IBM Tivoli Netcool/OMNIbus Message Bus Probe for Webhook Integrations Helm Chart 2.0.0

Reference Guide
February 28, 2019

SC27-9525-01

Note

Before using this information and the product it supports, read the information in Appendix A, “Notices and Trademarks,” on page 27.

Edition notice

This edition (SC27-9525-01) applies to version 2.0.0 of IBM Tivoli Netcool/OMNIbus Message Bus Probe for Webhook Integrations Helm Chart and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC27-9525-00.

Contents

About this guide

The following sections contain important information about using this guide.

Document control page

Use this information to track changes between versions of this guide.

The Message Bus Probe for Webhook Integrations Helm Chart documentation is provided in softcopy format only. To obtain the most recent version, visit the IBM® Tivoli® Knowledge Center:

https://www.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/helms/common/Helms.html

Table 1. Document modification history

Document version	Publication date	Comments
SC27-9525-00	October 11, 2018	First IBM publication.
SC27-9525-01	February 28, 2019	Guide updated for version 2.0.0 of the helm chart. Helm chart now supports ICP 3.1.x. The following topics were updated: • "Obtaining the PPA package" on page 1 • "Prerequisites" on page 2 • "Resources required" on page 2 • "Uninstalling the chart" on page 4 • "Configurable parameters" on page 5 • "Limitations" on page 25 The following topics were added: • "PodSecurityPolicy requirements" on page 2 • "Troubleshooting" on page 26

Chapter 1. Message Bus Probe for Webhook Integrations Helm Chart

The Message Bus Probe for Webhook Integrations Helm Chart allows you to deploy the IBM Netcool/OMNIbus Message Bus Probe configured with the Webhook transport onto Kubernetes to receive JSON data via HTTP notifications from a target server.

This guide contains the following sections:

The Knowledge Center contains the following additional topics that contain information that is common to all Helm Charts:

- Specifying the image repository
- Loading PPA packages to IBM Cloud Private
- Exposing the probe service
- Upgrading to a new version of the probe helm charts
- Changing the service type during a helm upgrade

Obtaining the PPA package

You can download the installation package from the **IBM Passport Advantage** website.

Use the **Find by part number** field to search for the following part number: `CC0FDEN`

Chart details

The chart deploys the Message Bus Probe configured with the Webhook transport onto Kubernetes to receive JSON data from HTTP notifications from a target server.

This chart can be deployed more than once on the same namespace.

Prerequisites

This solution requires the following applications:

- Kubernetes 1.11.1.
- Tiller 2.9.1

When connecting the probe as a HTTP client to a TLS enabled server, a pre-created Kubernetes secret which contains the remote Server certificate and a Keystore password. The expected keys in the secret are `server.crt` and `keystorepassword.txt`. Creating the secret requires Operator role or higher. The secret is then used in the `init` container. The `init` container has to be fully executed before the actual container can start. To understand the status of the pod when the `init` container is running, see Init container: Understanding Pod status.

Note : Operator role is a minimum requirement to install this chart.

The chart must be installed by a Administrator to perform the following task:

- Enable Pod Disruption Budget policy.
- Retrieve sensitive information from a secret such as TLS certificate.

The chart must be installed by a Cluster Administrator to perform the following task in addition to the one above:

- Obtain the Node IP using `kubectl get nodes` command if using the NodePort service type.
- Create a new namespace with custom PodSecurityPolicy if necessary. For details see "PodSecurityPolicy requirements" on page 2.

To connect the probe to a Kafka server using a secure connection, the Kubernetes Secrets for the client Truststore certificate and the client Keystore certificate must be created before deploying the helm chart. Creating the Secrets requires the Operator role or higher.

Resources required

This solution requires the following resources:

- CPU Requested : 100m (100 millicpu)
- Memory Requested : 128Mi (~ 134 MB)

PodSecurityPolicy requirements

This chart requires a PodSecurityPolicy to be bound to the target namespace prior to installation. You can choose either a predefined PodSecurityPolicy or have your cluster administrator create a custom PodSecurityPolicy for you.

The predefined PodSecurityPolicy name `ibm-restricted-psp` has been verified for this chart, see IBM Cloud Pak Pod Security Policy Definitions. If your target namespace is bound to this PodSecurityPolicy, you can proceed to install the chart. The predefined PodSecurityPolicy definitions can be viewed here: https://github.com/IBM/cloud-pak/blob/master/spec/security/psp/README.md

This chart also defines a custom PodSecurityPolicy which can be used to finely control the permissions/capabilities needed to deploy this chart. You can enable this custom PodSecurityPolicy using the ICP user interface or the supplied instructions/scripts in the `pak_extension pre-install` directory. For detailed steps on creating the PodSecurityPolicy see https://www.ibm.com/support/knowledgecenter/SSSHTQ/omnibus/helms/all_helms/wip/reference/hlm_common_psp.html

From the user interface, you can copy and paste the following snippets to enable the custom Pod Security Policy

- From the user interface, you can copy and paste the following snippets to enable the custom PodSecurityPolicy:

– Custom PodSecurityPolicy definition:

```
    apiVersion: extensions/v1beta1
    kind: PodSecurityPolicy
    metadata:
    annotations:
        kubernetes.io/description: "This policy is based on the most restrictive policy,
        requiring pods to run with a non-root UID, and preventing pods from accessing the
host."
        seccomp.security.alpha.kubernetes.io/allowedProfileNames: docker/default
        seccomp.security.alpha.kubernetes.io/defaultProfileName: docker/default
    name: ibm-netcool-probe-messagebus-webhook-prod-psp
    spec:
    allowPrivilegeEscalation: false
    forbiddenSysctls:
    - '*'
    fsGroup:
        ranges:
        - max: 65535
        min: 1
        rule: MustRunAs
    hostNetwork: false
    hostPID: false
    hostIPC: false
    requiredDropCapabilities:
    - ALL
    runAsUser:
        rule: MustRunAsNonRoot
    seLinux:
        rule: RunAsAny
    supplementalGroups:
        ranges:
        - max: 65535
        min: 1
        rule: MustRunAs
    volumes:
    - configMap
    - emptyDir
    - projected
    - secret
    - downwardAPI
    - persistentVolumeClaim
```

– Custom ClusterRole for the custom PodSecurityPolicy:

```
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: ibm-netcool-probe-messagebus-webhook-prod-clusterrole
    rules:
    - apiGroups:
      - extensions
      resourceNames:
      - ibm-netcool-probe-messagebus-webhook-prod-psp
      resources:
      - podsecuritypolicies
      verbs:
      - use
```

– RoleBinding for all service accounts in the current namespace. Replace {{ NAMESPACE }} in the template with the actual namespace:

```
    apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    metadata:
      name: ibm-netcool-probe-messagebus-webhook-prod-rolebinding
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: ClusterRole
      name: ibm-netcool-probe-messagebus-webhook-prod-clusterrole
    subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: Group
      name: system:serviceaccounts:{{ NAMESPACE }}
```

- From the command line, you can run the setup scripts included under `pak_extensions`.

