PUBLIC
SQL Anywhere - MobiLink
Document Version: 17.01.0 – 2021-10-15

MobiLink Server-Initiated Synchronization

THE BEST RUN

Content

1 MobiLink - Server-initiated Synchronization

This book describes MobiLink server-initiated synchronization, a feature that allows the MobiLink server to initiate synchronization or perform actions on remote devices.

In this section:

1.1 Server-initiated Synchronization

MobiLink server-initiated synchronization allows you to initiate synchronization from a consolidated database. You can send push notifications to remote databases, and cause remote databases to update the consolidated database.

This MobiLink component provides programmable options for detecting changes in the consolidated database to initiate synchronization, selecting devices to send push notifications to, and determining how devices react to those push notifications.

i Note

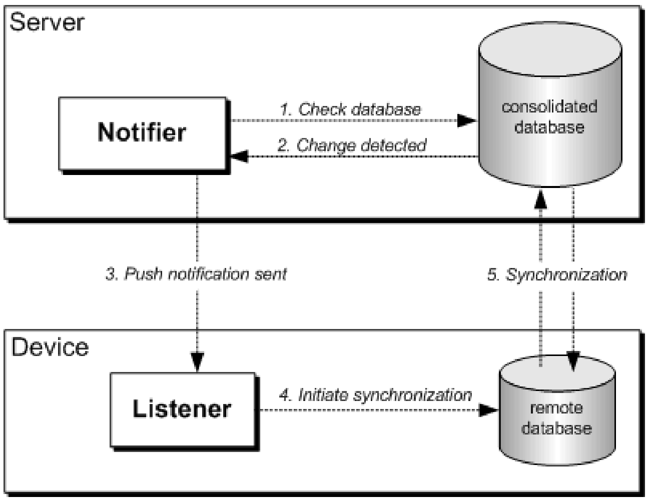
You can use SQL Central to administer remote databases, and then use server-initiated remote tasks (SIRT) as an alternative to server-initiated synchronization.

Example

A trucking organization issues mobile devices to their drivers. Each device runs a database that contains routes and delivery locations. When a driver submits a notice of a traffic disruption, the report is sent to a consolidated database. A server-side MobiLink component called a Notifier detects the report and sends a push notification to other drivers whose routes are affected by the disruption. This push notification causes the remote databases to synchronize so that the drivers can use an alternate route.

The server-initiated synchronization process

In the following illustration, the Notifier checks a consolidated database for changes. The Notifier sends a push notification to a device, resulting in the remote database being synchronized with the consolidated database.



During the server-initiated synchronization process, the following steps occur:

1. Using a query based on business logic, the Notifier checks a consolidated database for changes that need to be synchronized with the remote database.
2. When a change is detected, the Notifier prepares a push notification to send to the device.
3. The Notifier sends the push notification. Push notifications can be sent through a Device Tracker, UDP, SMTP, or SYNC gateway.
4. The Listener compares the subject, content, or sender against a message filter.
5. If the filter conditions are met, an action is initiated. For example, in a typical implementation, an action could run the MobiLink client or launch an UltraLite application.

In this section:

Related Information

Manage Remote Databases
Server-initiated Remote Tasks (SIRT)

1.1.1 Server-initiated Synchronization Components

MobiLink server-initiated synchronization requires push requests, a MobiLink Notifier, a MobiLink Listener, a lightweight poller, and/or a gateway.

Push requests

A push request is a row of values in a result set that tells a Notifier that you want to send a push notification to a device. Push requests cause server-initiated synchronizations to occur. Any database application can create push requests, including the Notifier. For example, a push request could be created using a database trigger that is activated when a price changes.

A MobiLink Notifier

A Notifier is a program integrated into the MobiLink server. It frequently checks the consolidated database for push requests. You can control how often the Notifier checks for push requests by specifying its properties. You must specify business logic to check for push requests, and to determine which devices should be notified. A push notification is sent to a device when the Notifier detects a push request.

A MobiLink Listener

A Listener is a program that runs on a device. It receives push notifications from the Notifier, then uses a message handler to filter messages and initiate an action. In a typical application, actions are synchronization calls, but applications are capable of performing other actions. You can configure the MobiLink Listener to act differently on push notifications from selected server sources, or on push notifications that contain specific content.

On Windows devices, the MobiLink Listener is an executable program that you configure with command line options. To receive push notifications, the device must be turned on and the MobiLink Listener must be running.

A lightweight poller

A lightweight poller is a device application that polls for push notifications at a specified time interval. Using a lightweight poller is an alternative to setting up a gateway, and is recommended because it does not require a persistent connection to the server, and can help extend battery life.

The MobiLink Listener is a lightweight poller that you can configure using MobiLink Listener command line options. Alternatively, you can use the lightweight polling API to create your own lightweight poller.

A gateway (An alternative to a lightweight poller)

A gateway provides a Notifier interface for sending push notifications to a device. Gateways are an alternative to lightweight pollers. You can send messages using a device tracking gateway, a SYNC gateway, a UDP gateway, or an SMTP gateway.

> i Note
>
> You can use to administer remote databases, and then use server-initiated remote tasks (SIRT) as an alternative to server-initiated synchronization.

Related Information

Push Requests [page 9]
Notifiers [page 16]
Gateways and Carriers [page 27]
Manage Remote Databases
MobiLink Listener Utility for Windows (dblsn) [page 61]
Lightweight Polling Option Setup [page 23]
Lightweight Polling API [page 91]
Server-initiated Remote Tasks (SIRT)

1.1.2 Server-initiated Synchronization Deployment Considerations

There are some issues to consider before deploying server-initiated synchronization applications.

Device Limitations When Using UDP Gateways

- The IP address on the device must be addressable directly from the MobiLink server.
- IP tracking for UDP notification does not work if the IP address on a Microsoft Windows device is not addressable directly from the MobiLink server.

Device Tracking Limitations

SQL Anywhere 9.0.0 or earlier MobiLink Listeners do not support device tracking. To use device tracking with these MobiLink Listeners, you must set up device tracking manually.

Supported Device Platforms

The MobiLink Listener is supported on Microsoft Windows and Microsoft Windows Mobile.

Related Information

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]

1.1.3 Quick Start to Server-initiated Synchronization

The following steps are required to set up server-initiated synchronization.

Before completing these steps, you must set up MobiLink for normal synchronization.

1. On the MobiLink server, prepare your consolidated database to store push requests.
2. On the MobiLink server, configure Notifier events to create and manage push requests.
3. On the device, set up a lightweight poller.
   If you do not want to use a lightweight poller, set up a supported gateway on the MobiLink server. When using an SMTP gateway, you also need to configure a carrier.
4. On the device, set up a MobiLink Listener to filter messages and perform actions.

Other resources

- Sample applications are installed to the *%SQLANYSAMP17%*`\MobiLink\` directory. All applications related to server-initiated synchronization are located in directories with the *SIS_* prefix.

Related Information

Notifier Events and Properties Configuration [page 17]
Lightweight Pollers [page 26]
Gateways and Carriers [page 27]
Message Handlers [page 19]

1.2 Server-initiated Synchronization Setup

Setting up server-initiated synchronization involves setting up push requests, MobiLink Notifiers, MobiLink Listeners, lightweight pollers, and gateways and carries.

In this section:

1.2.1 Push Requests

A push request is a row of values in a result set that a Notifier checks to determine if push notifications need to be sent to a device.

A Notifier places the push request inside a push notification then sends the push notification. In a typical server-initiated synchronization set up, a push request contains message content and target device information. Before a push notification can be sent, you need to configure a Notifier event so that the Notifier can detect the push request.

In this section:

Before push requests can be generated, your consolidated database must contain the push request columns required for server-initiated synchronization, and you must be able to obtain the values with a single database query.

1.2.1.1 Push Request Requirements

The requirements for push requests are dependent on the method the MobiLink server uses to communicate with devices. All push requests require subject and content columns.

If you are using lightweight pollers to poll for push notifications, you must create a poll key column to identify them.

If you are using gateways to send push notifications, you must create gateway and address columns.

You do not need to create push request columns if they already exist on your system. After you have satisfied the push request requirements, you are can work with them.

Push Request Requirements When Using Lightweight Pollers (Recommended)

Create the following columns when you are using lightweight pollers to poll for push notifications:

Column	Type	Description
Poll key	VARCHAR	The key used to identify a lightweight poller. Each lightweight poller sends a unique key to identify itself on the Mobi-Link server.
Subject	VARCHAR	The subject line of the message.
Content	VARCHAR	The content of the message.

Push Request Requirements When Using Gateways

Unless otherwise specified, create the following columns when using gateways to send push notifications:

Column	Type	Description
Request ID	INTEGER	Optional. The unique ID of a push re-quest. This column name is required for some Notifier events.

Column	Type	Description
Gateway	VARCHAR	The name of the gateway to which the message is sent.
Subject	VARCHAR	The subject line of the message.
Content	VARCHAR	The content of the message.
Address	VARCHAR	The destination address of a device.
Resend interval	VARCHAR	Optional. The time interval between message resends. The resend interval is useful when using a UDP gateway on an unreliable network. The Notifier assumes that all attributes associated with the push requests do not change; subsequent updates are ignored after the first poll of the request. The Notifier automatically adjusts the next polling interval if a push notification must be sent before the next polling time. You can stop a push request from being sent using synchronization logic in the request_cursor event. Delivery confirmation from the intended MobiLink Listener may stop a subsequent resend.
Time to live	VARCHAR	Optional. The time until the resend expires.

Example

The following example satisfies the push request requirements for using lightweight polling by creating the necessary columns in a SQL Anywhere consolidated database table:

```
CREATE TABLE PushRequest (
    req_id INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
    poll_key VARCHAR(128),
    subject VARCHAR(128),
    content VARCHAR(128)
)
```

You only need to create this table, or something like it, if the push request columns are not available elsewhere. These columns can exist across multiple tables, in existing tables, or in a view.

Related Information

Notifier Events [page 40]
Notifier Events and Properties Configuration [page 17]
How to Work with Push Requests [page 12]
request_cursor Event [page 46]

1.2.1.2 How to Work with Push Requests

Before push requests can be generated, your consolidated database must contain the push request columns required for server-initiated synchronization, and you must be able to obtain the values with a single database query.

Push requests are generated automatically when you provide a database query in the request_cursor event that selects push request columns.

Push Request Limitations

The following table lists the push request limitations for each column:

Column	Type	Limitation
Request ID	INTEGER	This value must be a unique primary key.
Poll key	VARCHAR	Only required when using lightweight pollers. There are no limitations on the poll key.
Gateway	VARCHAR	Only required when using gateways. This value must be set to the name of an enabled gateway. You specify your own custom gateway name, or choose one of the following preconfigured gateway names: • *Default-DeviceTracker* • *Default-SMTP* • *Default-SYNC* • *Default-UDP*

Column	Type	Limitation
Subject	VARCHAR	Avoid using non-alphanumeric characters when setting this value. Braces, chevrons, double quotations, parenthesis, single quotations, and square brackets are reserved for internal use, and should not be used in the subject column.
Content	VARCHAR	There are no limitations on the message content.
Address	VARCHAR	Only required when using gateways. For UDP gateways, this value should be an IP address or hostname. Port number suffixes are supported in the following formats: • `IP-address:port-number` • `hostname:port-number` For SMTP gateways, this value should be an email address. For SYNC and device tracking gateways, this value should be the recipient name defined with the MobiLink Listener -t+ option.
Resend interval	VARCHAR	By default, this value is measured in minutes. You can specify *S*, *M*, and *H* for units of seconds, minutes, and hours, respectively. You can also combine units; for example, `1H 30M 10S` informs the Notifier to resend the messages every one hour, thirty minutes, and ten seconds. If this value is null or not specified, the default is to send exactly once, with no resend.

Column	Type	Limitation
Time to live	VARCHAR	By default, this value is measured in minutes. You can specify *S*, *M*, and *H* for units of seconds, minutes, and hours, respectively. You can also combine units; for example, `3H 30M 10S` informs the Notifier to stop resending messages three hours, thirty minutes, and ten seconds after the initial send. If this value is null or not specified, the default is to send exactly once, with no resend.

Detecting Push Requests and Sending Push Notifications

A Notifier detects a push request by frequently firing the request_cursor event. By default, a script is not specified for this event; you must provide a request_cursor event script so that the Notifier can detect push requests. In a typical application, a request_cursor event script is a SELECT statement.

The following example uses the ml_add_property system procedure to create a request_cursor event script for a custom Notifier named Simple. The SELECT statement informs the Notifier to detect push requests from a table named PushRequest.

