文書番号: LK20190524-254-003

LifeKeeper for Windows
汎用アプリケーションリソース開発の手引

SIOS

目次

改版履歴

2014 年 12 月 26 日　　第 1 版

2019 年 5 月 24 日　　第 2 版

監視処理以外のスクリプトに対するタイムアウト処理の実装例を追記

2022 年 1 月 19 日　　第 3 版

bash の記述を削除（マニュアルに合わせる）

1 はじめに

本文書は、LifeKeeper for Windows において汎用アプリケーションリソースを作成する際に、参考資料としてご利用いただくことを想定した作成ガイドです。

2 汎用アプリケーションとは

LifeKeeper は、保護対象のリソースを簡易に保護するための仕組みとして、Application Recovery Kit(以下「ARK」という。)の機能を提供しております。ARK は、保護対象のアプリケーション毎に用意されており、これらを用いることで非常に少ない作業工数でアプリケーションの冗長化を実現することが出来ます。

しかし、全てのアプリケーション用の ARK が用意されているわけではありませんので、お客様が使用したいアプリケーションの種類によっては、ARK が存在しない場合もあります。また、既存の ARK ではお客様の期待する動作要件を満たさない可能性もあります。そのような場合は、LifeKeeper Core に含まれている、Generic Application Recovery Kit を用いて、そのアプリケーションを保護する必要があります。

3 汎用アプリケーションとスクリプト

3.1 作成するスクリプトの種類

汎用アプリケーションを利用するには、保護対象のアプリケーションを制御するためのスクリプトを作成する必要があります。また、複数の汎用アプリケーションを登録することも可能です。その場合はスクリプトも、登録する汎用アプリケーションの数だけ作成してください。作成するスクリプトの種類と、スクリプト内に記述するべき機能を以下に示します。

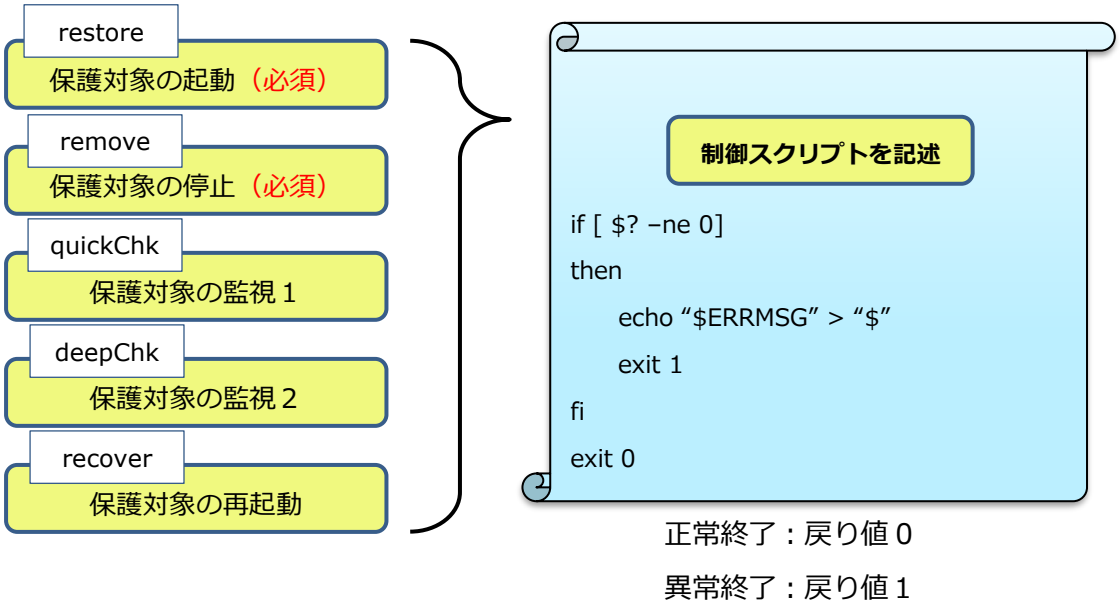
スクリプト	処理概要
restore(必須)	保護対象アプリケーションの起動を行います。
remove(必須)	保護対象アプリケーションの停止を行います。
quickchk(任意)	クイックチェックを実行し、保護対象アプリケーションが正しく動作しているかを確認します。
deepchk(任意)	詳細チェックを実行し、保護対象アプリケーションが正しく動作しているかを確認します。
recover(任意)	監視において障害を検知した場合、自ノード上で回復処理を試みます。

LifeKeeper for Windows では、クイックチェックと詳細チェックの２種類の監視処理を備えています。Generic ARK で設定するリソースでも監視を実施し、障害検知によるフェイルオーバをさせたい場合は、クイックチェックや詳細チェック用のスクリプトを作成する必要があります。