  As a cluster administrator, the pre-install scripts and instructions are in the following location:

`pre-install/clusterAdministration/createSecurityClusterPrereqs.sh`

As team admin/operator the namespace scoped scripts and instructions are in the following location:

`pre-install/namespaceAdministration/createSecurityNamespacePrereqs.sh`

Installing a helm release

To install the chart, use the following steps:

1. Extract the helm chart archive and customize `values.yaml`. The configuration section lists the parameters that can be configured during installation.
2. Install the chart with the release name `my-mb-webhook-probe` using the configuration specified in the customized `values.yaml` using following command:

   ```
   helm install --tls --namespace <your pre-created namespace> --name my-mb-webhook-probe -f values.yaml stable/ibm-netcool-probe-messagebus-webhook-prod
   ```

   Where: *my-mb-webhook-probe* is the release name for the chart.

   Helm searches for the `ibm-netcool-probe` chart in the helm repository called `stable`. This assumes that the chart exists in the stable repository.

   Tip : You can list all releases using `helm list --tls` or you can search for a chart using **helm search**.

   The command deploys on the Kubernetes cluster using a default configuration. For a list of the parameters that you can configure during installation see “Configurable parameters” on page 5.

Escaping quotes when performing a UI installation

To install the Webhook chart using UI, on IBM Cloud Private (ICP) version 2.1.0.2, you need to quote the whole JSON context with double quotes. Escaping double quotes using \ is not supported. The following example shows how to pass the whole JSON context as a string by quoting.

```
"{"grant_type":"password", "username":"++Username++", "password":"++Password++"}"
```

ICP version supports escaping using double quotes. The following example shows how to escape the quotes in JSON context.

```
{\"grant_type\":\"password\", \"username\":\"++Username++\", \"password\":\"++Password++\"}
```

Verifying the chart

See the instructions at the end of the helm installation for chart verification. The instructions can also be displayed by viewing the installed helm release under **Menu -> Workloads -> Helm Releases** or by running the following command:

```
helm status <release> --tls
```

Uninstalling the chart

To uninstall or delete the `my-mb-webhook-probe` deployment:, use the following command:

```
$ helm delete my-mb-webhook-probe --purge --tls
```

The command removes all the Kubernetes components associated with the chart and deletes the release.

Clean up any prerequisites that were created

As a Cluster Administrator, run the cluster administration cleanup script included under `pak_extensions` to clean up cluster scoped resources when appropriate.

`post-delete/clusterAdministration/deleteSecurityClusterPrereqs.sh`

As a Cluster Administrator, run the namespace administration cleanup script included under `pak_extensions` to clean up namespace scoped resources when appropriate.

`post-delete/namespaceAdministration/deleteSecurityNamespacePrereqs.sh`

Configuring the chart

The integration requires configuration of the chart parameters.

Configurable parameters

You use parameters to specify how the probe interacts with the device. You can override the chart's default parameter settings during installation.

There are two components that can be configured for the probe:

- HTTP Server (Webhook): to receive HTTP notifications (main operation mode)
- HTTP Client: to pull or re-synchronize events from a target server (optional in addition to HTTP Sever)

The following tables describe the configurable parameters for this chart and lists their default values. You can specify each parameter using the `--set key=value[,key=value]` argument with `helm install`.

HTTP server configuration

The HTTP server configuration describes how the probe receive HTTP notifications.

Table 2. Configurable parameters for the HTTP server

Parameter name	Description
`license`	Use this parameter to specify the license state of the image being deployed. Enter `accept` to install and use the image. The default value is `not accepted`
`replicaCount`	Use this parameter to specify the number of deployment replicas. The default value is `1`
`global.image.secretName`	Use this parameter to specify the name of the Secret containing the Docker Config to pull image from a private repository. Leave blank if the probe image already exists in the local image repository or the Service Account has been assigned with an Image Pull Secret. The default value is `nil`

Table 2. Configurable parameters for the HTTP server (continued)	
Parameter name	**Description**
image.repository	Use this parameter to specify the probe image repository. Update this repository name to pull from a private image repository. The image name should be set to `netcool-probe-messagebus` The default value is `netcool-probe-messagebus`
image.tag	Use this parameter to specify the `netcool-probe-messagebus` image tag. The default value is `9.0.9`
image.testRepository	Use this parameter to specify the utility image (busybox) repository. Update this repository name to pull from a private image repository. The default value is `busybox`
image.testImagetag	Use this parameter to specify the utility image tag. The default value is `1.28.4`
image.pullPolicy	Use this parameter to specify the image pull policy. The default value is `IfNotPresent`
image.tag	Use this parameter to specify the `netcool-probe-messagebus` image tag. The default value is `9.0.8`
netcool.primaryServer	Use this parameter to specify the primary Netcool/OMNIbus server the probe should connect to (required). The default value is `nil`
netcool.primaryHost	Use this parameter to specify the host of the primary Netcool/OMNIbus server (required). The default value is `nil`
netcool.primaryPort	Use this parameter to specify the port number of the primary Netcool/OMNIbus server (required). The default value is `nil`