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'request_cursor',
    'SELECT poll_key, subject, content FROM PushRequest'
);
```

i Note

You must select columns in the same order as they are specified in the push request.

Deleting Push Requests

The Notifier resends push notifications if the information about the notified device is never updated after being sent and satisfied according to your business rules. Once push requests are satisfied, you need to prevent the Notifier from detecting old push requests. You can delete the push requests using a synchronization script if the push notifications were sent for synchronization purposes.

You can use the request_delete event to delete push requests by their request ID, however, your push request must contain a request ID column, and you must enable delivery confirmation.

Example

Push request generation example for a lightweight poller - Assume that a remote device is identified on your MobiLink server as unique_device_ID, and your consolidated database contains a table named PushRequest, which was created using the following SQL statement:

```
CREATE TABLE PushRequest (
    req_id INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
    poll_key VARCHAR(128),
    subject VARCHAR(128),
    content VARCHAR(128)
)
```

In this example, you can execute the following SQL statement on the consolidated database to prepare a push request:

```
INSERT INTO PushRequest (poll_key, subject, content) VALUES ('unique_device_ID',
'synchronize', 'ASAP');
```

Using the above script to insert values into the PushRequest table does not generate a push request by itself. You must set up a database query in the request_cursor event on the MobiLink server so that the inserted values can be selected and a push request can be generated.

In this example, you can define the following SQL statement for the request_cursor event script on your MobiLink server:

```
SELECT poll_key, subject, content FROM PushRequest;
```

A push request is generated when the unique_device_ID device polls the server for push notifications and the request_cursor event detects data in the PushRequest table. When sent to the device, the push notification subject is defined as *synchronize*, and the content is defined as *ASAP*.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]
Gateways as an Alternative to Lightweight Pollers [page 27]
Tutorial: Configuring Server-initiated Synchronization Using Lightweight Polling [page 109]
Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]
Push Request Requirements [page 10]
request_cursor Event [page 46]
request_delete Event [page 48]
-t dblsn Option [page 76]

1.2.2 Notifiers

A Notifier is a program integrated into the MobiLink server that frequently checks the consolidated database for push requests. Once push requests are detected, it sends push notifications to devices.

A Notifier also executes a series of events, allowing you to create scripts for monitoring data, managing push requests, handling delivery confirmations, and handling errors.

Notifiers start when you first load the MobiLink server. You can have more than one Notifier running within a single instance of the MobiLink server. For an example of how to use multiple Notifiers, see the sample application located in the *%SQLANYSAMP17%*`\MobiLink\SIS_MultipleNotifier`.

If a Notifier loses the database connection, it attempts to recover the connection until it regains access. After recovery, the Notifier continues to operate with the same configuration settings.

In this section:

1.2.2.1 Notifiers in a MobiLink Server Farm

A Notifier can run on every MobiLink server in the farm; the Notifiers, together, ensure that there are no redundant push notifications to the same MobiLink Listener.

The mlsrv17 -lsc server option is used to pass information to other servers when they want to connect to the local MobiLink server.

This feature makes one Notifier the primary, and all other Notifiers secondary. The primary Notifier controls push notifications, either directly or indirectly, via the secondaries. The secondary Notifiers also route MobiLink Listener information to the primary Notifier, so it knows where the MobiLink Listeners are and how to reach them.

If the MobiLink server running the primary Notifier fails, the server farm chooses a new primary Notifier, and notifications continue.

MobiLink Listeners may connect to any MobiLink server in the farm without needing to know which is the primary server.

To use this feature, the following mlsrv17 command line options are required on all MobiLink servers in the farm:

- -lsc

- -notifier
- -zs

Example

On host001:

```
mlsrv17 -notifier -zs ml001 -lsc tcpip(host=host001;port=2439) ...
```

On host007:

```
mlsrv17 -notifier -zs ml007 -lsc tcpip(host=host007;port=2439) ...
```

Related Information

-lsc mlsrv17 Option
-notifier mlsrv17 Option
-zs mlsrv17 Option

1.2.2.2 Notifier Events and Properties Configuration

Notifier events allow you to embed scripts that manage the overall server-initiated synchronization process.

For example, you can configure Notifier events to perform the following tasks:

- Determine what, how, and to whom information is sent in a push request using the request_cursor event.
- Create push requests that respond to changes in the consolidated database using the begin_poll event. (Advanced usage)
- Delete push requests using the request_delete event. (If required)
- Track Notifier polls and clean up table data using the end_poll event. (Advanced usage)

Notifier properties are similar to events. While events manage the notification process, properties manage Notifier behavior. For example, Notifier properties determine how often the Notifier should poll the consolidated database, and if the Notifier should be enabled on startup. Notifier properties and events are configured as server-side settings.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]
Notifier Events [page 40]

1.2.2.3 Notifier Startup

When the MobiLink server loads, it starts all enabled Notifiers. To disable a Notifier, you must set the enable Notifier property value to false.

To start your Notifiers, use one of the following methods:

- Configure your Notifiers and run mlsrv17 with the -notifier option specified.
- If your settings are stored in a Notifier configuration file, load the database at the command line by running mlsrv17, and specify the file using the -notifier option. For example, to use a file called `myfirst.Notifier`, the following command configures the MobiLink server to use the properties and events specified in the file:

  ```
  mlsrv17 ... -notifier c:\myfirst.Notifier
  ```

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]
Notifier Events and Properties Configuration [page 17]
Notifier Properties [page 54]
-notifier mlsrv17 Option

1.2.3 Listeners

A Listener is a program that runs on a device. It receives push notifications from a Notifier and initiates actions. Listeners can upload device tracking information to the consolidated database when using a gateway for server-initiated synchronization.

Example

The following command starts the MobiLink Listener utility for Windows devices:

```
dblsn -v2 -m -ot dblsn.log
    -l "poll_connect='host=localhost';
    poll_notifier=notifier_name1;
    poll_key=sis_user1;
    poll_every=10;
    subject=sync;
    action='start dbmlsync.exe
        -c SERVER=rem1;UID=DBA;PWD=passwd
        -ot dbmlsyncOut.txt -qc';"
```

This command loads a MobiLink Listener with verbosity set to level 2, enables message logging, and specifies that the server is located at *localhost*. The `dblsn.log` file is truncated before output is written to it. The

MobiLink Listener polls for push notifications every 10 seconds. If the MobiLink Listener receives a push notification where the subject is named `sync`, the MobiLink client application is launched.

In this section:

Related Information

Device Tracking Gateways [page 28]
MobiLink Listener Utility for Windows (dblsn) [page 61]
MobiLink Listener Options for Microsoft Windows [page 64]

1.2.3.1 Message Handlers

A message handler is a MobiLink Listener component that scans the message contents of a push notification to initiate an action. It can also be used to specify lightweight polling options, such as the server location and the polling frequency.

Message handlers consist of the following components:

Filter keywords

After a push notification is preprocessed, you can use filter keywords to scan the message contents. When a filter condition is met, an action is initiated. For example, you can specify the *subject* keyword to filter a message that contains a specific subject, or you can specify the *sender* keyword to filter messages received from a specific MobiLink server.

An action

An action is initiated after filter conditions are met on a message. In a typical application, you specify an action to initiate synchronization, but you can also perform other operations. To assist with error processing, you can specify an alternative action to handle instances when the original action fails.

Poll settings

Poll settings allow you to configure how the MobiLink Listener polls the MobiLink server for push notifications.

Options

Options allow you to control remote settings, such as delivery and action confirmation.

You can create a message handler with the dblsn -l option. Multiple message handlers can be specified.

In this section:

When a push notification is received by a MobiLink Listener, it extracts a message, which is split up and divided into several keywords.

Advanced: Remote ID as a Filter [page 24]
: You can filter messages by remote ID using the dblsn -r option, and the $remote_id action variable.

Advanced: Connectivity-initiated Synchronization [page 25]
: On Windows devices, you can initiate synchronization when connectivity changes.

Related Information

-l dblsn Option [page 71]
MobiLink Listener Keywords for Microsoft Windows [page 82]
MobiLink Listener Action Commands for Microsoft Windows [page 84]

1.2.3.1.1 Message Filters

When a push notification is received by a MobiLink Listener, it extracts a message, which is split up and divided into several keywords.

The *message* keyword contains the entire message in a raw format. The message is then divided into *subject*, *content*, and *sender* keywords. These keywords are run through your message filter to determine which actions to initiate.

A filter keyword is used to compare part of a push notification to a user-defined phrase. If the two phrases are textually equivalent, then an action is initiated.

Filter keywords can be specified by running the MobiLink Listener with the following syntax:

```
dblsn ... -l "filter-keyword-name='content to filter';action='...'"
```

You can use the -l option multiple times to create multiple filters, but you must also specify an action for every -l instance. Actions are only initiated when all filters are satisfied.

Each of the following keywords can only appear once in a message handler:

content

This and the subject keyword are recommended for filtering messages. Use this keyword to filter messages based on their content. For example:

```
dblsn -l "content='your content filter here';action='...'"
```

subject

This and the content keyword are recommended for filtering messages. Use this keyword to filter messages based on their subject. For example:

```
dblsn -l "subject='your subject filter here';action='...'"
```

message

Use this keyword to filter messages based on their raw data. Your filter value must match the exact length of the message. This keyword is not recommended since it has a variable structure.

message_start

Use this keyword to filter messages based on part of their raw data, starting from the beginning.

When you specify this keyword, the MobiLink Listener creates the $message_start and $message_end action variables.

sender

Use this keyword to filter messages based on their sender. This keyword is useful for tracking push notifications sent by a particular Notifier. The value is dependent on the gateway being used. For UDP gateways, it is the IP address of the host of the gateway. For SYNC gateways, it is *MobiLink*. For SMTP gateways, it depends on your wireless carrier.

In this section:

Initiation of Actions [page 21]
When a message matches the conditions of a filter, an action is initiated.

Action Variables [page 22]
Action variables allow you to reference parts of a push notification from a message filter or an action.

Lightweight Polling Option Setup [page 23]
You can use message handlers to handle polling operations. The lightweight polling options allow you to specify the location of the server, the Notifier name, the polling frequency, and a poll key. Alternatively, you can use the lightweight polling API to specify these properties.

Related Information

Gateways and Carriers [page 27]
MobiLink Listener Keywords for Microsoft Windows [page 82]
Advanced: Message Syntax [page 106]

1.2.3.1.1.1 Initiation of Actions

When a message matches the conditions of a filter, an action is initiated.

Actions can be specified by running the MobiLink Listener with the following syntax:

```
dblsn ... -l "...;action='action-command command statement'"
```

The following action commands allow you to perform different tasks when a message is filtered:

START

Start an application, and allow it to run in the background.

RUN

Run an application and wait for it to complete before receiving more push notifications.

POST

Post a window message to a process that is already running. This command can only be used on Windows devices.

SOCKET

Send a message to an application using a TCP/IP connection.

DBLSN FULL SHUTDOWN

Shut down the MobiLink Listener.

Related Information

MobiLink Listener Action Commands for Microsoft Windows [page 84]
Action Variables [page 22]

1.2.3.1.1.2 Action Variables

Action variables allow you to reference parts of a push notification from a message filter or an action.

How Action Variables Are Set

Most action variables are set automatically every time a push notification is received. The variable names are similar to the names specified in the message syntax. For example, `message` sets the $message action variable, while `subject` sets $subject, `sender` sets $sender, and `content` sets $content.

Using Action Variables

Action variables are used in the command line when you run the MobiLink Listener. How they are used is dependent on the message handler, and the action you want to initiate. The following example demonstrates the use of the RUN action command, which is used to initiate the MobiLink client application:

```
dblsn ... -l "subject=publish;action='RUN dbmlsync.exe @dbmlsync.txt -n
$content'"
```

This message handler filters messages where the subject is textually equivalent to "publish". Once filtered, dbmlsync is run with the -n option, passing the $content action variable as a parameter. Assuming that `content` references the name of a synchronization publication, dbmlsync uses the publication to synchronize the device database with the consolidated database.

The following example demonstrates the use of an action variable to filter a message:

```
dblsn ... -l "subject=$content;action='RUN script.bat"
```

When *subject* is textually equivalent to *content*, this message handler filters messages. Once filtered, the device runs a custom batch script.

Related Information

Initiation of Actions [page 21]
MobiLink Listener Action Commands for Microsoft Windows [page 84]
MobiLink Listener Action Variables for Microsoft Windows [page 88]
Advanced: Remote ID as a Filter [page 24]
Advanced: Message Syntax [page 106]

1.2.3.1.1.3 Lightweight Polling Option Setup

You can use message handlers to handle polling operations. The lightweight polling options allow you to specify the location of the server, the Notifier name, the polling frequency, and a poll key. Alternatively, you can use the lightweight polling API to specify these properties.

Lightweight polling options can be specified by running the MobiLink Listener with the following syntax:

```
dblsn ... -l
    "poll_connect=protocol-options;
    poll_notifier=Notifier-name;
    poll_key=identifier-string;
    poll_every=number-of-seconds;..."
```

A single message handler can only contain one of each of the following options:

poll_connect

Use this option to specify the protocol options required to connect to the server. Alternatively, you can use the dblsn -x option to specify the default protocol options. The poll_connect option overrides the default protocol options for the message handler.

poll_notifier

Use this option to specify the Notifier used by the MobiLink server to handle push requests. This option is required since the MobiLink server can host multiple Notifiers.

poll_key

Use this option to identify the MobiLink Listener to the Notifier. The MobiLink server uses this value to send push notifications intended for the device. In a typical application, this value should be the remote ID of the device.

poll_every

Use this option to specify how often the MobiLink Listener should poll the Notifier. By default, the MobiLink Listener automatically retrieves this value from the MobiLink server. This value is measured in seconds.

Related Information

Notifier Events and Properties Configuration [page 17]
Lightweight Pollers [page 26]
Push Request Requirements [page 10]
MobiLink Listener Keywords for Microsoft Windows [page 82]
Lightweight Polling API [page 91]

1.2.3.1.2 Advanced: Remote ID as a Filter

You can filter messages by remote ID using the dblsn -r option, and the $remote_id action variable.

When synchronizing a SQL Anywhere remote database for the first time, a remote ID file containing the ID of your database is created. The file name is the same as the database, but has the `.rid` extension, and is stored in the same directory as the database. For UltraLite databases, there is no remote ID file; the remote ID is extracted from the database directly.

When you start the MobiLink Listener, use the dblsn -r option to provide the name and location of the remote ID file or the UltraLite database, then use the dblsn -l option to create your message handler.

You can type the remote ID directly into your message filter. However, remote IDs are GUID by default; the remote ID is not easy to remember unless you provide a meaningful name.

> **i Note**
>
> In the dblsn command line, you can specify multiple instances of the -r and -l options. The $remote_id action variable used in a -l option is always specified in the -r option that precedes it. So, it is important to specify the -r option before the -l option.

The following example demonstrates the use of multiple remote IDs. It assumes that your device has a SQL Anywhere database called `business.db`, and an UltraLite database called `personal.udb`. In this example, **`ulpersonal`** is the window class name of the UltraLite application.

