機能の概要

ここでは、次の内容について説明します。

概要

ワークフローは、ビジネスロジックの表現とモデリングの間のギャップを埋めるために、標準化された方法でビジネスプロセスを自動化するのに役立ちます。

ワークフロー定義は、Serverless Workflow 仕様に基づいて作成されます。Crosswork Workflow Manager バージョン 1.0 では、仕様のサブセットのみがサポートされています。この章では、サポートされているすべての機能について説明し、各機能の実用的な例を示します。

ワークフロー定義機能

新しいワークフローは、JSON または YAML 形式で定義できます。ワークフロー定義の構造は「仕様」で説明されています。

サポートされる上位コンポーネントは次のとおりです。

- id
- name

- 説明
- version
- start
- retries
- errors
- functions
- 状態
- メタデータ

Toplevel フィールド

表 *1 : Toplevel* フィールド

パラメータ	説明
id	ワークフローの一意の識別子。
name	ワークフロー名
version	セマンティック バージョニングに基づくワークフローバージョン。
specVersion	この定義が準拠している Serverless Workflow 仕様リリースのバージョン。現在の実装は0.9仕様に従っています。
説明	ワークフローの説明テキスト。
start	最初に実行される状態。

JSON の例：

```
{
"id": "MyWorkflow",
"version": "1.0.0",
"specVersion": "0.9",
"name": "My Workflow",
"description": "My Workflow Description",
"start": "SomeState",
"states": [],
"functions": [],
"retries":[]
}
```

（注） JSON の代わりに YAML を使用する場合は、このマニュアルの例にコンバータを使用できます。

再試行定義

再試行定義は、ワークフローで実行されるアクティビティに割り当て可能なポリシーであり、ワークフローエンジンによる障害の処理方法、および障害発生時の再試行方法を制御します。

再試行定義の次のプロパティがサポートされています。

表 *2:* 再試行定義

パラメータ	定義
name	定義名。
delay	ISO 8601 形式での再試行間の時間遅延（例：30 秒の遅延の場合は「PT30S」）。
maxAttempts	最大試行回数。0：無限。再試行しない場合は、maxAttempts を 1 に設定する必要があります。
maxDelay	再試行間の最大遅延時間。ISO 8601 形式を使用します。
multiplier	各再試行の前に指定された場合、遅延値を乗算するために使用されます。浮動小数点値。たとえば、初期遅延が 30 秒で、乗数が 1.5 の場合、再試行回数は毎回 50% 増加します。

例：

```
"retries": [
     {
         "name": "Default",
         "delay": "PT1M",
         "maxAttempts": 5,
         "multiplier": 1.2
         "maxDelay": "PT3M"
     }
 ]
```

エラー定義

エラー定義では、ワークフローの実行中に発生する可能性のあるエラーを記述します。Serverless 仕様では、エラーをリストする外部ファイル（JSON や YAML）の参照がサポートされていますが、CWM はワークフロー定義で定義されたエラーのみを処理します。

エラー定義の次のプロパティがサポートされています。

表 *3:* エラー定義

パラメータ	定義
name	定義名。
code	返される可能性のあるエラーコード。現在、このフィールドはエラーマッチングには使用されていません。
説明	エラーメッセージの説明です。この説明は、アクティビティによって返されるエラーと照合するために使用されます。

（注） Serverless Workflow 仕様には、エラーメッセージを指定するオプションはないため、現在の説明がエラーとの照合に使用されます。

例：

```
"errors": [
    {
        "name": "My Custom Error",
        "code": 0,
        "description": "Specific Error Message"
    }
    ]
```

関数定義

関数定義では、実行するワークフローで使用可能な関数と、その関数が呼び出されたときにエンジンによって呼び出されるアダプタとアクティビティの名前を記述します。Serverless Workflow 仕様ではさまざまなタイプの機能がサポートされますが、CWM ではアダプタを介して公開されるアクティビティにマッピングされるカスタムタイプの関数のみサポートされます。