Table 2. Configurable parameters for the HTTP server (continued)	
Parameter name	**Description**
netcool.backupServer	Use this parameter to specify the backup Netcool/OMNIbus server to connect to. If the **backupServer**, **backupHost** and **backupPort** parameters are defined in addition to the **primaryServer**, **primaryHost**, and **primaryPort** parameters, the probe will be configured to connect to a virtual object server pair called `AGG_V`. The default value is `nil`
netcool.backupHost	Use this parameter to specify the host of the backup Netcool/OMNIbus server. The default value is `nil`
netcool.backupPort	Use this parameter to specify the port of the backup Netcool/OMNIbus server. The default value is `nil`
probe.messageLevel	Use this parameter to specify the probe log message level. The default value is `warn`
probe.heartbeatInterval	Use this parameter to specify the probe heartbeat interval (in seconds) to check the transport connection status. The default value is `1`
probe.rulesConfigmap	Use this parameter to specify an alternative ConfigMap. If set,it overrides the template rules files with this ConfigMap containing a custom rules files in `message_bus.rules` key. Leave empty to use the default rules file template which can be customized later. The default value is `nil`
probe.jsonParserConfig.notification.messagePayload	Use this parameter to specify the JSON tree to be identified as message payload from the notification (webhook) channel. See example for more details on how to configure the Probe's JSON parser. The default value is `json`
probe.jsonParserConfig.notification.messageHeader	Use this parameter to specify the JSON tree to be identified as message header from the notification (webhook) channel. Attributes from the headers will be added to the generated event. The default value is `nil`

Table 2. Configurable parameters for the HTTP server (continued)	
Parameter name	**Description**
probe.jsonParserConfig.notification.jsonNestedPayload	Use this parameter to specify the JSON tree within a nested JSON or JSON string to be identified as message payload from the notification (webhook) channel. The **probe.jsonParserConfig.notification.messagePayload** must be set to point to the attribute containing the JSON String. The default value is `nil`
probe.jsonParserConfig.notification.jsonNestedHeader	Use this parameter to specify the JSON tree within a nested JSON or JSON string to be identified as message header from the notification (Webhook) channel. The **probe.jsonParserConfig.notification.messageHeader** must be set to point to the attribute containing the JSON String. The default value is `nil`
probe.jsonParserConfig.notification.messageDepth	Use this parameter to specify the number of levels in the message to traverse during parsing. The default value is 3
service.probe.type	Use this parameter to specify the probe k8 service type exposing ports, for example `ClusterIP` or `NodePort`. The default value is `ClusterIP`
service.probe.externalPort	Use this parameter to specify the external port for this service. The default value is `80`
ingress.enabled	Use this parameter to specify whether the Ingress is enabled. The default value is FALSE
ingress.hosts	Use this parameter to specify the Host to route requests based on. The Helm Release Name will be appended as a prefix. Required when using Ingress. Ignored when ingress is disabled and the default value can be used as dummy value to proceed with installation. The default value is `netcool-probe-messagebus-webhook.local`
ingress.annotations	Use this parameter to specify the meta data to drive Ingress class used. The default value is `nil`

Table 2. Configurable parameters for the HTTP server (continued)	
Parameter name	**Description**
ingress.tls.enabled	Use this parameter to enable or disable TLS to secure channel from external clients / hosts. The default value is FALSE
ingress.tls.secretName	Use this parameter to specify the TLS secret to secure channel from external clients / hosts. The secret must contain the `tls.crt` and `tls.key` entries. If `ingress.tls.enabled=true` and this parameter is unset, a TLS secret will be created. The default value is `nil`
ingress.tls.caName	Use this parameter to specify a Certificate Authority name used to create the CA certificate when signing the TLS certificate. Used when **ingress.tls.secretName** is unset. The default value is `IBM Netcool/OMNIbus Integration`
webhook.httpVersion	Use this parameter to specify the version of the HTTP protocol to use. The probe supports 1.1 or 1.0. The default value is `1.1`
webhook.uri	Use this parameter to specify the probe's Webhook URI into which the target device will POST notifications. The default value is `/probe`
webhook.respondWithContent	Use this parameter to specify whether the probe includes the HTTP body received from the client HTTP request in the HTTP response. Set to ON to specify that the probe includes the HTTP body. The default value is OFF
webhook.validateBodySyntax	Use this parameter to specify the probe performs a JSON format check. Set to ON to perform a JSON format check on the HTTP request body. The default value is ON
webhook.validateRequestURI	Use this parameter to enable or disable a URI path check. Set this property to ON to enable URI path check. Set this property to OFF to disable the URI check and the Webhook will accept all HTTP request regardless of the path set. The default value is ON

Table 2. Configurable parameters for the HTTP server (continued)	
Parameter name	**Description**
webhook.idleTimeout	Use this parameter to specify the time (in seconds) to allow an idle HTTP client to be connected. The default value is `180`
webhook.keepTokens	Use this parameter to specify a comma-separated list of the attributes that the probe extracts from the incoming JSON data. These data items can be used in token substitution The default value is `nil`
autoscaling.enabled	Use this parameter to enable or disable auto-scaling. The default value is TRUE
autoscaling.minReplicas	Use this parameter to specify the minimum number of probe replicas. The default value is 2
autoscaling.maxReplicas	Use this parameter to specify the maximum number of probe replicas. The default value is 5
autoscaling.cpuUtil	Use this parameter to specify the target CPU utilization. For example, enter `60` for `60%` target utilization. The default value is `60`
poddisruptionbudget.enabled	Use this parameter to specify the enable or disable Pod Disruption Budget to maintain high availability during a node maintenance. The default value is FALSE
poddisruptionbudget.minAvailable	Use this parameter to specify the minimum number of available pods during node drain. Can be set to a number or percentage, eg: 1 or 10%. Caution: Setting to 100% or equal to the number of replicas may block node drains entirely. The default value is `1`
resources.limits.memory	Use this parameter to specify the memory resource limits. The default value is `512Mi`
resources.limits.cpu	Use this parameter to specify the CPU resource limits. The default value is `500m`

Table 2. Configurable parameters for the HTTP server (continued)	
Parameter name	**Description**
resources.requests.cpu	Use this parameter to specify the CPU resource requests. The default value is `100m`
resources.requests.memory	Use this parameter to specify the memory resource requests. The default value is `128Mi`
arch	Use this parameter to specify the worker node architecture. This is fixed to `amd64`.

HTTP client configuration

The HTTP client configuration describes how the probe pulls or re-synchronizes events from a target server.