関数などが格納されたスクリプトを別途作成し、上記の各制御スクリプトから呼び出すことも可能です。但し、上記の４種類のスクリプト以外は、LifeKeeper の GUI 管理画面から登録することは出来ないため、外部ファイルは予め手動で適切な位置に配置しておく必要があります。

3.2 使用できるスクリプト言語

汎用アプリケーションの制御用スクリプトは、ShellScript(ksh)、Perl、VB Script のいずれかを使用することができます。また、1 つのリソースを作成するために使用する制御用スクリプトはどれか 1 つに統一して作成してください。たとえば、restore スクリプトに ShellScript、remove スクリプトに VB Script を使用するといったことはできません。



3.3 スクリプトの要件

汎用アプリケーションを組み込むスクリプトは、実行が完了した時点で正常/異常に対する戻り値を返す必要があります。戻り値は必ず以下のいずれかを返すように作成してください。

- **正常に終了　　=>　戻り値 0**
- **異常終了(障害検知)　=>　戻り値 1**

エラーの種類によって異なる戻り値を返すような仕組みは、正常に動作する保証がありません。エラーのステータス確認は、ログにのみ出力するなどして、スクリプト自体の戻り値は上記のルールを厳守してください。この点以外に LifeKeeper 固有の特別な機能要件はありません。どのような内容を記述しても、最終的に正常終了（0）か異常終了（1）となるように作成されていれば、仕様上は問題ありません。

尚、汎用アプリケーションで設定するスクリプト経由で、LifeKeeper 自体の操作を行うこと自体は禁止しておりませんが、要件に見合った動作となるかについては保証しておりません。操作を行った場合の影響範囲や挙動については、実機での動作検証等からご確認頂く必要があります。

4　汎用アプリケーションの仕様

4.1　クイックチェックと詳細チェックの実行とタイムアウト

汎用アプリケーションに登録したリソースは他の ARK と同様にリソース作成時に監視間隔を分単位で指定します。LifeKeeper の監視は、lkresmon.exe というプロセスが設定された監視間隔毎に監視スクリプトを実行します。Lkresmon.exe は監視スクリプトの起動と停止を管理していますが、それ以外に一定の時間内に各監視スクリプトが終了しているかを確認しています。

もし終了していない場合にはその監視スクリプトはタイムアウトとなり、lkresmon.exe がその監視プロセスを強制的に終了させます。タイムアウト値が過ぎたことによって強制的に監視が終了した場合は、フェイルオーバは行われません。[1] 次回の実行タイミングで再び監視が行われます。

Lkresmon.exe が監視プロセスを強制的に停止するまでの時間が、その監視スクリプトのタイムアウト値になります。監視スクリプトのタイムアウト値は、以下のように決められています。

- クイックチェックのタイムアウト値（秒）＝
  クイックチェックの監視間隔（秒）× MAX_QUICK_CHECK(1 回)
- 詳細チェックのタイムアウト値（秒）＝
  詳細チェックの監視間隔値（秒）× MAX_DEEP_CHECK（2 回）

たとえば、クイックチェックの監視間隔を 3 分とした場合、クイックチェックスクリプトのタイムアウト値は 180 秒 × 1 回 = 180 秒となります。従って、監視スクリプトの処理は、この時間内に終了するように作成する必要があります。MAX_QUICK_CHECK と MAX_DEEP_CHECK の値は、レジストリ内で定義されており、この値を変更することはできません。

尚、タイムアウトが発生した場合は、イベントビューアのアプリケーションログに以下のようなメッセージを確認することができます。

[1] デフォルトでは再起動などは行いませんが、タイムアウト発生時のサーバのリソース状態の記録や、タイムアウト発生時にフェイルオーバやサーバの再起動を実行させるように設定を行うことも可能です。詳細については LifeKeeper for Windows のドキュメントサイトの情報をご参照ください。

```
Process: LKRESMON.EXE(PID)  *WARNING* (No. 1517) System Failure Detected! quick Check process (PID=2200) did not return in タイムアウト値(秒) seconds for resource リソース名 Aborting check process. (Partner PID=0)
```

上記はクイックチェックの例です。詳細チェックの場合には、"quick check"の部分が"deep check"と出力されます。

4.2 監視スクリプト以外のタイムアウトについて

LifeKeeper は監視処理を除く restore、remove、recover の各処理に対してタイムアウト機能を提供していないため、ハングアップに備えてこれらの処理に対するタイムアウトの仕組みをスクリプトで実装すべきです。以下にタイムアウトのサンプルスクリプトを記載致します。