```
dblsn ... -r "c:\app\db\business.rid"
    -l "subject=$remote_id;action='dbmlsync.exe -k -c dsn=business';"
    -r "c:\ulapp\personal.udb"
    -l "subject=$remote_id;action=post dbas_synchronize to ulpersonal;"
```

Related Information

Remote IDs
Action Variables [page 22]
-r dblsn Option [page 75]
MobiLink Listener Action Variables for Microsoft Windows [page 88]

1.2.3.1.3 Advanced: Connectivity-initiated Synchronization

On Windows devices, you can initiate synchronization when connectivity changes.

When IP connectivity is gained or lost, the device sends a push notification to the MobiLink Listener with the message *_IP_CHANGED_*. When the device finds a new optimum path to the MobiLink server, it sends a push notification to the MobiLink Listener with the message *_BEST_IP_CHANGED_*. Using a message handler, you can detect these changes in connectivity and initiate an action.

Identifying Any Change in Connectivity

The *_IP_CHANGED_* message indicates that a change in IP connectivity has occurred. A change usually occurs when a device is within range of a Wi-Fi network, when the user makes a RAS connection, or when the user puts the device in a cradle. You can reference the *_IP_CHANGED_* message by running the MobiLink Listener with the following syntax:

```
dblsn ... -l "message=_IP_CHANGED_;action='...'"
```

The following example demonstrates how to use the *_IP_CHANGED_* message. The message handler filters the message, and sends it to the server. If the connection is lost, an error is generated.

```
dblsn -l "message=_IP_CHANGED_;
    action='
        SOCKET port=12345;
        sendText=IP changed: $adapters|$network_names;
        recvText=beeperAck;
        timeout=5';
    continue=yes;"
```

Identifying a Change in the Optimum Path to a MobiLink Server

The `_BEST_IP_CHANGED_` message indicates that a change in the optimum path to the MobiLink server has occurred. You can reference this message when you run the MobiLink Listener with the following syntax:

```
dblsn ... -x MobiLink-protocol-options -l
"message=_BEST_IP_CHANGED_;action='...'"
```

When filtering the `_BEST_IP_CHANGED_` message, the $best_ip action variable, which substitutes the local IP address that represents the best IP connection, can help you initiate useful actions. If there is no IP connection, $best_ip returns 0.0.0.0.

In the following example, the `_BEST_IP_CHANGED_` message is used to initiate a synchronization when the best IP connection changes. If the connection is lost, an error is generated.

```
dblsn -x http(host=mlserver.company.com)
   -v2 -m -i 3 -ot dblsn.log
   -l "message=_BEST_IP_CHANGED_;
       action='
           START dbmlsync.exe -ra -c SERVER=remote;UID=DBA;PWD=sql -n test_pub'"
```

i Note

When testing connectivity-initiated synchronization with your applications, run the MobiLink Listener on a separate computer from your MobiLink server.

Related Information

MobiLink Listener Utility for Windows (dblsn) [page 61]
MobiLink Listener Action Commands for Microsoft Windows [page 84]
MobiLink Listener Action Variables for Microsoft Windows [page 88]

1.2.4 Lightweight Pollers

A lightweight poller is a device application that polls for push notifications at a specified time interval.

Using a lightweight poller is an alternative to setting up a gateway, and is recommended because it does not require a persistent connection to the server like the SYNC gateway does, nor does it require a continuous connection like the UDP gateway does.

When a device polls the server, it sends a poll key and a Notifier name. The MobiLink server checks the Notifier name to determine which Notifier should check the cache for push requests. The poll key identifies the device to the Notifier, which uses the poll key to detect push requests intended for the device. Push notifications are sent after the push requests are detected.

Use MobiLink Listener command line options to configure a lightweight poller. Alternatively, use the lightweight polling API to integrate a lightweight poller into your device application.

i Note

You can use SQL Central to administer remote databases, and then use a server-initiated remote task (SIRT) to implement a push notification.

Related Information

Manage Remote Databases
Lightweight Polling API [page 91]
Lightweight Polling Option Setup [page 23]
MobiLink Listener Keywords for Microsoft Windows [page 82]
Server-initiated Remote Tasks (SIRT)

1.2.5 Gateways and Carriers

Gateways and carriers are MobiLink objects, stored in MobiLink system tables or a Notifier properties file. Gateways contain information about how to send messages for server-initiated synchronization and carriers contain information about a public carrier for use by server-initiated synchronization.

In this section:

1.2.5.1 Gateways as an Alternative to Lightweight Pollers

A gateway is a MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about how to send messages for server-initiated synchronization. They are an alternative to lightweight pollers and require a constant network connection.

Gateway properties are configured on a MobiLink server. You can configure multiple gateways on a single MobiLink server.

Supported Gateways

The following gateways are supported on the MobiLink server:

SYNC gateway

The SYNC gateway is a TCP/IP-based gateway; push notifications are sent through the same protocol as your MobiLink synchronizations.

The default SYNC gateway is named Default-SYNC. Typically, the default gateway settings do not need to be changed.

UDP gateway

The UDP gateway sends push notifications through a UDP gateway.

The default UDP gateway is named Default-UDP. Typically, the default gateway settings do not need to be changed. MobiLink Listeners use UDP by default when listening for push notifications.

SMTP gateway

The SMTP gateway sends push notifications using an email-to-SMS carrier service.

The default SMTP gateway is named Default-SMTP.

Device Tracking Gateway

In addition to the supported gateways, you can configure a device tracking gateway which automatically chooses the most appropriate gateway to send push notifications. The default device tracking gateway is Default-DeviceTracker. Use this gateway if you do not want to use a lightweight poller.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]
Device Tracking Gateways [page 28]
SYNC Gateway Properties [page 59]
UDP Gateway Properties [page 60]
SMTP Gateway Properties [page 58]

1.2.5.2 Device Tracking Gateways

Device tracking allows a MobiLink server to track devices using the remote ID information of a push request. A device tracking gateway uses automatically tracked IP addresses, phone numbers, and public wireless network provider IDs to deliver push notifications through SYNC, UDP, and SMTP gateways.

The gateway attempts to connect to the device using a SYNC gateway first. If delivery fails, a UDP gateway is attempted, followed by an SMTP gateway. This feature is useful when you expect device addresses to change.

A device tracking gateway can have a maximum of three subordinate gateways: one SYNC, one SMTP, and one UDP. Push notifications are automatically routed to one of the subordinate gateways based on the device tracking information sent by a MobiLink Listener. By enabling these subordinate gateways, device address changes are automatically managed by the MobiLink server. When an address changes, the MobiLink Listener synchronizes with the consolidated database to update the tracking information, which is located in the ml_device_address system table.

Most 9.0.1 or later MobiLink Listeners support device tracking. If you are using a MobiLink Listener that does not support device tracking, you can use a device tracking gateway by providing the tracking information.

In this section:

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]
Several system procedures are available to manually set up device tracking for 9.0.0 MobiLink Listeners. These procedures update the ml_device, ml_device_address, and ml_listening MobiLink system tables on the consolidated database.

Quick Start to Device Tracking Gateway Configuration [page 31]
The following procedure provides an overview of how to configure a device tracking gateway.

Carriers and Carrier Configuration [page 31]
A carrier is a MobiLink object that is stored in MobiLink system tables or a Notifier properties file, that contains information about a public carrier for use by server-initiated synchronization.

Related Information

Gateways as an Alternative to Lightweight Pollers [page 27]
Carriers and Carrier Configuration [page 31]

1.2.5.2.1 Setting up Device Tracking for 9.0.0 MobiLink Listeners

Several system procedures are available to manually set up device tracking for 9.0.0 MobiLink Listeners. These procedures update the ml_device, ml_device_address, and ml_listening MobiLink system tables on the consolidated database.

Context

You only need to support device tracking if you are using MobiLink Listeners running on SQL Anywhere 9.0.0 or earlier. Device tracking is already supported on all other MobiLink Listeners for Windows devices.

With manual device tracking, you can address recipients by MobiLink user name without providing network address information. However, the information cannot be automatically updated by MobiLink if it changes; you must change it manually. This method is especially useful for SMTP gateways because email addresses seldom change.

For UDP gateways, you cannot rely on static entries if your IP address changes every time you reconnect. You can resolve this problem by addressing the host name instead of IP address. However, this solution slows updates to DNS server tables and can misdirect push notifications. You can also set up system procedures to update the system tables programmatically.

Procedure

1. For each device, add a device record to the ml_device system table. For example:

```
CALL ml_set_device(
     'myWindowsMobile',
     'MobiLink Listeners for myWindowsMobile - 9.0.1',
     '1',
     'not used',
     'y',
     'manually entered by administrator'
);
```

   The first parameter, **`myWindowsMobile`**, is a unique user-defined device name. The second parameter contains optional remarks about the MobiLink Listener version. The third parameter specifies a MobiLink Listener version; use *0* for SQL Anywhere 9.0.0 MobiLink Listeners or *2* for post-9.0.0 MobiLink Listeners for Windows. The fourth parameter specifies optional device information. The fifth parameter specifies whether device tracking should be ignored. The final parameter contains optional remarks for this entry.

2. For each device, add an address record to the ml_device_address system table. For example:

```
CALL ml_set_device_address(
    'myWindowsMobile',
    'ROGERS AT&T',
    '55511234567',
    'y',
    'y',
    'manually entered by administrator'
);
```

   The first parameter, **myWindowsMobile**, is a user-defined unique device name. The second parameter is a network provider ID, which must match the network_provider_id carrier property. The third parameter is an IP address for UDP. The fourth parameter determines whether to activate this entry for sending push notifications. The fifth parameter specifies whether device tracking should be ignored. The final parameter contains optional remarks for this entry.
3. For each remote database, add a recipient record to the ml_listening system table for each device you added. This maps the device to the MobiLink user name. For example:

```
CALL ml_set_listening(
    'myULDB',
    'myWindowsMobile',
    'y',
    'y',
    'manually entered by administrator'
);
```

   The first parameter is a MobiLink user name. The second parameter is a user-defined unique device name. The third parameter determines whether to activate this entry for device tracking addressing. The fourth parameter specifies whether device tracking should be ignored. The final parameter contains optional remarks for this entry.

Results

The specified devices are set up for device tracking.

Related Information

ml_set_device System Procedure [page 101]
ml_set_listening System Procedure [page 104]
ml_set_device_address System Procedure [page 102]
Carrier Properties [page 61]

1.2.5.2.2 Quick Start to Device Tracking Gateway Configuration

The following procedure provides an overview of how to configure a device tracking gateway.

1. Set up a SYNC, UDP, or SMTP gateway.
   When you start the MobiLink server, these gateways are already set up with the default settings.

   > **i Note**
   >
   > The SMTP gateway requires carrier configuration.
2. Create a new Notifier and set up your request_cursor event with the following conditions:
   - The gateway name must be set to the name of a device tracking gateway you want to use. The default gateway is named Default-DeviceTracker. This name is represented by the first column of the result set.
   - The address name must be set to the remote ID of the device. Use the dblsn -t+ option to register the remote ID with the MobiLink server. This name is represented by the fourth column of the result set.
3. Add the MobiLink Listener name to the ml_user system table.
   The default MobiLink Listener name is `device_name` *-dblsn*, where `device_name` is the name of your device.
   Run the MobiLink Listener to view the device name, which can be found in the MobiLink Listener messages window. Alternatively, you can set the device name using the dblsn -e option, or set a different MobiLink Listener name using the dblsn -u option.
4. Start a MobiLink Listener with the required options.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]
Carriers and Carrier Configuration [page 31]
MobiLink User Creation and Registration
request_cursor Event [page 46]
MobiLink Listener Utility for Windows (dblsn) [page 61]
-u dblsn Option [page 79]
-e dblsn Option [page 69]

1.2.5.2.3 Carriers and Carrier Configuration

A carrier is a MobiLink object that is stored in MobiLink system tables or a Notifier properties file, that contains information about a public carrier for use by server-initiated synchronization.

You must configure a wireless carrier to send push notifications through an SMTP gateway because the Notifier needs to construct valid email addresses. You must also configure a wireless carrier when using a device tracking gateway with a subordinate SMTP gateway enabled.

Carrier properties, such as the network provider ID and the SMS email prefix, are configured on a MobiLink server. To accommodate multiple carrier services, configure multiple carriers on the MobiLink server.

Sender Syntax

When a push notification is received by a MobiLink Listener and preprocessed for message filtering, it is divided into several keywords. The *sender* keyword in a message is an email address, which is generated by the device and varies depending on the wireless carrier.

The sender syntax is in the following format:

```
sender = sms_email_user_prefix phone-number@sms_email_domain
```

> i Note
>
> There are no spaces between `sms_email_user_prefix` and `phone-number`.

The `sms_email_user_prefix` and `sms_email_domain` values are carrier properties that should be configured on the MobiLink server. The `phone-number` value is taken from the address column of the ml_device_address system table.

To determine the sender syntax, run the MobiLink Listener on a device that uses a carrier service. Enable message logging, and set verbosity to level 2 using the dblsn -m and -v options. Check the message log after loading the MobiLink Listener.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]
Carrier Properties [page 61]
Advanced: Message Syntax [page 106]

1.3 MobiLink Server Settings for Server-initiated Synchronization

Server-side settings consist of Notifier properties, gateway properties, carrier properties, and Notifier events.

To configure these settings, use one of the following methods:

- SQL Central
- A Notifier configuration file
- The ml_add_property system procedure

The SQL Central and ml_add_property system procedure methods add events and settings to the ml_property system table.

> i Note
>
> Changes made to server-side settings do not take effect while the MobiLink server is running. To apply new settings, you must shut down and restart the MobiLink server.

If you have already configured server-side settings in the ml_property system table and want to use a Notifier configuration file, the system table settings are always loaded first, followed by the file settings. The Notifier configuration file overwrites existing server-side settings, but the changes are not permanently applied to the consolidated database.

In this section:

1.3.1 Server-side Settings Configured Using the ml_add_property System Procedure

Use the ml_add_property system procedure to configure the server-side settings of a SQL Anywhere consolidated database. You can set these properties and events using Interactive SQL.

> **i Note**
>
> You must use the ANSI standard when naming your notifiers, gateways, and carriers.

Common Properties Syntax

```
CALL ml_add_property('SIS', '', 'Property', Value);
```

Notifier Properties and Events Syntax

```
CALL ml_add_property('SIS', 'Notifier(NotifierName)', 'Event-or-Property',
Value);
```

Gateway Properties Syntax

```
CALL ml_add_property('SIS', 'DeviceTracker(DeviceTrackerName)', 'Property',
Value);
```

```
CALL ml_add_property('SIS', 'SMTP(SMTPName)', 'Property', Value);
```

```
CALL ml_add_property('SIS', 'UDP(UDPName)', 'Property', Value);
```

```
CALL ml_add_property('SIS', 'SYNC(SYNCName)', 'Property', Value);
```

Carrier Properties Syntax

```
CALL ml_add_property('SIS', 'Carrier(CarrierName)', 'Property', Value);
```

Related Information

ml_add_property System Procedure

1.3.2 Setting up a Notifier, Gateway, or Carrier Using SQL Central

SQL Central provides a graphical user interface for modifying properties and events. You can use SQL Central to configure multiple Notifiers, gateways, and carriers.

Context

You must use the ANSI standard when naming your notifiers, gateways, and carriers.

By configuring your server-side settings through SQL Central, you do not need to specify a Notifier configuration file at the command line when using the mlsrv17 -notifier option.

Procedure

1. In SQL Central, use the MobiLink plug-in to create a MobiLink project for your consolidated database if you have not already created one.
2. Click *View* > *Folders*.
3. In the left pane, expand *MobiLink 17*, your MobiLink project name, *Consolidated databases*, your consolidated database name, and then select *Notification*.

   In the right pane, all available Notifiers, gateways, and carriers appear.
4. Create new Notifiers, gateways, and carriers.
   - To create a new Notifier, click the *Notifiers* tab in the right pane, then click *File* > *New* > *Notifier*.
   - To create a new gateway, click the *Gateways* tab in the right pane, then click *File* > *New* > *Gateway*.
   - To create a new carrier, click the *Carriers* tab in the right pane, then click *File* > *New* > *Carrier*.
5. Select a Notifier, gateway, or carrier to configure.
   - To set up Notifier properties or events, click the *Notifiers* tab in the right pane and choose the Notifier that you want to configure.
   - To set up gateway properties, click the *Gateways* tab in the right pane and choose the gateway that you want to configure.
   - To set up carrier properties, click the *Carriers* tab in the right pane and choose the carrier that you want to configure.

   Click *File* > *Properties*.

   A window appears where you can adjust all settings applicable to the chosen Notifier, gateway, or carrier.
6. Click *OK*.

Results

The Notifier, gateway, or carrier is set up and ready to use.

In this section:

Importing Server-side Settings from a Notifier Configuration File [page 36]
: Use a Notifier configuration file to import server-side settings into the ml_property_table.

Exporting Server-Side Settings to a Notifier Configuration File [page 37]
: You can export the server-side settings from the ml_property table into a Notifier configuration file. By exporting the settings, you can create multiple versions of your server-side settings, and you can load a different version using the mlsrv17 -notifier option.

Related Information

Creating a MobiLink Project

1.3.2.1 Importing Server-side Settings from a Notifier Configuration File

Use a Notifier configuration file to import server-side settings into the ml_property_table.

Procedure

1. In SQL Central, use the MobiLink plug-in to create a MobiLink project for your consolidated database if you have not already created one.
2. Click *View* > *Folders*.
3. In the left pane, expand *MobiLink 17*, your MobiLink project name, *Consolidated databases*, your consolidated database name, and then select *Notification*.
4. Click *File* > *Import Settings*, and follow the instructions in the wizard.

Results

The settings are imported from the Notifier configuration file into the ml_property_table.

Related Information

Creating a MobiLink Project

1.3.2.2 Exporting Server-Side Settings to a Notifier Configuration File

You can export the server-side settings from the ml_property table into a Notifier configuration file. By exporting the settings, you can create multiple versions of your server-side settings, and you can load a different version using the mlsrv17 -notifier option.

Procedure

1. In SQL Central, use the MobiLink plug-in to create a MobiLink project for your consolidated database if you have not already created one.
2. Click *View* > *Folders*.
3. In the left pane, expand *MobiLink 17*, your MobiLink project name, *Consolidated databases*, your consolidated database name, and then select *Notification*.
4. Click *File* > *Export Settings*, and follow the instructions in the wizard.

Results

The specified settings are exported to a Notifier configuration file.

Related Information

Creating a MobiLink Project

1.3.3 Server-side Settings Configured Using the Notifier Configuration File

Server-side settings can be stored in a Notifier configuration file. You can use this file to configure multiple Notifiers, gateways, and carriers.

> **Note**
>
> You must use the ANSI standard when naming your notifiers, gateways, and carriers.

Creating and Configuring a Notifier Configuration File

A Notifier configuration file can be created using a text editor, or it can be generated from property and event settings exported from SQL Central.

To view the layout of a typical Notifier configuration file, open the *%SQLANYSAMP17%*`\MobiLink\template.Notifier` template file. The template file provides examples for configuring server-side properties and events.

When you have configured the necessary settings, save the Notifier configuration file and load your server-side properties and events into the MobiLink server.

Common Properties Syntax

```
Property = Value
```

Notifier Events Syntax

```
Notifier(NotifierName).Event = \
# Replace this text with SQL script.          \
# Be sure to put a backslash (\) at           \
# the end of every line of code               \
# if your event requires multiple             \
# lines of text.
```

Notifier Properties Syntax

```
Notifier(NotifierName).Property = Value
```

Gateway Properties Syntax

```
# For Device tracking gateways:
DeviceTracker(DeviceTrackerName).Property = Value
# For SMTP gateways:
SMTP(SMTPName).Property = Value
# For SYNC gateways:
SYNC(SYNCName).Property = Value
# For UDP gateways:
UDP(UDPName).Property = Value
```

Carrier Properties Syntax

```
Carrier(CarrierName).Property = Value
```

Loading a Notifier Configuration File

To load a Notifier configuration file into the MobiLink server, run mlsrv17 from the command line with the -notifier option specified. For example, to use the server-side settings defined in a CarDealer.Notifier configuration file, run the following command:

```
mlsrv17 ... -notifier "c:\CarDealer.Notifier"
```

If a file is not specified, the `config.Notifier` file is loaded by default.

> i Note
>
> To use the default SYNC gateway, you cannot store server-side settings in a Notifier configuration file. You must store them in the ml_property system table using an alternative method.

Using Escape Sequences

The backslash (\ ) is the escape character. The following is a list of common escape sequences that you can use in a Notifier configuration file:

Escape sequence	Description
\b	Backspace
\t	Tab
\n	Line feed

Escape sequence	Description
\r	Carriage return
\"	Double quote (")
\'	Single quote (')
\\	Backslash (\ )
\e	Escape

Unicode escape sequences are of the form \uxxxx while ASCII escape sequences are of the form \xxx, where each x represents a hexadecimal digit.

When editing a property or event that requires multiple lines of text, add a single backslash character (\ ) at the end of each line.

Related Information

Setting up a Notifier, Gateway, or Carrier Using SQL Central [page 35]
-notifier mlsrv17 Option

1.3.4 Notifier Events

Events are fired whenever a Notifier polls a MobiLink Listener. When an event is fired, the SQL script associated with the event is executed. You can incorporate SQL script into any of the following Notifier events. Although scripting is optional, you must script the request_cursor polling event.

There are three classifications of Notifier events: polling events, connection events, and asynchronous events. Polling events are fired every time a Notifier checks the consolidated database, and include all the events that occur between a begin_poll event and an end_poll event. Connection events are fired during the Notifier database connection. Asynchronous events can be fired at any time during the synchronization process.

Unless otherwise specified, Notifier events can be configured using any of the recommended methods.

When a MobiLink Listener polls the Notifier, these events are fired in the following order:

```
Fire begin_connection event
For each poll (
    Fire begin_poll event
        Fire shutdown_query event
        Fire request_cursor event
        For all requests expired before required confirmation (
            Fire error_handler event
        )
        Fire request_delete event
    Fire end_poll event
)
Fire end_connection event
```

In this section:

1.3.4.1 Events During Polling

Polling events are a classification of Notifier events that are fired every time a Notifier checks the consolidated database. These events include all the events that occur between a begin_poll event and an end_poll event.

In this section:

1.3.4.1.1 begin_poll Event

This polling event accepts SQL script and is fired before the Notifier checks the consolidated database for push requests. The value is null by default, so this event is not fired.

Example

This example creates a push request for a Notifier named Notifier A. It uses a SQL statement that inserts rows into a table named PushRequest. Each row in this table represents a message to send to an address. The WHERE clause determines which push requests are inserted into the PushRequest table.

To use the ml_add_property system procedure with a SQL Anywhere consolidated database, run the following command:

```
ml_add_property(
    'SIS',
    'Notifier(Notifier A)',
    'begin_poll',
    'INSERT INTO PushRequest
        (gateway, mluser, subject, content)
        SELECT ''MyGateway'', DISTINCT mluser, ''sync'',
            stream_param
            FROM MLUserExtra, mluser_union, Dealer
            WHERE MLUserExtra.mluser = mluser_union.name
            AND (push_sync_status = ''waiting for request''
                OR datediff( hour, last_status_change, now() ) > 12 )
            AND ( mluser_union.publication_name is NULL
                OR mluser_union.publication_name =''FullSync'' )
            AND Dealer.last_modified > mluser_union.last_sync_time'
);
```

Related Information

Push Requests [page 9]
Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.4.1.2 end_poll Event

This polling event accepts SQL script and is fired after the Notifier checks the consolidated database for push requests. The value is null by default, so this event is not fired.

You can use this event to perform table cleanup or to log the results of a poll.

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.4.1.3 error_handler Event

Configure this event to indicate when a transmission fails or was not confirmed. For example, when a transmission fails, you can use this event to insert a row in an audit table or to send a push notification.

The following table details the parameters that can be captured using the error_handler event:

Script parameter	Type	Description
request_option (out)	Integer	Controls what the Notifier does to the push request after the error handler returns. The output can be one of the following values: • *0*: Perform default action based on the error code and log the error. • *1*: Do nothing. • *2*: Execute the request_delete event. • *3*: Attempt to deliver to a secondary gateway.
error_code (in)	Integer	Use one of the following values for the error code: • *-1*: The request timed out with confirmation of success. • *-8*: An error occurred during delivery attempt.
request_id (in)	Integer	Identifies the request.
gateway (in)	Varchar	Specifies the gateway associated with the push request.

Script parameter	Type	Description
address (in)	Varchar	Specifies the address associated with the push request. For security reasons, when the optional Notifier error handler is invoked, an asterisk (*) is substituted for any character in the subject or content of the notification that is not one of the following: • alphanumeric characters • period • colon • minus sign • plus sign • underscore The value sent in the notification is the same as the original value.
subject (in)	Varchar	Specifies the subject associated with the push request. For security reasons, when the optional Notifier error handler is invoked, an asterisk (*) is substituted for any character in the subject or content of the notification that is not one of the following: • alphanumeric characters • period • colon • minus sign • plus sign • underscore The value sent in the notification is the same as the original value.

Script parameter	Type	Description
content (in)	Varchar	Specifies the content associated with the push request. For security reasons, when the optional Notifier error handler is invoked, an asterisk (*) is substituted for any character in the subject or content of the notification that is not one of the following: • alphanumeric characters • period • colon • minus sign • plus sign • underscore The value sent in the notification is the same as the original value.

i Note

This event requires the use of a system procedure. You cannot configure this event directly using SQL Central.

Example

In the following example, you create a table called CustomError and log errors to the table using a stored procedure called CustomErrorHandler. The output parameter Notifier_opcode is always 0, which means that default Notifier handling is used.