Table 3. Configurable parameters for the HTTP client	
Parameter name	**Description**
probe.host	Use this parameter to specify the target server IP or hostname. The default value is `nil`
probe.port	Use this parameter to specify the target server port. The default value is `80`
probe.username	Use this parameter to specify the username to use when authenticating with the target server. The default value is `nil`
probe.password	Use this parameter to specify the password to use when authenticating with the target server. The default value is `nil`
probe.initialResync	Use this parameter to specify whether the probe sends a re-synchronization request before sending a subscription request. Set to true to send a re-synchronization request. The default value is `false`
probe.resyncInterval	Use this parameter to specify the interval (in seconds) to check the transport connection status. The default value is `0`

Table 3. Configurable parameters for the HTTP client (continued)	
Parameter name	**Description**
probe.sslSecretName	Use this parameter to specify a pre-created Kubernetes secret containing `server.crt` and `keystorepassword.txt` keys. It is required when connecting to a HTTP server with TLS. The default value is `nil`
probe.jsonParserConfig.resync. messagePayload	Use this parameter to specify the JSON tree to be identified as message payload from re-synchronization (HTTP REST API). See example for more details on how to configure the Probe's JSON parser. The default value is `json`
probe.jsonParserConfig.resync. messageHeader	Use this parameter to specify the JSON tree to be identified as message header from re-synchronization (HTTP REST API). Attributes from the headers will be added to the generated event. The default value is `nil`
probe.jsonParserConfig.resync. jsonNestedPayload	Use this parameter to specify the JSON tree within a nested JSON or JSON string to be identified as message payload from re-synchronization (HTTP REST API). The **probe.jsonParserConfig.resync. messagePayload** must be set to point to the attribute containing the JSON String. The default value is `nil`
probe.jsonParserConfig.resync. jsonNestedHeader	Use this parameter to specify the JSON tree within a nested JSON or JSON string to be identified as message header from re-synchronization (HTTP REST API). The **probe.jsonParserConfig.resync. messageHeader** must be set to point to the attribute containing the JSON String. The default value is `nil`
probe.jsonParserConfig.resync. messageDepth	Use this parameter to specify the number of levels in the message to traverse during parsing. The default value is 3

Table 3. Configurable parameters for the HTTP client (continued)	
Parameter name	**Description**
webhook.httpHeaders	Use this parameter to specify the a comma-separated list of HTTP header options to use in all HTTP requests. Accepts key-value pairs using the equals sign (=) as the value separator. For example, Accept=application/json,Content-Type=application/json,Use-Cookie=true,User-Agent=IBM Netcool/OMNIbus Message Bus Probe. The default value is `nil`
webhook.responseTimeout	Use this parameter to specify the time (in seconds) the probe waits for a response from the target system before timing out. The default value is `60`
webhook.autoReconnect	Use this parameter to specify whether the probe re-establishes the connection to the remote HTTP server when disconnected. Set to `ON` to re-establish the connection to the remote HTTP server when disconnected. The default value is `ON`
webhook.keepTokens	Use this parameter to specify a comma-separated list of the attributes that the probe extracts from the incoming JSON data. These data items can be used in token substitution in subsequent HTTP requests header or body. The default value is `nil`
webhook.securityProtocol	Use this parameter to specify the security protocol to use when retrieving events from the REST API with TLS. Example: TLSv1.2, TLSv1.1 or TLSv1.0. The probe.sslSecretName is required. The default value is `nil`
webhook.loginRequest.uri	Use this parameter to specify the URI that the probe uses to request a login. For example, can be used to obtain an access token. The default value is `nil`
webhook.loginRequest.method	Use this parameter to specify the HTTP method for the login request. The default value is `nil`

Table 3. Configurable parameters for the HTTP client (continued)	
Parameter name	**Description**
webhook.loginRequest.headers	Use this parameter to specify a comma-separated list of HTTP header options for login request. This overrides any header set in **webhook.httpHeaders**. The default value is `nil`
webhook.loginRequest.content	Use this parameter to specify the HTTP body to send in the login request. The default value is `nil`
webhook.resyncRequest.uri	Use this parameter to specify the URI that the probe uses to pull (resync) data. The default value is `nil`
webhook.resyncRequest.method	Use this parameter to specify the HTTP method for the resync request. The default value is `nil`
webhook.resyncRequest.headers	Use this parameter to specify a comma-separated list of HTTP header options for resync request. This overrides any header set in webhook.httpHeaders The default value is `nil`
webhook.resyncRequest.content	Use this parameter to specify the HTTP body to send in the resync request. The default value is `nil`
webhook.subscribeRequest.uri	Use this parameter to specify the URI that the probe uses to create a subscription for notification. The default value is `nil`
webhook.subscribeRequest.method	Use this parameter to specify the HTTP method for the subscription request. The default value is `nil`
webhook.subscribeRequest.headers	Use this parameter to specify a comma-separated list of HTTP header options for subscription request. This overrides any header set in **webhook.httpHeaders**. The default value is `nil`
webhook.subscribeRequest.content	Use this parameter to specify the HTTP body to send in the subscription request. The default value is `nil`

Table 3. Configurable parameters for the HTTP client (continued)	
Parameter name	**Description**
webhook.loginRefresh.uri	Use this parameter to specify the URI that the probe uses to refresh a login. For example, this can be used to refresh an access token. The default value is `nil`
webhook.loginRefresh.method	Use this parameter to specify the HTTP method for the login refresh request. The default value is `nil`
webhook.loginRefresh.headers	Use this parameter to specify a comma-separated list of HTTP header options for login refresh request. This overrides any header set in **webhook.httpHeaders**. The default value is `nil`
webhook.loginRefresh.content	Use this parameter to specify the HTTP body to send in the login refresh request. The default value is `nil`
webhook.loginRefresh.interval	Use this parameter to specify the interval (in seconds) between successive refresh requests. The default value is `nil`
webhook.subscribeRefresh.uri	Use this parameter to specify the URI that the probe uses to refresh the subscription. The default value is `nil`
webhook.subscribeRefresh.method	Use this parameter to specify the HTTP method for the subscription refresh request. The default value is `nil`
webhook.subscribeRefresh.headers	Use this parameter to specify a comma-separated list of HTTP header options for subscription refresh request. This overrides any header set in **webhook.httpHeaders**. The default value is `nil`
webhook.subscribeRefresh.content	Use this parameter to specify the HTTP body to send in the login refresh request. The default value is `nil`
webhook.subscribeRefresh.interval	Use this parameter to specify the interval (in seconds) between successive refresh requests. The default value is `nil`