関数定義の次のプロパティがサポートされています。

表 *4:* 関数定義

パラメータ	定義
name	関数定義の名前。

パラメータ	定義
operation	エンジンによって呼び出されるアダプタ名とアクティビティ名を定義します。形式は、<adapter name>.<activity name> です。たとえば、NSO アダプタには **RestconfGet** というアクティビティがあります。この operation は、ワーカーに登録されているアクティビティの名前（「RestconfGet」など）になります。この名前は大文字と小文字が区別されることに注意してください。
メタデータ	Serverless Workflow 仕様のコア定義を超えた情報のモデリングを可能にします。「worker」キーは、アクティビティが実行されるタスクキューを定義するために使用されます。CWM 1.0 では、アクティビティを実行し、リッスンするタスクキューが割り当てられるワーカーの概念がサポートされています。アクティビティの実行をスケジュールするために、ワークフローエンジンはアクティビティをタスクキューに配置します。ワーカープロセスは、タスクキューから実行するタスクを取得し、アクティビティを実行します。

例：

```
"functions": [
        {
            "name": "NSO.RestconfGet",
            "operation": "restconf_Get"
            "metadata":  {
                "worker": "defaultWorker"
            }
        },
        {
            "name": "NSO.RestconfPut",
            "operation": "restconf_Put"
            "metadata":  {
                "worker": "defaultWorker"
            }
        },
        {
            "name": "NSO.RestconfPost",
            "operation": "restconf_Post"
            "metadata":  {
                "worker": "defaultWorker"
            }
        },
        {
            "name": "NSO.RestconfPatch",
            "operation": "restconf_Patch"
            "metadata":  {
```

```
            "worker": "defaultWorker"
        }
    },
    {
        "name": "NSO.RestconfDelete",
        "operation": "restconf_Delete"
        "metadata":  {
            "worker": "defaultWorker"
        }
    },
    {
        "name": "NSO.SyncFrom",
        "operation": "device_SyncFrom"
        "metadata":  {
            "worker": "defaultWorker"
        }
    },
    {
        "name": "REST.Post",
        "operation": "rest_Post"
        "metadata":  {
            "worker": "defaultWorker"
        }
    } ]
```

SubFlowRef 定義

SubFlowRef 定義は、親ワークフロー内で子ワークフローを呼び出すために使用されます。子ワークフローを使用すると、次のことができます。

- 親ワークフローのコードとワーカーを、子ワークフローのコードとワーカーから分離する。
- イベント履歴をより適切に分離するために、ワークフローによって実行されるワークロードを小さなチャンクに分割する。これは、ワークフローが多数のアクティビティ実行を生成することを目的としている場合に特に役立ちます。

subFlowRef 定義の次のプロパティがサポートされています。

パラメータ	説明
workflowId	子ワークフローの一意の ID。
version	子ワークフローのバージョン。
invoke	子ワークフローを `sync` または `async` のどちらで呼び出すかを指定します。デフォルトは `sync` です。これは、ワークフローの実行が、子ワークフローが完了するまで待機されることを意味します。
onParentComplete	invoke が `async` の場合、親ワークフローの完了時に子ワークフローの実行を終了するか続行するかを指定します。デフォルトは `terminate` です。

例

```
"states": [
    {
```

```
        "end": true,
        "name": "SpawnChildWorkflow",
        "type": "operation",
        "actions": [
            {
            "subFlowRef": {
                "version": "1.0",
                "workflowId": "subtest",
                "invoke": "sync",
                "onParentComplete": "terminate"
            }
          }
        ]
      }
    ]
```

状態

状態では、ワークフロー実行ロジックの構成要素を定義します。さまざまなタイプの状態により、実行エンジンに制御フローロジックが提供され、実行するアクティビティも定義できます。