```
CREATE TABLE CustomError(
    error_code  integer,
    request_id  integer,
    gateway  varchar(255),
    address  varchar(255),
    subject  varchar(255),
    content  varchar(255),
    occurAt  timestamp not null default timestamp
);
CREATE PROCEDURE CustomErrorHandler(
    out @Notifier_opcode integer,
    in @error_code   integer,
    in @request_id   integer,
    in @gateway    varchar(255),
    in @address    varchar(255),
    in @subject    varchar(255),
    in @content    varchar(255)
)
BEGIN
    INSERT INTO CustomError(
        error_code,
        request_id,
        gateway,
        address,
        subject,
        content)
    VALUES(
        @error_code,
        @request_id,
        @gateway,
        @address,
        @subject,
        @content
    );
    SET @Notifier_opcode = 0;
END
```

To use this ml_add_property system procedure with a SQL Anywhere consolidated database, run the following command:

```
call ml_add_property(
    'SIS',
    'Notifier(myNotifier)',
    'error_handler',
    'call CustomErrorHandler(?, ?, ?, ?, ?, ?, ?)');
```

Alternatively, you can fire this event by adding the following line to a Notifier configuration file:

```
Notifier(myNotifier).error_handler = call CustomErrorHandler(?, ?, ?, ?, ?, ?, ?)
```

Run the file using the mlsrv17 -notifier option.

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]
Server-side Settings Configured Using the Notifier Configuration File [page 38]

1.3.4.1.4 request_cursor Event

This polling event accepts SQL script and is fired to detect push requests. You must configure this event.

Fetching push requests when using a lightweight poller (recommended)

When this event contains up to three columns in a result set, the Notifier acknowledges that there is no persistent connection between the server and the device, and that a device must poll the Notifier before push notifications can be sent. The Notifier caches the result set before sending push notifications. The MobiLink server identifies the device by the poll key, which is sent by the device every time the device polls the Notifier.

The result set of this event must contain the following columns in the specified order:

- Poll key
- Subject (optional)
- Content (optional)

Fetching push requests when using a gateway

When this event contains more than three columns in a result set, the Notifier acknowledges that a persistent connection exists between the server and the device, and then sends push notifications using a gateway when push requests are detected.

The result set of this event must contain the following columns in the specified order:

- Request ID (optional)
- Gateway
- Subject
- Content
- Address
- Resend interval (optional)
- Time to live (optional)

Example

The following example uses the ml_add_property system procedure to create a request_cursor event script for a custom Notifier named Simple. The SELECT statement tells the Notifier to detect push requests from a table named PushRequest.

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'request_cursor',
    'SELECT poll_key,
        subject,
        content
    FROM PushRequest'
);
```

Include a WHERE clause in your script to filter out requests that have already been sent. For example, you can add a push request column to track the moment you inserted a request, and then use a WHERE clause in this event to filter out requests that were inserted before the last time the user synchronized.

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]
Push Request Requirements [page 10]

1.3.4.1.5 request_delete Event

This polling event accepts SQL script and is fired to perform cleanup operations when the need for push request deletion is detected.

It accepts the request ID as a parameter and is executed per request ID. Your request_cursor event must contain a request ID column to use the request_delete event. You can reference the request ID using a named parameter or a question mark (?). This event is optional if you have already assigned cleanup operations to another process or event, such as the end_poll event.

The Notifier can use the DELETE statement to remove the following forms of push requests:

Implicitly dropped

These push requests appeared previously but did not appear in the current set of requests obtained from the request_cursor event.

Confirmed

These are push requests confirmed as delivered.

Expired

These push requests expired based on their resend attributes and the current time. Requests without resend attributes are considered expired even if they appear in the next request.

You can use the request_delete event to prevent expired or implicitly dropped requests from being deleted. For example, the CarDealer sample in the *%SQLANYSAMP17%*`\MobiLink\SIS_CarDealer` directory uses the request_delete event to set the status field of the PushRequest table to 'processed'.

```
UPDATE PushRequest SET status='processed' WHERE req_id = ?
```

The begin_poll event in the sample uses the last synchronization time to check if remote devices are up-to-date before eliminating processed push requests.

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.4.1.6 shutdown_query Event

This polling event accepts SQL script and is fired after a begin_poll event. The return value specifies the shutdown state of the Notifier. The value is null by default, so this event is not fired.

To shut down the Notifier, set up your SQL script to return 'yes'; otherwise, set it to return 'no'. If the Notifier shuts down, the end_poll event is not fired.

When storing the shutdown state in a table, use the end_connection event to reset the state.

Example

The following example uses the ml_add_property system procedure to create a shutdown_query event script for a custom Notifier named Simple. The SELECT statement informs the Notifier to shut down if the tooManyNotifierErrors method returns true.

```
CALL ml_add_property('SIS', 'Notifier(Simple)', 'shutdown_query',
    'SELECT
        IF tooManyNotifierErrors() THEN
            ''yes''
        ELSE
            ''no''
        ENDIF'
);
```

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]
end_connection Event [page 50]

1.3.4.2 Connection Events

Connection events are a classification of Notifier events that are fired during the Notifier database connection.

In this section:

begin_connection Event [page 49]
    This event accepts SQL script and is fired after the Notifier connects to the consolidated database, but before it checks for push requests. The value is null by default, so this event is not fired.

end_connection Event [page 50]
    This event accepts SQL script and is fired just before the Notifier disconnects from the consolidated database. The value is null by default, so this event is not fired.

1.3.4.2.1 begin_connection Event

This event accepts SQL script and is fired after the Notifier connects to the consolidated database, but before it checks for push requests. The value is null by default, so this event is not fired.

You can use this event to create temporary tables or variables. You should not use this event to change isolation levels. To control isolation levels, use the isolation property.

If the Notifier loses the connection to the consolidated database, it re-runs this event immediately after reconnecting.

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]
Notifier Properties [page 54]

1.3.4.2.2 end_connection Event

This event accepts SQL script and is fired just before the Notifier disconnects from the consolidated database. The value is null by default, so this event is not fired.

You can use this event to clean up temporary storage, such as SQL variables and temporary tables.

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.4.3 Asynchronous Events

Asynchronous events are a classification of Notifier events that can be fired at any time during the synchronization process.

In this section:

1.3.4.3.1 confirmation_handler Event

Configure this event to handle delivery confirmation information uploaded by MobiLink Listeners. If the status parameter returns 0, then the push request identified by request_id was successfully received by the MobiLink Listener identified by the remote_device parameters.

You can use the request_option parameter to initiate an action in response to the delivery confirmation. If request_option is 0, the confirmation_handler event initiates the default action, where the request_delete event is executed to delete the original push request. If the device sending the delivery confirmation does not match the device identified by the request_id, the default action is to send the original push request through a secondary gateway.

i Note

Use the dblsn -x option to allow MobiLink Listeners to upload delivery confirmation information. Use the dblsn -ni option if you want delivery confirmation but do not want IP tracking.

i Note

This event requires the use of a system procedure. You cannot configure this event directly using the SQL Central method.

The following parameters can be captured using the confirmation_handler event:

Script parameter	Type	Description
request_option (out)	Integer	Controls what the Notifier does to the request after the handler returns. The following values can be returned: • *0*: Perform default Notifier action based on the value of the status parameter. If status indicates that the responding device is the target one, then the Notifier deletes the request; otherwise the Notifier attempts to deliver on a secondary gateway. • *1*: Do nothing. • *2*: Execute Notifier.request_delete. • *3*: Attempt to deliver to a secondary gateway.
status (in)	Integer	The situation summary. The status can be used during development to identify problems such as incorrect filters and handler attributes. The following values can be returned: • *0*: Received and confirmed. • *-2*: Right respondent but the message was rejected. • *-3*: Right respondent and the message was accepted but the action failed. • *-4*: Wrong respondent and the message was accepted. • *-5*: Wrong respondent and the message was rejected. • *-6*: Wrong respondent. The message was accepted and the action succeeded. • *-7*: Wrong respondent. The message was accepted but the action failed.

Script parameter	Type	Description
request_id (in)	Integer	The request ID. Your request_cursor event must contain a request ID column to use the confirmation_handler event.
remote_code (in)	Integer	The summary reported by the MobiLink Listener. The following values can be returned: • *1*: Message accepted. • *2*: Message rejected. • *3*: Message accepted and action succeeded. • *4*: Message accepted and action failed.
remote_device (in)	Varchar	The device name of the responding MobiLink Listener.
remote_mluser (in)	Varchar	The MobiLink user name of the responding MobiLink Listener.
remote_action_return (in)	Varchar	The return code of the remote action.
remote_action (in)	Varchar	Reserved for the action command.
gateway (in)	Varchar	The gateway associated with the request.
address (in)	Varchar	The address associated with the request.
subject (in)	Varchar	The subject associated with the request.
content (in)	Varchar	The content associated with the request.

Example

In the following example, you create a table called CustomConfirmation and then log confirmations to it using a stored procedure named CustomConfirmationHandler. The output parameter request_option is always set to 0, which means that default Notifier handling is used.

```
CREATE TABLE CustomConfirmation(
    error_code   integer,
    request_id   integer,
    remote_code   integer,
    remote_device  varchar(128),
    remote_mluser  varchar(128),
    remote_action_return varchar(128),
    remote_action  varchar(128),
    gateway   varchar(255),
    address   varchar(255),
    subject   varchar(255),
    content   varchar(255),
    occurAt   timestamp not null default timestamp
);
CREATE PROCEDURE CustomConfirmationHandler(
    out @request_option integer,
    in @error_code  integer,
    in @request_id  integer,
    in @remote_code  integer,
    in @remote_device  varchar(128),
    in @remote_mluser  varchar(128),
    in @remote_action_return varchar(128),
    in @remote_action  varchar(128),
    in @gateway   varchar(255),
    in @address   varchar(255),
    in @subject   varchar(255),
    in @content   varchar(255)
)
BEGIN
    INSERT INTO CustomConfirmation(
        error_code,
        request_id,
        remote_code,
        remote_device,
        remote_mluser,
        remote_action_return,
        remote_action,
        gateway,
        address,
        subject,
        content)
    VALUES (
        @error_code,
        @request_id,
        @remote_code,
        @remote_device,
        @remote_mluser,
        @remote_action_return,
        @remote_action,
        @gateway,
        @address,
        @subject,
        @content
    );
    SET @request_option = 0;
END
```

To use the ml_add_property system procedure with a SQL Anywhere consolidated database, run the following command:

```
call ml_add_property(
    'SIS',
    'Notifier(myNotifier)',
    'confirmation_handler',
    'call CustomConfirmation(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)');
```