Table 3. Configurable parameters for the HTTP client (continued)	
Parameter name	**Description**
webhook.refreshRetryCount	Use this parameter to specify the number of attempts to re-send the refresh requests before disconnecting from the server. This is used when the probe receives a non-0K refresh response. The default value is 0
webhook.logoutRequest.uri	Use this parameter to specify the URI that the probe uses to request a logout. For example, can be used to delete a subscription on the target server. The default value is nil
webhook.logoutRequest.method	Use this parameter to specify the HTTP method for the logout request. The default value is nil
webhook.logoutRequest.headers	Use this parameter to specify a comma-separated list of HTTP header options for logout request. This overrides any header set in **webhook.httpHeaders**. The default value is nil
webhook.logoutRequest.content	Use this parameter to specify the HTTP body to send in the logout request. The default value is nil

Configuring the probe's HTTP client component

The IBM Tivoli Netcool/OMNIbus Message Bus Probe for Webhook Integrations can be configured to integrate with event sources which support REST API while also acting as a HTTP server to listen to HTTP notifications via the webhook URL. The probe can use REST API to pull data (or re-synchronize events) by sending HTTP requests.

Configuring the probe to perform an initial resync

To configure the probe to send HTTP requests, use the following steps :

1. Set the **probe.host** and **probe.port** with the target server IP or hostname and the port number. For HTTP, it is usually port 80 and 443 for TLS enabled servers.
2. Set the **probe.initialResync** parameter to true to configure the probe to perform a re-synchronization during startup. The probe will create a HTTP request using the parameters configured by **webhook.resyncRequest.uri** , **webhook.resyncRequest.method**, **webhook.resyncRequest.headers** and **webhook.resyncRequest.content**.
3. Optionally, set the **probe.resyncInterval** parameter to a number greater than 0 (seconds) to set the re-synchronization interval.
4. Configure the **webhook.resyncRequest.*** parameters as follows:

Table 4. Configurable parameters for the HTTP server	
Parameter name	**Value**
webhook.resyncRequest.uri	The URI that the probe uses to pull (resync) data. For example: `/api/alarms/active`
webhook.resyncRequest.method	The HTTP method for the resync request. Set to POST or any of the supported REST API methods supported by the target server.
webhook.resyncRequest.headers	(Optional) A comma-separated list of HTTP header options for resync request. This overrides any header set in **webhook.httpHeaders**. For example, to use `Bearer` token authentication is this request, set this parameter to: `Authorization=Bearer <your token>`
webhook.resyncRequest.content	(Optional) The HTTP body to send in the resync request. For example, to send a JSON body, set this parameter to: `{\"severities\":[\"minor\",\"major\",\"critical\",\"clear\"]}`

Note : When using the ICP UI, if any of the parameter values use quote character see "Escaping quotes when performing a UI installation" on page 4

5. For SSL enabled servers, the probe needs the server certificate in order to send a HTTPS request. This server certificate must be put into a Secret along with a Keystore password. See "Enabling a secured connection with a TLS enabled event source" on page 17.
6. (Optional) You can configure other request parameters in addition to the webhook.resyncRequest.* parameters to perform other REST API calls to the target server. For more details about the configurable parameters available, see "Configurable parameters" on page 5.

Configuring the probe with a custom rules file

The rules files for the Message Bus probe can be customized and loaded with the helm chart. For details about creating a config map from a custom rules file, see "Specifying a custom probe rules file" on page 21.

Enabling a secured connection with a TLS enabled event source

This is only applicable when configuring the probe to connect as a client to a TLS enabled target server.

Pre-requisite

To connect to a TLS enabled target server, the probe requires a secret containing the TLS configuration to establish a secured connection. An Administrator needs to pre-create a secret that contains the target server's certificate and keystore password before installing the chart. The created secret must contain the

following keys `server.crt` and `keystorepassword.txt` where the former key contains the server certificate and the latter key contains the keystore password.

- Server certificate is the target server's TLS certificate usually is a X.509 PEM file. Make sure it is save in .crt extension. An example command to obtain the server certificate using Open SSL tool is shown below:

```
export TARGET_HOST=target-server.host;
 export TARGET_PORT=443;
 openssl s_client \
 -connect $TARGET_HOST:$TARGET_PORT \
 -showcerts 2>/dev/null \
 | openssl x509 -outform PEM > server.crt
```

- Keystore password is a text file that contains the password to be used as the probe's keystore password. The password will be encrypted using `nco_aes_crypt` tool during chart installation. A simple keystore password can be created by running:

```
echo -n "password123" > keystorepassword.txt
```

Creating a secret for the probe

To create a secret for the probe, use the following steps:

1. The administrator must import the server certificate and name it `server.crt` and then create a password using the name `keystorepassword.txt`.

   Note : The password will be encrypted during the installation process.

2. To create a secret, use the command `kubectl create secret tls <Secret Name> --from-file=server.crt --from-file=keystorepassword.txt`. Substitute `<Secret Name>` in the command with the actual name, for example `probe-client-secret`.

   Note : The expected secret keys are `server.crt` and `keystorepassword.txt`, hence the files must have the correct names.

Configuring the probe to use the pre-created secret

To configure the probe to use the pre-created secret, use the following steps:

1. When installing the chart, set the **`probe.sslSecretName`** parameter with value of the secret created in **Creating a Secret For The Probe** section (for example, `probe-client-secret` as used by the sample command above).
2. Set the **`webhook.securityProtocol`** parameter to the server supported security protocol version, for example `TLSv1.2`, `TLSv1.1` or `TLSv1`.

Troubleshooting a failed installation when Secured Mode is enabled

If the status of the pods remains in the pending state after the installation, this may indicate that installation might have failed and the probe is unable to start.

In this case,it is recommended to use the command line to troubleshoot the condition of the pods.