Common 状態のプロパティ

次のプロパティはすべての状態に共通です。

表 *5 : Common* 状態のプロパティ

パラメータ	定義
name	状態名。
タイプ	サポートされるタイプは、「operation」、「switch」、「sleep」、「inject」、「foreach」です。
transition	ワークフローの次の遷移：詳細については、以下を参照してください。*Switch* 状態には適用されません。*Switch* 状態の場合、*transition* オプションは条件ごとに定義されます。
end	この状態の後にワークフローを終了する必要がある場合：詳細については、以下を参照してください。*Switch* 状態には適用されません。*Switch* 状態の場合、*end* オプションは条件ごとに定義されます。
stateDataFilter	状態のフィルタデータ入力と出力：「sleep」状態には適用されません。

パラメータ	定義
onErrors	特定の状態のエラー処理を定義します。詳細については、以下を参照してください。エラー定義に基づいて一致させることができ、補正を含む一致したエラーに基づいて transition/end を制御できます。
usedForCompensation	`true` の場合、この状態は別の状態を補うために使用されます。デフォルト：`false`。
compensatedBy	この状態の補正の原因となる状態の一意の名前。ここで識別される状態は、transition/end プロパティの「compensate」が true に設定されている場合に実行されます。

（注） いずれの状態でも、1 つの `transition` または `end` オブジェクトのみを指定できます。少なくとも 1 つ指定する必要があります。

補正

補正は、ワークフローの一部として実行された作業の取り消しを定義する方法を提供します。状態ごとに、補正状態を定義できます。実行中に、補正ロジックを実行する必要がある条件に達した場合は、transition/end を定義するときに「補正」フラグを設定できます。フラグは、実行状態、つまり **usedForCompensation** になります。詳細については、Serverless Workflow 仕様の「Workflow Compensation」を参照してください。

CWM 1.0 の実装では、補正対象としてマークされた各状態がキューに追加されます。補正状態は、後入れ先出しの観点から実行されます。

Transition

Serverless 仕様では、追加のプロパティを持つ `string` または `object` として遷移を定義できます。Crosswork Workflow Manager は、`object` 形式のみをサポートしています。現在の CWM 1.0 の実装では、「nextState」プロパティのみがサポートされています。

表 6 : Transition

パラメータ	定義
nextState	ワークフローが次に遷移する状態の名前。
compensate	true に設定すると、次の遷移が行われる前にワークフロー補正がトリガーされます。デフォルト：`false`。

終了（End）

Serverless 仕様では、end を `string` または追加のプロパティを持つ `object` として定義できます。Crosswork Workflow Manager は、`object` 形式のみをサポートしています。現在の CWM 1.0 の実装では、「nextState」プロパティのみがサポートされています。

表 *7 : End* 状態

パラメータ	定義
強制終了	この状態でワークフローを終了するかどうかを定義するブール値。
compensate	true に設定すると、実行が完了する前にワークフロー補正がトリガーされます。デフォルト：`false`。

stateDataFilter

状態データフィルタを使用すると、入力および出力データフィルタを定義できます。入力データフィルタを使用すると、必要なデータを選択できます。出力データフィルタは、次の状態に遷移する前に適用されるため、次の状態に渡されるデータをフィルタ処理できます。状態データフィルタの詳細については、こちらを参照してください。入力フィルタと出力フィルタの両方が、jq で定義されたワークフロー式です。フィルタが指定されていない場合は、すべてのデータが渡されます。

表 *8 : stateDataFilter*

パラメータ	定義
input	入力フィルタ jq 式。
output	出力フィルタ jq 式。

例：

```
"states": [
        {
            "name": "step1",
            "type": "operation",
            "stateDataFilter" : {
                "input": "${ . }"
                "output": "${ . }"
            }
            "transition": {
                "nextState": "downloadImage"
            }
        },
        {
            "name": "step2",
            "type": "operation",
            "end": {
                "terminate": "true"
```

```
            }
        }
    ]
```

onErrors

ある状態の onErrors プロパティでは、状態の実行中に発生する可能性のあるエラーとその処理方法を定義します。onErrors の詳細については、こちらを参照してください。