本例は、以下パスに格納されている Windows サービスを保護するサンプルスクリプトを例としています。

C:¥LK¥Admin¥kit¥app¥templates¥example_service¥restore.pl

```
#!perl

sub serviceControl {
  my $action = shift;
  my $serviceName = shift;
  my $serviceState;
  my $maxWait;
  my $cTime;
  my $goodState = 0;

  # タイムアウト設定
  $maxWait = 300;

  #
  # サービスの状態を取得し、値を設定
  #

  if (lc($action) eq "start") {
    $goodState = 4;
  }

  if (lc($action) eq "stop") {
    $goodState = 1;
  }
```

```
  ($goodState == 0) && return 1;

  #
  # 既にサービスが起動していれば正常終了
  #

  $serviceState = getServiceState($serviceName);
  if ($serviceState == $goodState) {
    return 0;
  }

  `sc $action $serviceName >NUL: 2>&1`;  # sc コマンド実行

  # サービスが起動完了するまで待機する。コマンド実行時間が maxWait で
  # 設定した秒数を超過した場合、異常終了する(return 1)

  $cTime = time;

  while ((time < $cTime + $maxWait) && (getServiceState($serviceName) !=
$goodState)) {
    sleep 5;
  }

  (getServiceState($serviceName) == $goodState) && return 0;

  # タイムアウト時の処理

  return 1;
};
```

4.3 汎用アプリケーションスクリプトの変更

LifeKeeper に汎用アプリケーションを登録する際にスクリプトの PATH を指定しますが、このとき LifeKeeper は各スクリプトを自身のデータベースにコピーし、以降のスクリプトの実行はコピーしたファイルを使用します。元のファイルは参照しないため、制御内容の変更を元のファイルに記述したとしても、LifeKeeper 上の操作には反映されません。制御スクリプトの内容を変更したい場合は、以下の 2 通りの方法があります。

1. 既存の汎用アプリケーションリソースを削除し、処理内容を変更した新しいスクリプトで再度汎用アプリケーションリソースを作成する。

2. LifeKeeper が取り込んだスクリプトのコピーを直接変更する。

   ① GUI から変更する

   LifeKeeper GUI より、”スクリプト・アップデート”の機能を使ってスクリプトの更新を行うことが可能です。詳細については、以下のドキュメントサイトにてご案内しておりますのでご参照ください。

   Generic Application スクリプトの設定

   ② ファイルを直接編集する

   以下のパスにあるファイルを直接編集することでもスクリプトの更新を行うことが可能です。この方法を採る場合は、汎用アプリケーションリソース設定している全てのノード上で同じ変更作業を行う必要がございます。

   - restore スクリプト
     C:¥LK¥Subsys¥gen¥resources¥app¥actions¥!remove¥$TAG
   - remove スクリプト
     C:¥LK¥Subsys¥gen¥resources¥app¥actions¥!restore¥$TAG
   - quickchk スクリプト
     C:¥LK¥Subsys¥gen¥resources¥app¥actions¥!quickchk¥$TAG
   - deepchk スクリプト
     C:¥LK¥Subsys¥gen¥resources¥app¥actions¥!deepchk¥$TAG
   - recover スクリプト
     C:¥LK¥Subsys¥gen¥resources¥app¥actions¥!recover¥$TAG

注記 : $TAG は汎用アプリケーションリソースに対して与えたタグ名(リソース名)です。作成したスクリプトファイル名とは関係無く、常にタグ名で保存します。

5　汎用アプリケーションリソースの作成

5.1　保護対象アプリケーションの調査と検討事項

スクリプトの設計を行う前に、以下の様な要素に対して予め調査、検討を行う必要がございます。

- アプリケーションの切替は必要か
  仮想 IP やアプリケーションが使用するファイルシステムの切替のみでは不足があるかを確認する必要があります。

- アプリケーションの動作に必要な要素は何か
  保護対象が他のリソースに依存する場合や、LifeKeeper のデフォルトの機能では保護対象にしないサービスやアプリケーションとも依存する場合は、それらも含めて保護対象とするかを検討する必要があります。

- ノード固有の情報は必要か
  保護対象とするアプリケーションが、ノード固有の情報を使用するかを確認します。ノード固有の情報が正常な動作に必要となる場合は、その情報を引き継ぐ仕組みをスクリプトに記述する必要があります。

- アプリケーションの制御に対話型インターフェースが介在するか
  対話型インターフェースで行っていた処理の場合は、応答のやりとりも含めてスクリプトに実装する必要があります。