1. To identify the pod names that belongs to the unhealthy release, use command `kubectl get pods -l release==<release name>`, where `<release name>` is the Helm release name. Alternatively using the UI, go to **Menu > Workloads > Deployments** and find the probe deployment which has the **Ready** and **Available** pod count less than the **Desired** and **Current** counts.
2. If the pods are terminating or pending, run the following command:

   ```
   kubectl describe pod <pod name>
   ```

3. Identify possible warning from the generated output. (Troubleshoot based on the output)
   - If the generated warning message looks similar to the example provided below, the pod cannot start due to the provided secret name does not exist in kubernetes environment.

```
Warning FailedMount 45m (x8 over 46m) kubelet, MountVolume.SetUp failed for
volume "tls-secret" : secrets "wrongsecret" not found
```

- If the generated warning message looks similar to the example provided below, the pod cannot start due to the provided secret does not contain server.crt and/or keystorepassword.txt as key.

```
Warning FailedMount 21s (x7 over 52s) kubelet, MountVolume.SetUp failed for
volume "tls-secret" : references non-existent secret key
```

4. For more information about the failed installation, check the init container's log by running the following command:

```
kubectl logs <pod name> -c init-webhook
```

5. To delete the unhealthy release, use the following command:

```
helm del --purge <release name>
```

If the pods are not successfully deleted, use the following command to forcefully delete them:

```
kubectl delete pod <pod name> --force --grace-period=0
```

Enabling Ingress

This topic describes how to configure Ingress. For events to reach the probe from outside of the cluster, you must enable Ingress with the ClusterIP service. Ingress acts as a front-end that can be configured to provide services with externally reachable URLs for the purpose of routing traffic to the backend service.

How to enable Ingress

The following configurable parameters are required to enable Ingress.

Ingress section:

- **`Ingress Enabled`**: To enable Ingress.
- **`Ingress host name`**: To specify the domain name for Ingress host. <.> will be prepended upon installation
- **`Annotations`**: Ingress annotations.

Web Hook transport configuration section:

- **`Webhook URI`**: The Web Hook URI will be used as the Ingress path.

To enable Ingress, check the **Ingress Enabled** check box and install the chart using the following values for each of the configurable parameters.

Table 5. Configurable parameters for Ingress

Parameter name	**Value**
`Release Name`	`sample`
`Ingress Enabled`	`checked`
`Ingress Host Name`	**`netcool-probe-messagebus-webhook.local`**
`Annotations`	**`nil`**
`Webhook URI`	**`/probe`**

Note : The highlighted values are the default values for each of the parameters.

The Ingress host name will be prepended with < ReleaseName > and <.>, thus forming sample.netcool-probe-messagebus-webhook.local with the Ingress path set as /probe.

Before sending events, configure the source server to resolve the Ingress host by adding the hostname and proxy IP to the server's proxy or /etc/hosts.

Securing Ingress with TLS

This is only applicable when configuring the probe to start as a server module with webhook.

You can secure Ingress by specifying a secret that contains a TLS private key and certificate. The created secret must contain keys named tls.crt and tls.key which contains the server certificate and the private key.

You can obtain the secret for the chart in the following ways:

- "Using a secret generated by the chart" on page 20
- "Providing a pre-created secret" on page 20

Using a secret generated by the chart

On Message Bus Webhook chart, check the **TLS enabled** or set ingress.tls.enable to true using CLI to enable the TLS. If the **TLS Secret Name** is left empty, by default the chart will generate a signed certificate that is signed by IBM Netcool/OMNIbus Integration as the Certificate Authority (CA). To change the CA Name, edit the value for parameter **Certificate Authority (CA) Name**.

To send events to the probe when TLS is enabled, you must extract the tls-ca.crt from the generated secret.

1. Run the following command to obtain the secret name:

   ```
   kubectl get secret -l release==<ReleaseName>
   ```

2. Run the following command to get the CA cert:

   ```
   kubectl get secrets/{{ $secretName }} --output=jsonpath={.data.tls-ca\\.crt}
   | base64 --decode > ca_crt.pem
   ```

To view the complete guide in the installation notes from the UI go to **Workloads > Helm Releases > Release Name > Notes**.

Providing a pre-created secret

Cluster Admin can pre-create a signed certificate using a trusted CA and then allow users to install the Webhook chart using the secret.

Note : Currently, to enable the TLS, the secret must contain keys named tls.crt and tls.key.

For the certificate to validate the correct hostname, the server certificate must include the correct hostname as the certificate's commonName (CN). Cluster Admin must inform the users so that they know that they are required to install the chart with a specified **Release Name**.

Note : When Ingress is enabled, "ReleaseName" and "." will be appended as a prefix for the Ingress host. This ensures the chart is able to deploy more than once with Ingress and TLS enabled using the default values.yaml.

For example, if a user installs the helm chart using the release name my-probe, and the default ingress host, the certificate must contain a CN that looks like this:

my-probe.netcool-probe-messagebus-webhook.local

Where my-probe. is the appended part.

Using a self-signed certificate

Cluster Admin can also opt to pre-create a self-signed certificate, then allow users to install the Webhook chart using the secret. **Note:** Currently to enable the TLS, the secret must contain keys named `tls.crt` and `tls.key`. Self-signed certificates can be created using tools like OpenSSL and Java's `keytool`.

To create a simple self-signed certificate using OpenSSL, run the following command:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
```

Testing the TLS enabled Ingress

A sample `curl` command to test the Wbhook is shown below and the response code should be `200` which indicates that the HTTP request is successfully received by the probe:

```
$ export PROXY_NODE=$(kubectl get ingress -l release==my-probe  -o jsonpath='{.items[0].status.loadBalancer.ingress[].ip}')