表 *9 : onErrors*

パラメータ	定義
errorRef または errorRefs	この状態に一致する単一の errorDef または errorDefs の配列を定義します。
transition	状態で返されたエラーが errorRef/errorRefs のエラーの説明のいずれかと一致する場合のワークフローの次の遷移。`transition` または `end` のみを定義できます。
end	状態で返されたエラーが errorRef/errorRefs のエラーの説明のいずれかに一致する場合、ワークフローは終了する必要があります。`transition` または `end` のみを定義できます。

例：

```
"onErrors": [
        {
            "errorRef": "My Custom Error",
            "end" : {
                "terminate": true
                "compensate": true
            }
        }
    ]
```

operation 状態の概要

Serverless Workflow 仕様に従って、operation 状態では、順次または並列実行される一連のアクションを定義します。Crosswork Workflow Manager は、アクションの順次実行のみをサポートしています。

1 つのアクションで、3 つの異なるタイプのサービスの起動を定義できます。

- 関数定義の実行。
- 子ワークフローとしての別のワークフロー定義の実行（現在の実装ではサポートされていません）。

- 「生成」または「消費」される可能性のあるイベントの参照（現在の実装ではサポートされていません）。

（注） 現在の実装では、関数定義の実行のみがサポートされています。

アクション

アクション定義では、この状態に対して実行する必要がある関数を指定します。次のプロパティがサポートされています。

パラメータ	説明
name	アクション名。
functionRef	実行する関数の名前を定義するオブジェクト。必要に応じて、関数が指すアクティビティに渡す引数を定義します。詳細については、以下を参照してください。
retryRef	グローバルに定義された再試行定義の名前。例：`default`。
sleep	アクションの実行前後のスリープ時間を任意で定義するオブジェクト。詳細については、以下を参照してください。
actionDataFilter	アクションに渡す必要があるデータ、アクションによって返される結果をフィルタ処理する方法、およびグローバル状態データ内のフィルタ処理結果の保存場所を制御するフィルタ。詳細については、以下を参照してください。

functionRef

パラメータ	説明
refName	関数定義を参照する関数の名前。

引数	関数に渡される引数。複雑な構造のJSONオブジェクトを指定できます。アダプタアクティビティの場合、構造はJSONである必要があります（以下を参照）。
	`{`
	`"input": {`
	`...`
	`},`
	`"resource": {`
	`...`
	`}`
	`}`

actionDataFilter

例を含む actionDataFilter の詳細については、こちらを参照してください。

パラメータ	説明
fromStateData	状態データのデータをフィルタ処理して関数に渡す jq のワークフロー式。
useResults	関数の実行から返されたデータを状態データ出力に追加またはマージするかどうかを制御するブール型フラグ。
results	関数の実行から返されたデータをフィルタ処理する jq のワークフロー式。useResults が `false` の場合は無視されます。デフォルト：`true`。
toStateData	ワークフロー式では、結果を追加またはマージする必要がある状態データを定義します。指定しない場合、結果は最上位レベルでマージされます。

sleep

パラメータ	説明
before	関数実行前のスリープ時間。例：ISO 8601 形式「PT30S」の場合、30 秒間スリープ。
after	関数実行後のスリープ時間。例：ISO 8601 形式「PT30S」の場合、30 秒間スリープ。

```
{
    "id": "example",
    "version": "1.0",
    "specVersion": "0.9",
    "start": "step1",
    "functions": [
        {
            "name": "NSO.RestconfPost",
            "operation": "RestconfPost"
        }
    ],
    "retries": [
        {
            "name": "Default",
            "maxAttempts": 5,
            "delay": "PT30S",
            "multiplier": 1.1
        }
    ],
    "states": [
        {
            "name": "step1",
            "type": "operation",
            "sleep": {
                "before": "PT1M"
            },
            "actions": [
                {
                    "retryRef": "Default",
                    "name": "showVersion",
                    "functionRef": {
                        "refName": "NSO.RestconfPost",
                        "arguments": {
                            "input": {
                                "path": "restconf/operations/devices/device=${
```

```
.deviceName }/live-status/tailf-ned-cisco-ios-stats:exec/any",
                                    "data": "{\"input\": {\"args\": \"show version\"}}"
                                }
                            }
                        },
                        "actionDataFilter": {
                            "results": "${ if (.data) then .data |
fromjson.\"tailf-ned-cisco-ios-stats:output\".result else null end }",
                            "toStateData": "${ .showVersionPreCheck }"
                        }
                    }
                ],
                "end": {
                    "terminate": "true"
                }
            }
        ]
    }
```

