NA_MAIN  (通常ﾓｰﾄﾞ 前方)
<15639190537744> 09:39:34.507    –   4,84627  ms7_ini(): >>>>>
<15639190537744> 09:39:34.509    –   ,84627  ms7_ini(): MicrosoftSQLServer , Version uniPaaS 1.8 SP1a PT1-0 19-Jan-2010
<15639190537744> 09:39:34.510    –   ,84627  Version uniPaaS 1.8 SP1a PT1, Log = mgmonitor.log,  Level = Support/QA,  Sync = Reopen,
<15639190537744> 09:39:34.512    –   ,84627  ms7_ini(): <<<<< retcode = 36
```

この出力は、基本的に1行1レコードに対応し、フォーマットは次のようなものです。

```
<コンテキスト ID> <時刻> <メッセージ>
```

1行づつ読み込み、このフォーマットに従ってレコードを作成していく、というようなプログラムになります。
ただし、アクティビティモニタの出力はばらつきがあり、データベースとして標準化するために細かな加工が必要になります。

- コンテキスト ID: コンテキスト ID は「<」と「>」とで囲まれた数値ですが、桁数はまちまちであり、可変長として切り出す必要があります。また、「-1」というものもあります。
- 時刻: ミリ秒(小数点以下 3 桁)の単位まで表示され、12 桁固定です。
- メッセージ: この部分は、メッセージの内容により形式がかなり違います。
- 情報、タスクやイベントの開始/終了、フローの実行などは、「–」記号の後に、複数の空白文字でインデントされています。解析する上でインデントは普通要らないので、「–」記号と合わせて、LTrim で削除してしまいましょう。
- ゲートウェイの出力では、「–」記号の後にもう一つの時刻が記録されています。例えば、

  ```
  <15639190537744> 09:39:34.507    –   4,84627  ms7_ini(): >>>>>
  ```

  の「4,84627」の部分です。これはこれで有用な情報なのですが、メッセージとは切り離して別項目として格納した方が良いでしょうから、切り離します。
- リッチクライアントタスク実行中には、クライアント側の活動ログも併せて記録されます。クライアント側のログは、次の例のように、<時刻> の直後に「(c)」という記号で区別されています。