$ curl -H "Content-Type: application/json" \
  --resolve my-probe.netcool-probe-messagebus-webhook.local:443:$PROXY_NODE \
  -X POST https://my-probe.netcool-probe-messagebus-webhook.local/probe \
  -d '{"key123" :"value123"}' --cacert ca_crt.pem \
  --write-out '%{http_code}'
```

Where:

- `my-probe` is the release name.
- `my-probe.netcool-probe-messagebus-webhook.local` is the ingress hostname set in ingress.host parameter.
- `ca_crt.pem` is the CA certificate retrieved from the TLS secret.

For details about securing Ingress with TLS, see https://kubernetes.io/docs/concepts/services-networking/ingress/#tls.

Specifying a custom probe rules file

The Message Bus Probe for Kafka Integration helm chart and the Message Bus Probe for Webhook Integration helm chart allow you to specify a custom probe rules file by overriding the default rules file config map.

By specifying an overriding config map, the custom probe rules config map is not managed by the helm release and will not be removed when the release is deleted. It can also be re-used by more than one Helm releases in the same namespace.

Creating An Overriding Rules Config Map

To create a new config map to override the default rules files config map, use the following steps:

1. Create a new custom rules file or re-use an existing rules file.
2. Name the rules file `message_bus.rules` or create a new copy of the existing rules file using the copy command as shown below.

   ```
   cp my_custom.rules message_bus.rules
   ```

3. Create a new config map from the `message_bus.rules` file because the expected config map key is `message_bus.rules`. This config map must be in the same namespace as the probe deployment. Optionally, specify the target namespace using the `--namespace <namespace>` option in the following command:

   ```
   kubectl create configmap mb-custom-rules \
   --from-file=message_bus.rules \
   --namespace default
   ```

4. Configure a new probe release and set the **`rulesConfigmap`** parameter to the config map name created in the step above (`mb-custom-rules`).

Note : The full parameter path may differ between charts. For example, the Webhook Helm Chart uses `probe.rulesConfigmap` and the Kafka Helm Chart uses `messagebus.probe.rulesConfigmap`.

5. For UI installation, the **`rulesConfigmap`** parameter is labeled `Custom Rules File ConfigMap Name.`

Configuring the JSON parser

The Message Bus Probe JSON parser can be configured to process different JSON event structures according to the endpoint set by the probe's transport module. This is useful when the event source system sends different JSON data structure depending on the API. For example, some event sources supports alarm query for clients to pull or re-synchronize alarms will return a batch or array of alarms while notification events are usually sent individually.

The JSON parser has several configurable property in order to traverse the JSON event and extract data to create an event. The following table shows the configurable parameters of the parser:

Table 6. Configurable parser parameters

Parameter name	Description
`endpoint`	The endpoint name. This value must match the endpoint set by the probe's transport module and may vary between transport types. Some of the probe helm charts may not expose this parameter,and it is preconfigured in the helm chart template.
`messagePayload`	Specifies the JSON tree to be identified as message payload.
`messageHeader`	Specifies the JSON tree to be identified as a message header. Attributes from the headers will be added to the generated event.
`jsonNestedPayload`	Specifies JSON tree within a nested JSON or JSON string to be identified as message payload. `messagePayload` is used to specify the path to the attribute which value has the nested JSON string.
`jsonNestedHeader`	Similar to **`jsonNestedPayload`**, this parameter specifies the JSON tree within a nested JSON or JSON string to be identified as message header. `messagePayload` is used to specify the path to the attribute which value has the nested JSON string.
`messageDepth`	Specifies the number of levels in the message to traverse during parsing.

Parsing a JSON event with child nodes

The following sample shows a JSON event with child nodes.

```
{
    "message" : {
        "alarm": {
            "type":1,
            "title":"Bluemix Alert SEV2 - ibm.env5_syd.fabric.bosh.memoryPercent : st_bosh_mem_high memory percent > 93",
            "message":"Metrics: ibm.env5_syd.fabric.bosh.memoryPercent, Situation: st_bosh_mem_high ",
```

```
            "target":"ibm.env5_syd.fabric.bosh.memoryPercent",
            "situation":"st_bosh_mem_high",
            "isrecovered":false,
            "severity":2,
            "receivers":"administrator@mydomain.com",
            "timestamp":1481871659514
        }
    }
}
```

Given the sample JSON event as shown above, the parser can be configured to produce the following output for further rules files parsing and mapping to Object Server fields.

Table 7. Parser configuration for a JSON event with child nodes

Parser configuration	Parser output (key=value)
messagePayload= json.message.alarm	isrecovered=false message=Metrics: ibm.env5_syd.fabric.bosh.memoryPercent, Situation: st_bosh_mem_high receivers=administrator@mydomain.com resync_event=false severity=2 situation=st_bosh_mem_high target=ibm.env5_syd.fabric.bosh.memoryPercent timestamp=1481871659514 title=Bluemix Alert SEV2 - ibm.env5_syd.fabric.bosh.memoryPercent : st_bosh_mem_high memory percent > 93 type=1
messagePayload=json.message	alarm.isrecovered=false alarm.message=Metrics: ibm.env5_syd.fabric.bosh.memoryPercent, Situation: st_bosh_mem_high alarm.receivers=administrator@mydomain.com alarm.severity=2 alarm.situation=st_bosh_mem_high alarm.target=ibm.env5_syd.fabric.bosh.memoryPercent alarm.timestamp=1481871659514 alarm.title=Bluemix Alert SEV2 - ibm.env5_syd.fabric.bosh.memoryPercent : st_bosh_mem_high memory percent > 93 alarm.type=1 resync_event=false

Parsing a JSON event with an array

The sample JSON below contains a `data` attribute which is an array of data elements.

```
{
  "data": [
    {
      "id": "1222383102379613532",
      "type": "FilteredAlarm",
      "attributes": {
        "id": "1234567",
        "node-id": "node01",
        "state": "ACTIVE",
        "resource": "SWITCH",
        "condition-severity": "CRITICAL",
        "first-raise-time": "2017-07-06T13:43:19.000+0000",
        "last-raise-time": "2017-07-06T13:43:19.000+0000",
        "number-of-occurrences": 1,
        "acknowledge-state": "NOT_ACKNOWLEDGED"
        }
    },
    {
      "id": "-4067638085156070319",
      "type": "FilteredAlarm",
      "attributes": {
        "id": "7654321",
        "node-id": "node02",
        "state": "ACTIVE",
        "resource": "SWITCH",
        "condition-severity": "CRITICAL",
```