Switch 状態の概要

Switch 状態では、特定の条件に基づいてワークフローを特定のパスにルーティングするための決定ポイントを定義できます。Serverless Workflow 仕様では、データベースの条件とイベントベースの条件がサポートされています。CWM は、「データベースの条件」のみをサポートしています。

dataConditions

Switch 状態のデータ条件プロパティは、実行エンジンによって評価される一連の条件です。実行エンジンは、一致する最初の条件を選択し、そのパスに沿って処理を続行します。後続の条件も一致する場合、それらの条件は無視されます。

パラメータ	説明
name	条件名。
condition	条件を表す jq のワークフロー式。評価は `true/false` である必要があります。
transition	条件が一致する場合のワークフローの次の遷移。
end	条件が一致する場合、ワークフローは終了する必要があります。

（注） `transition` オブジェクトまたは `end` オブジェクトのみを指定できます。少なくとも 1 つ指定する必要があります。

defaultCondition

一致する条件がない場合に適用されるデフォルトの条件。

パラメータ	説明
transition	ワークフローの次の遷移（条件が一致しない場合）。
end	条件が一致する場合、ワークフローは終了する必要があります。

（注） transition オブジェクトまたは end オブジェクトのみを指定できます。少なくとも 1 つ指定する必要があります。

```
{
    "name": "ConditionName",
    "type": "switch",
    "dataConditions": [
        {
            "name": "IsTrue",
            "condition": "${ true  }",
            "transition": {
                "nextState": "TrueState"
            }
        },
        {
            "name": "IsFalse",
            "condition": "${ false }",
            "transition": {
                "nextState": "FalseState"
            }
        }
    ],
    "defaultCondition": {
        "end": {
            "terminate": true
        }
    }
}
```

Sleep 状態

Sleep 状態は、指定された期間、ワークフローの実行を一時停止します。

パラメータ	説明
duration	ワークフローがスリープする時間（ISO 8601 形式）。たとえば、PT1M の場合、ワークフローは 1 分間スリープ状態になります。

```
{
    "name": "Sleep3Minutes",
    "type": "sleep",
    "duration": "PT3M",
    "transition": {
        "nextState": "NextState"
    }
}
```

Inject 状態

Inject 状態は静的データを状態データに挿入するために使用されます。

パラメータ	説明
data	JSON オブジェクトが状態データに追加されました。

```
{
    "id": "example",
    "version": "1.0",
    "specVersion": "0.9",
    "start": "HelloWorld",
    "states": [
        {
            "name": "HelloWorld",
            "type": "inject",
            "data": {
                "name": "Cisco",
                "message": "Hello World"
            },
            "stateDataFilter":{
                "output": "${ .message + \" from \" + .name + \"!\"  }"
            },
            "end": {
                "terminate": "true"
            }
        }
    ]
}
```

ForEach 状態

ForEach 状態では、状態データで定義された配列またはリストの各要素に対して実行する一連のアクションを定義できます。たとえば、デバイス配列内の各デバイスについて、デバイスが同期していることを確認します。Serverless Workflow 仕様では、アクションの並列実行と順次実行のサポートが定義されていますが、現在の実装では、配列内の各要素に対するアクションの順次実行のみがサポートされています。