- アプリケーションの機能と汎用アプリケーションリソース数の検討
  保護するアプリケーションが複数の機能に分かれている場合は、各機能を個別にリソース化して保護するか、１つの汎用アプリケーションでまとめて保護するかを検討する必要があります。

5.2 スクリプト作成における検討事項

汎用アプリケーションのスクリプトは、戻り値が 0 か 1 になるよう作成すれば、他はどのように記述頂いても問題はございません。アプリケーションの仕様や運用設計に従ってスクリプトを設計してください。本項では、LifeKeeper が各スクリプトを実行する際の動作について解説致します。

- 監視スクリプト以外のタイムアウトについて
  restore、remove、recover についてはタイムアウトの設定が備わっていません。これらの処理にタイムアウトが必要であれば、その仕組みをスクリプト内で実装する必要があります。タイムアウト機能の実装例は、**4.2 監視スクリプト以外のタイムアウトについて** を参考としてください。

- スクリプトのリトライについて
  restore、remove、recover、quickchk、deepchk の各処理は、スクリプト自身をリトライを行う仕組みはございません。複数回の確認を行ってから成否判定を行う必要がある場合は、リトライの仕組みをスクリプトに実装する必要があります。

- ２重起動の防止について
  アプリケーション自体に２重起動を防止する機能がない場合は、２重起動を防止する仕組みをスクリプトに実装する必要があります。

- lkstop –f への対応について
  LifeKeeper を”lkstop -f”で停止すると、各リソースの remove スクリプトをスキップし、LifeKeeper のみ停止を行い、保護対象としているサービスは起動したままになります。この状態で”lkstart”から LifeKeeper を起動した場合、各リソースが既に起動状態にあることを検知して起動処理をスキップして正常終了します。汎用アプリケーションリソースを設定する環境で、”lkstop –f を実行する運用の場合は、同様の配慮が必要となります。

- スクリプトの実行権限について
  LifeKeeper の各スクリプトは LOCAL SYSTEM アカウントで動作するため、スクリプト内で実行するスクリプトやアプリケーションによっては、この点

を考慮する必要があります。

- アプリケーションが参照する変数の扱いについて
  LifeKeeper が実行するスクリプトに対して渡される変数は、LifeKeeper 固有のもののみです。インタプリタの設定によっては他の変数を引き継ぐことも考えられますが、基本的にアプリケーションが参照する変数は、全てスクリプト内で定義してください。

- スクリプトの引数について
  汎用アプリケーションのスクリプトに引数を指定して実行することは出来ません。

5.3 アプリケーションの制御をスクリプト単体で実現可能か確認する

汎用アプリケーションリソースは、作成したスクリプトをユーザーに代わって実行する仕組みと捉えることができます。つまり、LifeKeeper が存在しなくとも、作成したスクリプトをコマンドラインから実行し、対象アプリケーションの制御を正しく行えることが前提となります。また、LifeKeeper で保護するにあたって、異なったノード間でアプリケーションの切替を行うことが出来なければなりません。この切替のテストを行うには、実際の環境と同数のノードを用意し、事前に動作確認を行うことをお勧めします。

5.4 GUI 管理画面からの汎用アプリケーションリソースの作成

汎用アプリケーションリソースを作成するには、GUI 管理画面を起動し、[編集] > [サーバー] > [リソース階層の作成] の順にクリックし、[リソース作成ウィザード] から [汎用アプリケーション] を選択してください。具体的な画面の遷移はここでは省略します。汎用アプリケーションリソース作成のウィザードの遷移や詳細についてはオンラインマニュアルを参照してください。

汎用アプリケーションリソース作成のウィザードの中で注意しなければならない点が 1 点あります。ウィザードの中で「リソースをサービス中の状態にする] という設問に対して [はい/いいえ] の回答を行う項目が表示されます。ここで[はい] を選択した場合は、登録した restore スクリプトを実行してアプリケーションの起動を行います。restore を実行した戻り値が 0 であれば起動成功と判断し、[リソースの拡張ウィザード] に進むことが出来ます。

[いいえ] を回答したり、restore の戻り値が 0 以外であった場合は、[リソースの作成ウ

ィザード] はここで終了します。リソースの Extend はそのリソースが起動していなければならないため、改めてリソースの起動を行ってから他のノードへ Extend する作業が必要になります。そのため、特別な理由がない限り「リソースをサービス中の状態にする」という問いに対しては「はい」を選択してください。

尚、汎用アプリケーションのスクリプトは、リソースを作成した時点で、両ノードにコピーされます。事前に両ノードにスクリプトを展開しておく必要はありません。

6　スクリプトで利用可能なユーティリティ

汎用アプリケーションリソースの中で、LifeKeeper が提供するコマンドやユーティリティを実行することも可能です。スクリプトに組み込みやすい便利なコマンドや、テンプレートの情報について以下にご案内致します。