Alternatively, you can call this event by adding the following line to a Notifier configuration file:

```
Notifier(myNotifier).confirmation_handler = call
CustomConfirmation(?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Run the file using the mlsrv17 -notifier option.

Related Information

Notifier Events [page 40]
MobiLink Server Settings for Server-initiated Synchronization [page 32]
Gateway Properties [page 56]
Server-side Settings Configured Using the Notifier Configuration File [page 38]
MobiLink Listener Options for Microsoft Windows [page 64]

1.3.5 Common Properties

Common properties are shared between Notifiers, gateways, and carriers. All common properties are optional.

Property	Value	Description
verbosity	{ *0* | *1* | *2* | *3* }	Specifies the verbosity level for Notifiers, gateways, and carriers. The following values can be used: • *0*: No trace. • *1*: Startup, shutdown, and property trace. • *2*: Display notifications. • *3*: Full-level trace. The default value is *0*.

1.3.6 Notifier Properties

Notifier properties allow you to change the behavior of a Notifier. All Notifier properties are optional.

Property	Value	Description
connect_string	`connection_string`	Overrides the default connection behavior used to connect to a database. The default value is *com.sap.ml.script.ServerContext*, which uses the connection string specified in the mlsrv17 command line. It may be useful to connect to another database when you want notification logic and data to be separate from your synchronization data. Most deployments do not set this property.

Property	Value	Description
enable	{ *yes* | *no* }	Specifies whether the Notifier should be enabled. All enabled Notifiers start when you run the -notifier mlsrv17 option.
gui	{ *yes* | *no* }	Specifies whether the Notifier window should be displayed while the Notifier is running. The default value is *yes*. This Notifier window allows users to temporarily change the polling interval, or to poll immediately. It can also be used to shut down the Notifier without shutting down the MobiLink server. Once stopped, the Notifier can only be restarted by shutting down and restarting the MobiLink server.
isolation	{ *0* | *1* | *2* | *3* }	Specifies the isolation level of the Notifier's database connection. The following values can be used: • *0*: Read uncommitted. • *1*: Read committed. • *2*: Repeatable read. • *3*: Serializable. The default value is *1*. Higher levels increase contention, but could adversely affect performance. Isolation level *0* allows reads of uncommitted data, which could get rolled back.
poll_every	`number`{ *s* | *m* | *h* }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: • *s*: Denotes seconds. • *m*: Denotes minutes. • *h*: Denotes hours. The default value is *1m*. Time units can be combined in the `HH` *h* `MM` *m* `SS` *s* format. If a time unit is not specified, time is measured in seconds.

Property	Value	Description
shared_database_connection	{ *yes* | *no* }	Specifies whether Notifiers should share database connections. The default value is *no*. Notifiers can only share connections when their isolation levels are the same. Specify *yes* to conserve resources without incurring performance penalties. Connection sharing is not possible in some situations, such as when applications use non-unique SQL variable names among Notifiers.

1.3.7 Gateway Properties

By default, four preconfigured gateways are created when you start the MobiLink server. They are installed when you run the MobiLink setup scripts for your consolidated database.

The default gateways are named as follows:

- Default-DeviceTracker gateway
- Default-SYNC gateway
- Default-UDP gateway
- Default-SMTP gateway

Do not remove the default gateways or change their names. Instead, you can create additional gateways with different names.

You should not need to change the properties defined in DefaultSYNC and DefaultUDP, but you must provide SMTP server information to the DefaultSYNC gateway. You should use the default gateways but, if required, you can use an alternative configuration.

In this section:

Device tracking Gateway Properties [page 57]
Device tracking gateway properties allow you to change the behavior of a device tracking gateway. All device tracking gateway properties are optional.

SMTP Gateway Properties [page 58]
SMTP gateway properties allow you to change the behavior of an SMTP gateway. The server property is required; all other SMTP gateway properties are optional.

SYNC Gateway Properties [page 59]
SYNC gateway properties allow you to change the behavior of a SYNC gateway. All SYNC gateway properties are optional.

UDP Gateway Properties [page 60]
UDP gateway properties allow you to change the behavior of an UDP gateway, such as the IP address and the port number. All UDP gateway properties are optional.

1.3.7.1 Device tracking Gateway Properties

Device tracking gateway properties allow you to change the behavior of a device tracking gateway. All device tracking gateway properties are optional.

Property	Value	Description
confirm_action	{ *yes* | *no* }	Specifies whether confirmation is sent on delivery through this gateway. The default value is *no*.
confirm_delivery	{ *yes* | *no* }	Specifies whether the MobiLink Listener should confirm with the consolidated database that the message was received. The default value is *yes*. The MobiLink Listener must be started with the -x MobiLink Listener option specified.
description	`description_text`	Describes the gateway.
enable	{ *yes* | *no* }	Specifies whether the device tracking gateway should be used.
smtp_gateway	`smtp_gateway_name`	Specifies the name of the SMTP subordinate gateway. The default value is *DefaultSMTP*. A device tracking gateway can only use one SMTP gateway. The gateway must be enabled.
sync_gateway	`sync_gateway_name`	Specifies the name of the SYNC subordinate gateway. The default value is *DefaultSYNC*. A device tracking gateway can only use one SYNC gateway. The gateway must be enabled.
udp_gateway	`udp_gateway_name`	Specifies the name of the UDP subordinate gateway. The default value is *DefaultUDP*. A device tracking gateway can only use one UDP gateway. The gateway must be enabled.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.7.2 SMTP Gateway Properties

SMTP gateway properties allow you to change the behavior of an SMTP gateway. The server property is required; all other SMTP gateway properties are optional.

Property	Value	Description
confirm_action	{ *yes* | *no* }	Specifies whether confirmation is sent on delivery through this gateway. The default value is *no*.
confirm_delivery	{ *yes* | *no* }	Specifies whether this gateway confirms delivery. The default value is *no*.
confirm_timeout	`number`{ *s* | *m* | *h* }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: • *s*: Denotes seconds. • *m*: Denotes minutes. • *h*: Denotes hours. The default value is *1m*. Time units can be combined in the `HH` *h* `MM` *m* `SS` *s* format. If a time unit is not specified, time is measured in seconds.
description	`description_text`	Describes the gateway.
enable	{ *yes* | *no* }	Specifies whether the SYNC gateway should be used.
Listeners_are_900	{ *yes* | *no* }	Specifies whether all MobiLink Listeners are SQL Anywhere 9.0.0 clients. The default value is *no*. For SQL Anywhere 9.0.1 clients or later, leave this value at *no*.
password	`password`	Specifies the password for the SMTP service. This is required by some services.
sender	`SMTP_address`	Specifies the sender address of SMTP push notifications. The default value is *anonymous*.
server	`IP_address_or_hostname`	Specifies the IP address or host name of the SMTP server used to send messages to a MobiLink Listener. The default value is *mail*.
user	`username`	Specifies the user name of the SMTP service. This is required by some services.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.7.3 SYNC Gateway Properties

SYNC gateway properties allow you to change the behavior of a SYNC gateway. All SYNC gateway properties are optional.

Property	Value	Description
confirm_action	{ *yes* | *no* }	Specifies whether confirmation is sent on delivery through this gateway. The default value is *no*.
confirm_delivery	{ *yes* | *no* }	Specifies whether this gateway confirms delivery. The default value is *no*.
confirm_timeout	`number`{ *s* | *m* | *h* }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: • *s*: Denotes seconds. • *m*: Denotes minutes. • *h*: Denotes hours. The default value is *1m*. Time units can be combined in the `HH` *h* `MM` *m* `SS` *s* format. If a time unit is not specified, time is measured in seconds.
description	`description_text`	Describes the gateway.
enable	{ *yes* | *no* }	Specifies whether the SYNC gateway should be used.
Listeners_are_900	{ *yes* | *no* }	Specifies whether all MobiLink Listeners are SQL Anywhere 9.0.0 clients. The default value is *no*. Leave this value at *no* for SQL Anywhere 9.0.1 clients or later.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.7.4 UDP Gateway Properties

UDP gateway properties allow you to change the behavior of an UDP gateway, such as the IP address and the port number. All UDP gateway properties are optional.

Property	Value	Description
confirm_action	{ *yes* | *no* }	Specifies whether confirmation is sent on delivery through this gateway. The default value is *no*.
confirm_delivery	{ *yes* | *no* }	Specifies whether this gateway confirms delivery. The default value is *yes*.
confirm_timeout	`number`{ *s* | *m* | *h* }	Specifies the amount of time to wait before confirmations timeout. The following is a list of acceptable time units: • *s*: Denotes seconds. • *m*: Denotes minutes. • *h*: Denotes hours. The default value is *1m*. Time units can be combined in the `HH` *h* `MM` *m* `SS` *s* format. If a time unit is not specified, time is measured in seconds.
description	`description_text`	Describes the gateway.
enable	{ *yes* | *no* }	Specifies whether the UDP gateway should be used.
Listeners_are_900	{ *yes* | *no* }	Specifies whether all MobiLink Listeners are SQL Anywhere 9.0.0 clients. The default value is *no*. For SQL Anywhere 9.0.1 clients or later, leave this value at *no*.
Listener_port	`port_number`	Specifies the port that the remote devices uses to send UDP packets. The default value is *5001*.
sender	`IP_address_or_hostname`	Used for multi-homed hosts only. Specifies the IP address or hostname of the sender. The default value is *localhost*.
sender_port	`port_number`	Specifies the port number used to send UDP packets. By default, a free port number is randomly assigned by the operating system.

Related Information

MobiLink Server Settings for Server-initiated Synchronization [page 32]

1.3.8 Carrier Properties

Carrier properties allow you to change the behavior of a wireless carrier configuration, which provides information about mapping automatically tracked phone numbers and network providers to email addresses. All carrier properties are optional, and are only required if you are using an SMTP gateway.

Property	Value	Description
enable	{ *yes* | *no* }	Specifies whether the carrier should be used.
description	`description_text`	Describes the carrier.
network_provider_id	`id_text`	Specifies the network provider ID. To use SMS on Microsoft Windows Mobile Phone edition, set this property to *_generic_*.
sms_email_domain	`domain_name`	Specifies the domain name of the carrier.
sms_email_user_prefix	`prefix_name`	Specifies the prefix used in email addresses.

1.4 MobiLink Listener Utility for Windows (dblsn)

The MobiLink Listener utility allows Windows to receive push notifications and initiate actions.

Syntax

```
dblsn [ options ] -l message-handler [ -l message-handler... ]
```

```
message-handler :
[ polling-option;... ] [ filter;... ]action; [ option;... ]
```

```
polling-option :
  [ poll_connect = string ]
  [ poll_notifier = string ]
  [ poll_key = string ]
  [ poll_every = number ]
```

```
option :
  [ continue  = yes ]
  [ confirm_action  = yes ]
  [ confirm_delivery  = no ]
  [ maydial  =  no ]
```

```
filter :
  [ subject = string ]
  [ content = string ]
  [ message = string | message_start  =  string ]
  [ sender = string ]
```

```
action :
  action = command[;altaction = command ]
```

```
command :
  START program [ program-arguments ]
  | RUN program [ program-arguments ]
  | POST window-message TO { window-class-name | window-title }
  | tcpip-socket-action
  | DBLSN FULL SHUTDOWN
```

```
tcpip-socket-action :
  SOCKET port=app-port
  [ ;host=app-host ]
  [ ;sendText=text1 ]
  [ ;recvText= text2 [ ;timeout=num-sec ] ]
window-message : string | message-id
```

Remarks

The -l message-handler for dblsn is a semicolon separated name-equal-value pair. String values in the name-equal-value pairs must be surrounded by single quotes; otherwise, dblsn fails to start up if the string value contains a semicolon.

In this section:

1.4.1 Secure Listener Deployment

The Listener receives external notifications and can invoke an application while passing in information from the notifications. External notifications could, in theory, be used to inject harmful data causing undesirable results. Care must be taken to secure the Listener deployment.

It is recommended that you implement the following recommendations to secure the Listener:

- Use the dblsn -x option and specify a TLS-based protocol, HTTPS for example, to verify the server (the notifier) and secure the network communications.
- Do not use a UDP listener because it is not secure. UDP is disabled by default.
- All actions configured via the dblsn -l option must either reject invalid input, or be guaranteed to have no harmful effects when arbitrary input is received.
- Do not invoke a very powerful or a very general application, for example `cmd.com`, in an action specification via the dblsn -l option. Deploy and configure a very specific application that does exactly what you need, and no more, and rejects invalid input.
- Use message filters to limit the invocation of actions.
- Avoid using action variables to specify functionality.

Related Information

Message Filters [page 20]
-x dblsn Option [page 81]
MobiLink Listener Keywords for Microsoft Windows [page 82]
-l dblsn Option [page 71]

1.4.2 Listening Libraries for Microsoft Windows

The MobiLink Listener comes with a UDP listening library, `lsn_udp`*17*`.dll`, which is loaded by default.

You can specify the listening library explicitly with the dblsn -d option, and specify library options with the dblsn -a option.

UDP (`lsn_udp`*17*`.dll`)

The following is a list of options supported by the UDP listening library:

Option	Description
Port= `port-number`	This option specifies the port number to listen to. The default port is *5001*.

Option	Description
Timeout= `seconds`	This option specifies the maximum blocking time of a read operation on the UDP listening port. This value must be smaller than the polling interval of the UDP listening thread. The default is *0*.
ShowSenderPort	This option reveals the sender port number in all occurrences of the $sender action variable. By default, the port number is hidden. When this option is specified, the port number is appended at the end of the sender address with the *:* `port-number` syntax.
HideWSAErrorBox	Suppresses the error window showing errors on socket operations.

Related Information

-d dblsn Option [page 68]
-a dblsn Option [page 67]

1.4.3 MobiLink Listener Options for Microsoft Windows

The following options can be used to configure the MobiLink Listener:

Option	Description
@{ `variable` | `filename` }	Applies MobiLink Listener options from the specified environment variable or text file.
-a `value`	Specifies a single library option for a listening library.
-d `filename`	Specifies a listening library.
-e `device-name`	Specifies the device name.
-f `string`	Specifies extra information about the device.
-gi `seconds`	Specifies the IP tracker polling interval.
-i `seconds`	Specifies the polling interval for SMTP connections.
-l " `keyword=value;...` "	Defines and creates a message handler.
-lu	Enables UDP listening.
-m	Turns on message logging.
-ni	Disables IP tracking.
-o `prefix`	Logs output to a file.
-os `bytes`	Specifies the maximum size of the message log file.
-ot `filename`	Truncates a file, then logs output to that file.

Option	Description
-pc { + | - }	Enables or disables persistent connections.
-q	Runs the MobiLink Listener in quiet mode.
-qi	Prevents the dblsn icon and messages window from appearing.
-r `filename`	Identifies a remote database involved in the responding action of a message filter.
-sv `script-version`	Specifies the script version used for authentication.
-t {+ | - } `name`	Registers or unregisters the remote ID for a remote database.
-ts `session-name`(`session-option`=`option-value`[;...])	Sets up a MobiLink Listener tracing session.
-u `username`	Specifies a MobiLink user name.
-v { *0* | *1* | *2* | *3* }	Specifies the verbosity level for the message log.
-w `password`	Specifies a MobiLink password.
-x { *http* | *https* | *tcpip* } [*(* `protocol-option` = `value`;...*)*]	Specifies the network protocol, and the MobiLink server protocol options.
-y `newpassword`	Specifies a new MobiLink password.

In this section:

-ni dblsn Option [page 72]
    Disables IP tracking.

-o dblsn Option [page 73]
    Logs output to a file.

-os dblsn Option [page 73]
    Specifies the maximum size of the message log file.

-ot dblsn Option [page 73]
    Truncates a file, then logs output to that file.

-pc dblsn Option [page 74]
    Enables or disables persistent connections.

-q dblsn Option [page 75]
    Runs the MobiLink Listener in quiet mode.

-qi dblsn Option [page 75]
    Prevents the dblsn icon and messages window from appearing.

-r dblsn Option [page 75]
    Identifies a remote database involved in the responding action of a message filter.

-sv dblsn Option [page 76]
    Specifies the script version used for authentication.

-t dblsn Option [page 76]
    Registers or unregisters the remote ID for a remote database.

-ts dblsn Option [page 77]
    Sets up a MobiLink Listener tracing session.

-u dblsn Option [page 79]
    Specifies a MobiLink user name for the MobiLink Listener.

-v dblsn Option [page 79]
    Specifies the verbosity level for the message log.

-w dblsn Option [page 80]
    Specifies a MobiLink password.

-x dblsn Option [page 81]
    Specifies the network protocol, and the MobiLink server protocol options.

-y dblsn Option [page 81]
    Specifies a new MobiLink password.

1.4.3.1 @data dblsn Option

Applies MobiLink Listener options from the specified environment variable or text file.

Syntax