```
<15639190537744> 09:39:34.001(c) -     Starts Record Prefix
```

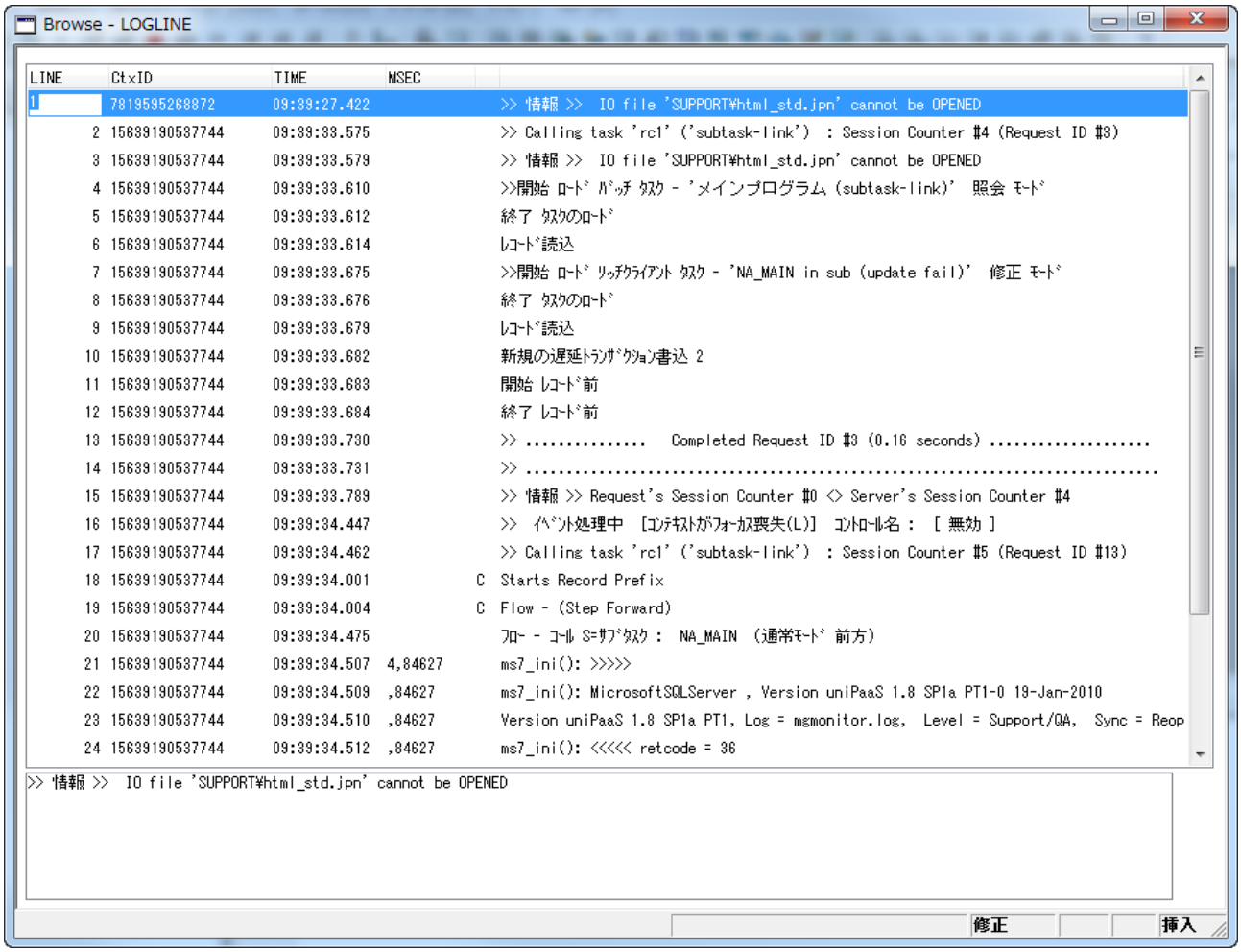
これもまた有用な情報なので、メッセージとは切り離して、別項目として格納します。

- データベースゲートウェイの出力で、改行コードがメッセージの中に入っているものがあります。例えば、次のようなものがあります。

```
<15639190537744> 09:39:35.672     -      ,85782  ms7_esqlc_fil_exist():
      select id from MAGICDB..sysobjects where name = 'NA_LINK' AND uid = user_id('') AND (type = 'U' OR type = 'V' OR type = 'S')
```

このような場合には、2行以上を1行に連結してやる必要があります。連結の必要があるか否かは、行頭に「<…>」の形式のコンテキスト ID があるかどうかをチェックして確認します。コンテキスト ID が認識できない行は、前行からの継続行として、前行のデータに連結します。

以下に、このプログラムでデータベース化したログのイメージを示します。



このようにデータベース化することにより、例えば、次のようなことができるようになります。

- コンテキスト ID 15639190537744 の活動だけを表示させたい：CtxID カラムで範囲付けを行います。
- テーブル NA_MAIN の活動だけを表示させたい：「InStr (G,'NA_MAIN') > 0」という式で範囲付けします。
- タスクの実行順序を調べたい：「>>開始 ﾛｰﾄﾞ」で始まる行のみ表示させます。

Magic uniPaaS V1Plus
トラブル シューティング ツール

第 2 版　2011 年 2 月 8 日
発行　〒151-0053 東京都渋谷区代々木三丁目二十五番地三号
あいおいニッセイ同和損保新宿ビル 14 階
マジック ソフトウェア・ジャパン (株)
http://www.magicsoftware.co.jp/