6.1　スクリプト実行ログの出力方法

スクリプト内で発生する何らかのイベントや処理に応じてログを出力させたい場合には、”lk_err”ユーティリティを使用することができます。詳細については、ドキュメントサイトの lk_err をご参照ください。

6.2　スクリプトのテンプレート

Generic ARK スクリプトのテンプレートが以下の PATH に用意されています。

C:¥LK¥Admin¥kit¥app¥templates¥

テンプレートは VB スクリプトと Perl スクリプトが用意されています。テンプレートを使用する場合は、スクリプトのコメント等を参照の上ご利用ください。また、任意のサービスを保護する簡易的な処理を記載したスクリプトも以下の PATH に用意されております。

C:¥LK¥Admin¥kit¥app¥templates¥example_service

保護対象サービスの起動状態の監視や、リソース切替時の起動/停止といった簡易的なものですが、スクリプトの開発を行うこと無くリソース化することが可能です。詳しくはユーザーサイトのこちらの記事をご参照頂ければと存じます。

6.3　リソースの登録情報とステータスの取得

登録されたリソースの情報や、現在のステータスを確認するコマンドとして、C:¥lk¥bin¥ins_list.exe が用意されています。このコマンドの出力は、一般の利用者が参照することを前提とした作りではないため、フィールドの区切り文字がメタキャラクタとなっていますが、-f<区切り文字>を付与することで、に似の文字を区切り文字として指定することができます。区切り文字を“,”(カンマ)としてタグ名 Vol.E の情報を出力した

例が以下になります。

```
C:¥LK¥Bin>ins_list -f , -t Vol.E
LK001,filesys,volume,Vol.E,E:,E: - NULL - Vol.E - SCSI - 0,ISP,restore
action has succeeded,AUTORES_ISP,,0,0,,180,300,0,INTELLIGENT
```

リソースのステータスを取得するのであれば、７番目のフィールドを切り出します。このフィールドは、リソースのステータスに応じて”ISP,OSU,OSF”といった情報が格納されます。

6.4　その他のコマンドに関する情報源

上記以外にも、LifeKeeper には様々なコマンドが用意されており、汎用アプリケーションスクリプトから呼び出して使用することも可能です。LifeKeeper のコマンドに関しては、オンラインマニュアルやドキュメントサイトからご参照ください

7　汎用アプリケーションの特殊な使い方

これまでに、LifeKeeper の汎用アプリケーションスクリプトを作成する上での必要な情報についてご紹介してきました。7 章では、LifeKeeper の特性を利用した特殊な使い方についてご紹介致します。

7.1　複数の root リソースを束ねる汎用アプリケーションリソース

root リソース（リソース階層の最上位にあるリソース）が複数設定されている場合、それぞれのリソースツリーは独立して動作することになります。この挙動は障害検知によるフェイルオーバや、手動操作でのスイッチオーバでも同様です。フェイルオーバやスイッチオーバの際には、全てのリソースツリーも含めて系切り替えさせることを想定している場合は、各リソースを束ねる汎用アプリケーションリソースを作成することで 2 つのリソースツリーを同時に切り替えることが可能です。

```
Generic Root
  +  Resource Tree 1
  |       +  Resource 1
  +  Resource Tree 2
        +  Resource 2
```

上記の最上位にある Generic Root リソースを作成し、各リソースツリーと依存関係を構築することで実現させています。こちらでご紹介している汎用リソースは、複数のリソースを束ねるためだけに設定するため、restore と remove だけを作成します。このとき、いずれも exit 0 が実行されるように設定するのがポイントです。

8 免責事項

- 本書に記載された情報は予告なしに変更、削除される場合があります。最新のものをご確認下さい。
- 本書に記載された情報は、全て慎重に作成され、記載されていますが、本書をもって、その妥当性や正確性についていかなる種類の保証もするものではありません。
- 本書に含まれた誤りに起因して、本書の利用者に生じた損害については、サイオステクノロジー株式会社は一切の責任を負うものではありません。
- 第三者による本書の記載事項の変更、削除、ホームページ及び本書等に対する不正なアクセス、その他第三者の行為により本書の利用者に応じた一切の損害について、サイオステクノロジー株式会社は一切の責任を負うものではありません。
- システム障害などの原因によりメールフォームからのお問合せが届かず、または延着する場合がありますので、あらかじめご了承ください。お問合せの不着及び延着に関し、サイオステクノロジー株式会社は一切の責任を負うものではありません。

9 著作権