```
dblsn @data ...
```

Remarks

By default, `dblsn.txt` is the configuration file when the MobiLink Listener is run without parameters.

If a file and an environment variable with the same name exist, the environment variable is used.

To protect information in the configuration file, you can use the File Hiding utility (dbfhide) to encode the contents of the configuration file.

Example

The following example reads command line options from a dblsnoptions environment variable:

```
dblsn @dblsnoptions
```

The following example reads command line options from `mydblsn.txt`, assuming there is no identically named environment variable:

```
dblsn @mydblsn.txt
```

Related Information

Configuration Files
File Hiding Utility (dbfhide)

1.4.3.2 -a dblsn Option

Specifies a single library option for a listening library.

Syntax

```
dblsn -a value ...
```

Remarks

By default, the MobiLink Listener uses `lsn_udp`*17*`.dll`.

The -a option is used with the -d option that specifies the listening library. When both -d and -a are specified, the -d option must be specified first.

Use the -a option multiple times to set additional library options.

Use `-a ?` with -d to view all available library options.

Example

port

The following example specifies the port option and declares the ShowSenderPort option in the `lsn_udp17.dll` listening library:

```
dblsn -d lsn_udp17.dll -a port=1234 -a ShowSenderPort
```

The following example displays all available library options in the default library:

```
dblsn -d lsn_udp17 -a ?
```

Related Information

Listening Libraries for Microsoft Windows [page 63]
-d dblsn Option [page 68]

1.4.3.3 -d dblsn Option

Specifies a listening library.

Syntax

```
dblsn -d filename ...
```

Remarks

By default, the MobiLink Listener uses the `lsn_udp17.dll` listening library.

Use the -d option to explicitly load the listener library.

Example

The following example specifies the `lsn_udp17.dll` listening library:

```
dblsn -d lsn_udp17.dll
```

The use of the -d option is not necessary since this listening library is loaded by default.

Related Information

Listening Libraries for Microsoft Windows [page 63]

1.4.3.4 -e dblsn Option

Specifies the device name.

Syntax

```
dblsn -e device-name ...
```

Remarks

By default, the device name is automatically generated by the operating system.

When connecting to the MobiLink server, make sure that all device names are unique.

The device name should be restricted to the following:

- alphanumeric characters
- period
- colon
- minus sign
- plus sign
- underscore

The device name can be found in the MobiLink Listener window.

1.4.3.5 -f dblsn Option

Specifies extra information about the device.

Syntax

```
dblsn -f string ...
```

Remarks

By default, this information is the version number of the operating system running on the device.

1.4.3.6 -gi dblsn Option

Specifies the IP tracker polling interval.

Syntax

```
dblsn -gi number ...
```

Remarks

By default, the IP tracker polls every 60 seconds.

1.4.3.7 -i dblsn Option

Specifies the polling interval for connections.

Syntax

```
dblsn -i number ...
```

Remarks

The -i option specifies the frequency at which the MobiLink Listener checks for messages.

For UDP connections, this option should not be specified.

The -i option is used with the -d option that specifies the listening library. When both -d and -i are specified, the -d option must be specified first.

Related Information

-d dblsn Option [page 68]

1.4.3.8 -l dblsn Option

Defines and creates a message handler.

Syntax

```
dblsn -l "keyword=value;..." ...
```

Remarks

Use the -l option multiple times to define additional message handlers for push notifications. Message handlers are processed in the order they are specified.

Related Information

Message Handlers [page 19]
MobiLink Listener Keywords for Microsoft Windows [page 82]

1.4.3.9 -lu dblsn Option

Enables UDP listening. UDP listeners are unsecured and should be avoided.

Syntax

```
dblsn -lu ...
```

Remarks

UDP listening is disabled by default.

Related Information

Secure Listener Deployment [page 63]
MobiLink Listener Keywords for Microsoft Windows [page 82]
-x dblsn Option [page 81]

1.4.3.10 -m dblsn Option

Turns on message logging.

Syntax

```
dblsn -m  ...
```

Remarks

By default, message logging is turned off.

1.4.3.11 -ni dblsn Option

Disables IP tracking.

Syntax

```
dblsn -ni  ...
```

Remarks

By default, IP tracking is enabled.

This option does not stop delivery confirmation.

The -ni option disables UDP address tracking when used with the -x option. This feature is useful if you want device tracking to exclude UDP address updates.

Related Information

-x dblsn Option [page 81]

1.4.3.12 -o dblsn Option

Logs output to a file.

Syntax

```
dblsn -o filename ...
```

Remarks

By default, output is logged to the MobiLink Listener window.

Related Information

-ot dblsn Option [page 73]

1.4.3.13 -os dblsn Option

Specifies the maximum size of the message log file.

Syntax

```
dblsn -os bytes ...
```

Remarks

By default, there is no maximum size limit. The minimum size limit is 10000.

1.4.3.14 -ot dblsn Option

Truncates a file, then logs output to that file.

Syntax

```
dblsn -ot filename ...
```

Remarks

The file contents are deleted before output is logged.

Related Information

-o dblsn Option [page 73]

1.4.3.15 -pc dblsn Option

Enables or disables persistent connections.

Syntax

```
dblsn -pc { + | - } ...
```

Remarks

By default, persistent connections are enabled. The - flag disables persistent connections; the + flag enables them.

Disabling persistent connections prevents the MobiLink Listener from receiving push notifications, but allows short-lived persistent connections for device tracking and confirmation.

If a persistent connection is broken, the MobiLink Listener continuously attempts to reconnect.

Example

The following example disables persistent connections:

```
dblsn -pc-
```

1.4.3.16 -q dblsn Option

Runs the MobiLink Listener in quiet mode.

Syntax

```
dblsn -q ...
```

Remarks

The -q option minimizes the MobiLink Listener window. By default, the MobiLink Listener window is displayed.

1.4.3.17 -qi dblsn Option

Prevents the dblsn icon and messages window from appearing.

Syntax

```
dblsn -qi ...
```

Remarks

This option leaves no visual indication that dblsn is running, other than possible startup error windows. You can use the -o log files to diagnose errors.

Related Information

-o dblsn Option [page 73]

1.4.3.18 -r dblsn Option

Identifies a remote database involved in the responding action of a message filter.

Syntax

```
dblsn -r filename ...
```

Remarks

The `filename` must contain the full path of an RID file. This file is automatically created by dbmlsync after the first synchronization. It uses the same location and name as the database file. For UltraLite databases, `filename` should be the same as the database name.

When applying the -r option, the $remote_id action variable can be used in message handlers to reference the remote ID in the RID file. By default, Remote IDs are GUIDs.

Use the -r option multiple times to identify multiple databases.

Related Information

Message Filters [page 20]
MobiLink Listener Action Commands for Microsoft Windows [page 84]

1.4.3.19 -sv dblsn Option

Specifies the script version used for authentication.

Syntax

```
dblsn -sv script-version ...
```

Remarks

By default, the MobiLink Listener uses the ml_global server script version if it is defined.

1.4.3.20 -t dblsn Option

Registers or unregisters the remote ID for a remote database.

Syntax

```
dblsn -t { + | - } name ...
```

Remarks

The + flag registers a remote ID; the - flag unregisters ID.

Registering allows the MobiLink Listener to address push notifications by referencing that remote ID.

When device tracking information is uploaded successfully, registered IDs are retained in the ml_listening system table on the server. You only need to register the ID once.

Use the -t option multiple times to register or unregister multiple IDs. Registering multiple IDs is useful for addressing push notifications to multiple remote databases.

Related Information

MobiLink Listener Action Commands for Microsoft Windows [page 84]

1.4.3.21 -ts dblsn Option

Sets up a MobiLink Listener tracing session.

Syntax

```
dblsn -ts session-name(session-option=option-value[;...])
```

The session name must be **`logging`**.

Session option	Option value
events	Comma separated list of system trace events. The supported events are Info, Warning, and Error.
targets	`target-type` (`target-option` = `value` *[;...]*) where `target-type` can only be *file*.

The target options are specified as name-value pairs. The target file can have the following options:

Target option name	Expected value	Description
filename_prefix	String	An ETD file name prefix with or without a path. All ETD files have the extension `.etd`. This parameter is required.
max_size	Integer	The maximum size of the file in bytes. The default is 0, which means there is no limit on the file size and it grows as long as disk space is available. Once the specified size is reached, a new file is started.

Target option name	Expected value	Description
num_files	Integer	The number of files where event tracing information is written, and it is used only if max_size is set. If all the files reach the maximum specified size, the MobiLink Listener starts overwriting the oldest file.
flush_on_write	yes, true, no, false	A value that controls whether disk buffers are flushed for each event that is logged. The values yes, true, no, and false are accepted. The default is false. When this parameter is turned on, the performance of the MobiLink Listener may be reduced if many trace events are being logged.
compressed	yes, true, no, false	A value that controls compression of the ETD file to conserve disk space. The default is false.

Remarks

All information specified after the `-ts logging` portion of the option must be specified without any spaces.

Example

Following is an example of the -ts option:

```
-ts
logging(events=Info,warning,Error;targets=file(filename_prefix=mls_etd;max_size=1
0000000;num_files=10;flush_on_write=true))
```

Related Information

Event Trace Data (ETD) File Management Utility (dbmanageetd)

1.4.3.22 -u dblsn Option

Specifies a MobiLink user name for the MobiLink Listener.

Syntax

```
dblsn -u username ...
```

Remarks

By default, the user name is `device-name`-*dblsn*, where `device-name` is the name of your device. You can specify a device name using the -e option.

The MobiLink Listener uses the user name to connect to the MobiLink server for device tracking, confirmations, and persistent connections.

The user name must be a unique MobiLink user name registered with the MobiLink server. The name must exist in the ml_user system table on the consolidated database.

Related Information

MobiLink User Creation and Registration
-e dblsn Option [page 69]
-w dblsn Option [page 80]

1.4.3.23 -v dblsn Option

Specifies the verbosity level for the message log.

Syntax

```
dblsn -v { 0 | 1 | 2 | 3 } ...
```

Remarks

By default, the verbosity level is set to 0.

The following table summarizes the possible verbosity level values.

Verbosity level	Description
0	Verbosity is turned off.
1	Displays listening library messages, basic action tracing steps, and command line options.
2	Displays level 1 verbosity messages, and detailed action tracing steps.
3	Displays level 2 verbosity messages, polling states, and listening states.

You must use the -m option to output push notifications to the message log.

Related Information

How to Log Database Server Messages to a File
-m dblsn Option [page 72]

1.4.3.24 -w dblsn Option

Specifies a MobiLink password.

Syntax

```
dblsn -w password ...
```

Remarks

Passwords must be registered with the MobiLink server under the associated MobiLink user name.

The MobiLink Listener uses the password to connect to the MobiLink server for device tracking, confirmations, and persistent connections.

Related Information

-u dblsn Option [page 79]

1.4.3.25 -x dblsn Option

Specifies the network protocol, and the MobiLink server protocol options.

Syntax

```
dblsn -x { http | https | tcpip } [ (protocol-option=value;...) ] ...
```

Remarks

A connection to the MobiLink server is required so the MobiLink Listener can send device tracking information and delivery confirmation to the consolidated database. Use the *host* protocol option to specify the location of the MobiLink server.

The -x option allows the device to update the consolidated database if the server address changes.

Related Information

MobiLink Client Network Protocol Options
host MobiLink Client Network Protocol Option

1.4.3.26 -y dblsn Option

Specifies a new MobiLink password.

Syntax