パラメータ	説明
inputCollection	状態データの配列を指す jq のワークフロー式。
iterationParam	各データ要素のアクションで参照できるパラメータの名前。
outputCollection	結果が追加される状態データの配列を指す jq のワークフロー式。配列が存在しない場合は作成されます。

```
{
    "id": "example",
    "version": "1.0",
    "specVersion": "0.9",
```

```
        "start": "InjectData",
        "functions": [
            {
                "name": "HelloWorld",
                "operation": "HelloWorld"
            }
        ],
        "states": [
            {
                "name": "InjectData",
                "type": "inject",
                "data": {
                    "people": [
                        {
                            "Firstname": "Peter",
                            "Surname": "Parker"
                        },
                        {
                            "Firstname": "Thor",
                            "Surname": "Odinson"
                        },
                        {
                            "Firstname": "Bruce",
                            "Surname": "Banner"
                        }
                    ]
                },
                "transition":{
                    "nextStat": "SayHelloToEveryone"
                }
            },
            {
                "name": "SayHelloToEveryone",
                "type": "foreach",
                "inputCollection": "${ .people }",
                "iterationParam": "person",
                "outputCollection": "${  .messages }",
                "actions": [
                    {
                        "name": "SayHello",
                        "functionRef":{
                            "refName": "HelloWorld",
                            "arguments": {
                                "name": "${ .person.Firstname + \" \" + .person.Surname
}"
                            }
                        }
                    }
                ],
                "end": {
                    "terminate": "true"
                }
            }
        ]
    }
```

Parallel 状態

Parallel 状態では、並行して実行されるブランチのコレクションを定義できます。ある状態の各ブランチでは、独自のアクションセットを定義できます。実行が完了すると、**completionType** 属性に基づいて並列ブランチが現在のパスに結合されます。

completeType 属性では、次の 2 つの値を定義できます。

- allOf：状態が transition/end になる前に、すべてのブランチが実行を完了する必要があります。これはデフォルト値です。
- atLeast：atLeast で指定された数のブランチの実行が完了した場合、状態は transition/end になります。`completeType` 属性が「atLeast」の場合は、numCompleted も設定する必要があります。

パラメータ	説明
completionType	ブランチの実行に基づいて状態の完了を評価する方法を定義します。「allOf」または「atLeast」。デフォルト：「allOf」
numCompleted	`completeType` が「atLeast」の場合、この値を指定する必要があります。実行を続けるために完了する必要があるブランチの最小数を定義します。

ブランチ

Parallel 状態で実行されるブランチのリスト。詳細については、こちらを参照してください。

パラメータ	説明
name	ブランチの名前。
アクション	このブランチに対して実行するアクション。1 つのブランチで一連のアクションをサポートできます。各アクションの定義は、operation 状態タイプと同じです。こちらを参照してください。

状態データ

状態データは、ワークフローのライフサイクル中に重要な役割を果たします。状態により、データのフィルタ処理、データの挿入、およびデータの追加ができます。Jq は、データのフィルタ処理、作成、および操作で重要な役割を果たします。データの処理方法の詳細については、「Serverless Workflow 仕様」を参照してください。

ワークフローを作成する場合、CMW 内のデータ管理に関しては、次のルールが適用されます。

- ワークフローの実行に渡される初期データは、入力として状態データに渡されます。
- 最後に実行された状態からのデータ出力は、ワークフロー出力です。

- 状態入力フィルタが指定されていない場合、すべてのデータが状態に渡されます。
- 状態出力フィルタが指定されていない場合、すべてのデータが次の状態に渡されます。
- jq のワークフロー式を使用すると、データをフィルタ処理および操作できます。
- アクションでも、データをフィルタ処理できます。また、アクションからの戻りデータを状態データにマージする必要がある場合にも使用できます。
- フィルタは JSON オブジェクトを返す必要があります。jq ワークフロー式の結果が文字列リテラルになると、エラーが発生します。
- jq を使用する場合は、https://jqplay.org/ を使用して jq 式をテストすることを強く推奨します。あるいは、jq をローカルにダウンロードしてテストに使用できます。

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。