```
        "first-raise-time": "2017-07-06T13:42:59.000+0000",
        "last-raise-time": "2017-07-06T13:42:59.000+0000",
        "number-of-occurrences": 1,
        "acknowledge-state": "NOT_ACKNOWLEDGED"
        }
    }
  ]
}
```

Given the previous sample JSON event, the parser can be configured with the following.

Note : If the `messagePayload` is set to the parent attribute of the payload (json or the root), the array element will be indexed. When messagePayload is pointed to an array node, each array element becomes a single event.

Table 8. Parser configuration for a JSON event with an array

Parser configuration	Parser output (key=value)
messagePayload=json.data	attributes.acknowledge-state=NOT_ACKNOWLEDGED attributes.condition-severity=CRITICAL attributes.first-raise-time=2017-07-06T13:43:19.000+0000 attributes.id=1234567 attributes.last-raise-time=2017-07-06T13:43:19.000+0000 attributes.node-id=node01 attributes.number-of-occurrences=1 attributes.resource=SWITCH attributes.state=ACTIVE id=1222383102379613532 resync_event=false type=FilteredAlarm
messagePayload=json	data.0.attributes.acknowledge-state=NOT_ACKNOWLEDGED data.0.attributes.condition-severity=CRITICAL data.0.attributes.first-raise-time=2017-07-06T13:43:19.000+0000 data.0.attributes.id=1234567 data.0.attributes.last-raise-time=2017-07-06T13:43:19.000+0000 data.0.attributes.node-id=node01 data.0.attributes.number-of-occurrences=1 data.0.attributes.resource=SWITCH data.0.attributes.state=ACTIVE data.0.id=1222383102379613532 data.0.type=FilteredAlarm data.1.attributes.acknowledge-state=NOT_ACKNOWLEDGED data.1.attributes.condition-severity=CRITICAL data.1.attributes.first-raise-time=2017-07-06T13:42:59.000+0000 data.1.attributes.id=7654321 data.1.attributes.last-raise-time=2017-07-06T13:42:59.000+0000 data.1.attributes.node-id=node02 data.1.attributes.number-of-occurrences=1 data.1.attributes.resource=SWITCH data.1.attributes.state=ACTIVE data.1.id=-4067638085156070319 data.1.type=FilteredAlarm resync_event=false

Parsing a JSON event with a Nested JSON String

The following sample JSON contains a `value` attribute which has a nested JSON string.

```
{
  "payload": {
    "id": 1,
    "body": {
      "value": "{\"header\":{\"timestamp\":\"2017-07-06T09:07:56Z\",\"event\":{\"alarm\":{\"id\":\"1234567\",\"resource\":
       \"4\",\"node-id\":\"node01\",\"condition-type\":\"Link Down\",\"condition-severity\":\"WARNING\",\"condition-source\":
       \"NETWORK\",\"condition-state\":\"ACTIVE\",\"additional-text\":\"Link
       Down\",\"first-raise-time\":\"2000-01-01T00:00:58.000+0000\",\"last-raise-time\":\"2000-01-01T00:00:58.000+0000\",\
       "number-of-occurrences\":1,\"acknowledge-state\":\"ACKNOWLEDGED\",\"acknowledge-update-time\":
       \"2017-07-06T09:07:56.463+0000\",\"additional-attrs\":{\"source\":\"EMS-1\"}},\"_type\":\"alarmAcknowledged\"}}}",
      "attributes": 1
    }
  },
```

```
  "event": "sample_alarm"
}
```

The following table shows several configurations and the generated output of the probe parser for rules files parsing.

Note : The **messagePayload** parameter is set to the attribute which value contains a JSON string and **jsonNestedPayload** is set to the attribute within the JSON string to be treated as a payload to create an event.

Table 9. Parser configuration for a JSON event with a nested JSON string

Parser configuration	Parser output (key=value)
messagePayload=json.payload.body.value jsonNestedPayload=json.header.event.alarm	acknowledge-state=ACKNOWLEDGED acknowledge-update-time=2017-07-06T09:07:56.463+0000 additional-attrs.source=EMS-1 additional-text=Link Down condition-severity=WARNING condition-source=NETWORK condition-state=ACTIVE condition-type=Link Down first-raise-time=2000-01-01T00:00:58.000+0000 id=1234567 last-raise-time=2000-01-01T00:00:58.000+0000 node-id=node01 number-of-occurrences=1 resource=4 resync_event=false
messagePayload=json.payload.body.value jsonNestedPayload=json.header.event	type=alarmAcknowledged alarm.acknowledge-state=ACKNOWLEDGED alarm.acknowledge-update-time=2017-07-06T09:07:56.463+0000 alarm.additional-attrs.source=EMS-1 alarm.additional-text=Link Down alarm.condition-severity=WARNING alarm.condition-source=NETWORK alarm.condition-state=ACTIVE alarm.condition-type=Link Down alarm.first-raise-time=2000-01-01T00:00:58.000+0000 alarm.id=1234567 alarm.last-raise-time=2000-01-01T00:00:58.000+0000 alarm.node-id=node01 alarm.number-of-occurrences=1 alarm.resource=4 resync_event=false

Limitations

This solution has the following limitations:

- Only supports amd64 architecture.
- Validated to run on IBM Cloud Private 3.1.0 or 3.1.1.
- Only supports JSON events from event sources.
- There is known issue on the IBM Cloud Private UI where YAML object keys with dot (.) character is not rendered correctly. This affects the `ingress.annotations` parameter where the keys usually contain dot character.

For details about the Message Bus Probe, see https://www.ibm.com/support/knowledgecenter/en/SSSHTQ/omnibus/probes/message_bus/wip/concept/messbuspr_intro.html.

Troubleshooting

The following table describes how to troubleshoot issues when deploying the chart and how to resolve them.

Table 10. Problems

Problem	Cause	Resolution
Ingress annotation set during installation using ICP UI does not render correctly.	There is known issue on the IBM Cloud Private UI where YAML object keys with dot (.) character is not rendered correctly. This affects the `ingress.annotations` parameter where the keys usually contains dot character.	A fix will be delivered in the next ICP version. As a workaround, use the command-line to install the chart in order to set additional ingress annotations.
There are no events seen in the Object Server event list even though there are events sent to the probe endpoint and `Event Processor` log messages are seen in the probe debug log (needs to be configured to run in debug mode).	A potential cause is because the probe rules file is not configured correctly to parse tokens generated by the probe JSON parser.	By default, the probe starts using the default rules files. Configure the probe to use a custom rules file by creating a Configmap from a file. See Creating An Overriding Rules Config Map guide for more details.

Appendix A. Notices and Trademarks

This appendix contains the following sections:

- Notices
- Trademarks

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA

3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, ibm.com, AIX, Tivoli, zSeries, and Netcool are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Adobe, Acrobat, Portable Document Format (PDF), PostScript, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

SC27-9525-01