```
dblsn -y newpassword ...
```

Remarks

The -y option is not applicable if your authentication system does not allow remote devices to change their passwords.

1.4.4 MobiLink Listener Keywords for Microsoft Windows

The following keywords can be used to configure message handlers created with the dblsn -l option.

Filter Keywords

Use the following keywords to filter messages in a push notification:

Keyword syntax	Description
subject= `subject-string`	Filters a message if the subject is textually equivalent to `subject-string`.
content= `content-string`	Filters a message if the content is textually equivalent to `content-string`.
message= `message-string`	Filters a message if the entire message is textually equivalent to `message-string`.
message_start= `message-string`	Filters a message if it begins with `message-string`.
sender= `sender-string`	Filters a message if it is sent by `sender-string`.

Action Keywords

Use the following keywords to initiate an action when a filter condition is met:

Keyword syntax	Description
action= `command`	Specifies an action command.
altaction= `command`	Specifies an alternative action command that is initiated when the action command fails.

Polling Options

Use the following options to configure a lightweight poller:

Keyword syntax	Description
poll_connect={ *http* | *https* | *tcpip* } [*(* `protocol-option` = `value`;*...)*]	Specifies the lightweight network protocol options required to connect to the server. The default value is inherited from the dblsn -x option.

Keyword syntax	Description
poll_notifier= `Notifier-string`	Specifies the name of the Notifier that handles push requests. Required.
poll_key= `key-string`	Specifies the name of the MobiLink Listener to identify itself to the Notifier. This value must be unique. Required.
poll_every= `seconds-number`	Specifies how often the MobiLink Listener should poll the server. The interval is measured in seconds. The default value is auto-retrieved from the MobiLink server.

Options

The following can be used to configure message handler behavior:

Keyword syntax	Description
continue=[*yes* | *no*]	Specifies whether the MobiLink Listener should continue listening after finding the first match. The default value is *no*. A *yes* value is useful when specifying multiple filters, where one message initiates multiple actions.
confirm_action=[*yes* | *no*]	Specifies whether the filter should confirm the action. The default value is *yes*.
confirm_delivery=[*yes* | *no*]	Specifies whether the filter should confirm qualified message delivery. The default value is *yes*, so delivery confirmation is sent after the first filter accepts the message. Delivery can only be confirmed if the message requires confirmation, and if the filter accepts the message. A message requires confirmation if the specified gateway has its confirm_delivery keyword value set to *yes*. A *no* value can be used when multiple filters accept the same message to give you finer control over which filter should confirm the delivery. Use the confirmation_handler event to handle delivery confirmation information uploaded by MobiLink Listeners.
maydial=[*yes* | *no*]	Specifies whether the action has permission to dial the modem. The default value is *yes*. A *no* value causes the MobiLink Listener to release the modem before the action.

Related Information

Message Handlers [page 19]
-x dblsn Option [page 81]
confirmation_handler Event [page 50]
MobiLink Listener Action Commands for Microsoft Windows [page 84]

1.4.5 MobiLink Listener Action Commands for Microsoft Windows

An action is specified when you configure a new message handler. When a filter condition is met, an action is initiated. If the action fails, an alternative action is initiated. Actions are defined using the *action* keyword; alternative actions are defined using the *altaction* keyword.

Following is a list of action commands:

Command	Description
START `program` `arglist`	Initiates a program while the MobiLink Listener runs in the background.
RUN `program` `arglist`	Pauses the MobiLink Listener to run a program.
POST `windowmessage` | `id` to `windowclass` | `windowtitle`	Posts a window message to a window class.
SOCKET port= `windowname`[;*host=* `hostname`][;*sendText=* `text`][;*recvText=* `text`[;*timeout=* `seconds`]]	Sends a message to an application using a TCP/IP connection.
DBLSN FULL SHUTDOWN	Forces the MobiLink Listener to shutdown.

You can only specify one action per *action* or *altaction* keyword. If you want an action to perform multiple tasks, create a batch file that contains multiple commands and run the file using the *RUN* action command.

In this section:

Related Information

Initiation of Actions [page 21]
MobiLink Listener Keywords for Microsoft Windows [page 82]

1.4.5.1 START Action Command

Initiates a program while the MobiLink Listener runs in the background.

Syntax

```
action='START program arglist'
```

Remarks

When you start a program, the MobiLink Listener continues listening for more push notifications.

The MobiLink Listener does not wait for the program to finish, so it can only determine if an action command has failed, or if it cannot start the specified program.

Example

The following example starts dbmlsync with some command line options, parts of which are obtained from messages using the $content action variable:

```
dblsn -l "action='start dbmlsync.exe @dbmlsync.txt -n
  $content -wc dbmlsync_$content -e sch=INFINITE';"
```

1.4.5.2 RUN Action Command

Pauses the MobiLink Listener to run a program.

Syntax

```
action='RUN program arglist'
```

Remarks

The MobiLink Listener waits for the program to finish, then resumes listening.

When running a program, the MobiLink Listener determines if the program has failed, if the MobiLink Listener cannot start the program or if the program returns a non-zero return code.

Example

The following example runs dbmlsync with some command line options, parts of which are obtained from the message using the $content action variable:

```
dblsn -l "action='run dbmlsync.exe @dbmlsync.txt -n $content';"
```

1.4.5.3 POST Action Command

Posts a window message to a window class.

Syntax

```
action='POST windowmessage | id to windowclass | windowtitle'
```

Remarks

The POST command can be used to signal applications that use window messages.

You can identify the window message by message contents, or by the window message ID, if one exists.

You can identify the window class by its class name or the window title. If you identify by name, you can use the -wc dbmlsync option to specify the window class name. If you identify the window class by the window title, you can only reference it by the top level window.

If your window message or window class name contains non-alphanumeric characters, such as spaces or punctuation marks, encapsulate the text in single quotes ('). The escape character is also a single quote, so if your window message or window class name contains single quotes, reference the quote using two single quotes ('').

The following are valid for POST:

Post by decimal id

For example, `post 999 to <wc|wt>`

Post by hex id

For example, `post 0x3E7 to <wc|wt>`

Post by registered message name

For example, `post myRegisteredMsgName to <wc|wt>`

Example

To demonstrate the use of windows and messages containing single quotes, the following example posts mike's_message window message to the mike's_class window class:

```
dblsn -l "action='post mike''s_message to mike''s_class';"
```

The following example posts a window message, dbas_synchronize, to a dbmlsync instance registered with the dbmlsync_FullSync class name:

```
dblsn -l "action='post dbas_synchronize to dbmlsync_FullSync';"
```

Related Information

-wc dbmlsync Option

1.4.5.4 SOCKET Action Command

Sends a message to an application using a TCP/IP connection.

Syntax

```
action='SOCKET port=windowname[;host=hostname][;sendText=text]
[;recvText=text[;timeout=seconds]]'
```

Remarks

The SOCKET command is used for passing dynamic information to a running application, and for integrating messages into Java and Microsoft Visual Basic applications. Both languages do not support custom window messaging, and Microsoft eMbedded Visual Basic does not support command line parameters.

To connect to a socket, you must specify the port and the host. Use sendText to input your message.

Use recvText to display a message when confirming that sendText is successfully received by the application. When using recvText, you can specify a timeout limit. The action fails if the MobiLink Listener cannot connect, send acknowledgements, or receive acknowledgements during the timeout limit.

Example

The following example forwards the string in $sender=$message to a local application that is listening on port 12345. The MobiLink Listener expects the application to send "beeperAck" as an acknowledgement within 5 seconds.

```
dblsn -l "action='socket port=12345;
   sendText=$sender=$message;
   recvText=beeperAck;
   timeout=5'"
```

1.4.5.5 DBLSN FULL SHUTDOWN Action Command

Forces the MobiLink Listener to shutdown.

Syntax

action='DBLSN FULL SHUTDOWN'

Remarks

After shutdown, the MobiLink Listener stops handling push notifications and stops synchronizing device tracking information. You must restart the MobiLink Listener to continue with server-initiated synchronization.

1.4.6 MobiLink Listener Action Variables for Microsoft Windows

The following action variables can be used in an action or a filter. The value is substituted into the action variable before initiating the message handler.

Action variables start with a dollar sign ($). The escape character is also a dollar sign, so use two dollar signs ($$) to specify a single dollar sign as plain text.

Variable	Description
$subject	The subject of a message.
$content	The content of a message.
$message	An entire message, including subject, content, and sender.
$message_start	The beginning portion of a message, as specified by the message_start filter keyword. This variable is only available if you have specified the message_start filter keyword.

Variable	Description
$message_end	The portion of a message, excluded from the message_start filter keyword. This variable is only available if you have specified the message_start filter keyword.
$ml_connect	The MobiLink network protocol options as specified by the dblsn -x option. The default is an empty string.
$ml_user	The MobiLink user name as specified by the dblsn -u option. The default name is `device-name-`*dblsn*.
$ml_password	The MobiLink password as specified by dblsn -w option, or the new MobiLink password if -y is used.
$priority	The meaning of this variable is carrier library-dependent.
$request_id	The request ID that was specified in a push request.
$remote_id	The remote ID. This variable can only be used when the dblsn -r option is specified.
$sender	The sender of a message.
$type	The meaning of this variable is carrier library-dependent.
$year	The meaning of this variable is carrier library-dependent.
$month	The meaning of this variable is carrier library-dependent. Values can be from 1-12.
$day	The meaning of this variable is carrier library-dependent. Values can be from 1-31.
$hour	The meaning of this variable is carrier library-dependent. Values can be from 0-23.
$minute	The meaning of this variable is carrier library-dependent. Values can be from 0-59.
$second	The meaning of this variable is carrier library-dependent. Values can be from 0-59.
$best_adapter_mac	The MAC address of the best NIC for reaching the MobiLink server that is specified by the dblsn -x option. If the best route does not go through a NIC, the value is an empty string.
$best_adapter_name	The adapter name of the best NIC for reaching the MobiLink server that is specified by the dblsn -x option. If the best route does not go through a NIC, the value is an empty string.
$best_ip	The IP address of the best IP interface for reaching the MobiLink server that is specified by the dblsn -x option. If that server is unreachable, the value is 0.0.0.0.
$best_network_name	The RAS or dial-up profile name of the best profile for reaching the MobiLink server that is specified by the dblsn -x option. If the best route does not go through a RAS/dial-up connection, the value is an empty string.
$adapters	A list of active network adapter names, each separated by a vertical bar (|).

Variable	Description
$network_names	A list of connected RAS entry names, each separated by a vertical bar (|). RAS entry names are sometimes referred to as dial-up entry names or Dial-Up Network (DUN).
$poll_connect	The MobiLink network protocol options as specified by the poll_connect polling keyword. The default is an empty string.
$poll_notifier	The name of the Notifier as specified by the poll_notifier polling keyword.
$poll_key	The poll key as specified by the poll_key polling keyword.
$poll_every	The polling frequency as specified by the poll_every polling keyword.

Example

The following examples uses the $message_end action variable to determine which publication to synchronize:

```
dblsn -l "message_start=start-of-message;action='run dbmlsync.exe -c ... -n
$message_end'"
```

Related Information

Push Requests [page 9]
Advanced: Remote ID as a Filter [page 24]
-x dblsn Option [page 81]
-l dblsn Option [page 71]
MobiLink Listener Keywords for Microsoft Windows [page 82]
MobiLink Listener Action Commands for Microsoft Windows [page 84]

1.5 Lightweight Polling API

The lightweight polling API is a programming interface that you can integrate into your device application. It contains the methods needed to poll a server.

Required Files

All directories are relative to *%SQLANY17%*. The following is a list of files required to compile the lightweight polling API:

File name or location	Description
`Bin32\mllplib`*17*`.dll`	Lightweight polling API runtime dynamic library.
`SDK\Lib\x86\mllplib`*17*`.lib` and `SDK\Lib\x64\mllplib`*17*`.lib`	Lightweight polling API runtime import library.
`SDK\Include\mllplib.h`	Lightweight polling API header file.

A SIS_CarDealer_LP_API sample application written in C is supplied in *%SQLANYSAMP17%*`\MobiLink\SIS_CarDealer_LP_API`.

API Members

Method	Description
MLLightPoller class	Represents a lightweight poller object.
MLLPCreatePoller method	Creates an instance of an MLLightPoller.
MLLPDestroyPoller method	Destroys an instance of an MLLightPoller.

In this section:

1.5.1 MLLightPoller Class

Represents a lightweight poller object.

Syntax

```
public class MLLightPoller
```

Members

Name	Description
Poll method	Polls the server, prompting a Notifier to check a cache for push requests.
SetConnectInfo method	Sets up a MobiLink client stream type, and network protocol options.
auth_status enumeration	Specifies possible authentication status codes.
return_code enumeration	Specifies possible return codes.

Example

```
MLLightPoller * poller = MLLCreatePoller();
```

In this section:

1.5.1.1 Poll Method

Polls the server, prompting a Notifier to check a cache for push requests.

Syntax

```
public virtual return_code MLLightPoller::Poll(
    const char * notifier,
    const char * key,
    char * subject = 0,
    size_t * subjectSize = 0,
    char * content = 0,
    size_t * contentSize = 0
)
```

Parameters

notifier

The name of the Notifier.

key

The name of the poll key identifying the MobiLink Listener.

subject

The buffer to receive a message subject. (null-terminated)

subjectSize

IN: The size of the subject buffer.

OUT: The size of the received subject, including a null-terminator, where zero indicates a null subject.

content

The buffer to receive message content. (null-terminated)

contentSize

IN: The size of the content buffer.

OUT: The size of the received content, including a null-terminator, where zero indicates null content.

Returns

One of the codes listed in the return_code enumeration.

Remarks

The MobiLink Listener connects to the Notifier, then disconnects after the Notifier checks its cache for a push notification targeted for the given poll key.

Related Information

return_code Enumeration [page 95]

1.5.1.2 SetConnectInfo Method

Sets up a MobiLink client stream type, and network protocol options.

Syntax

```
public virtual return_code MLLightPoller::SetConnectInfo(
    const char * streamName,
    const char * streamParams
);
```

Parameters

streamName

The network protocol to use. Acceptable values are: tcpip, http, https, or tls.

streamParams

A protocol options string, formatted in a semicolon-delimited list.

Returns

One of the codes listed in the return_code enumeration.

Example

```
poller_ret = poller->SetConnectInfo("http", "host=localhost;port=80;")
```

Related Information

MobiLink Client Network Protocol Options
return_code Enumeration [page 95]

1.5.1.3 auth_status Enumeration

Specifies possible authentication status codes.

Syntax

```
public typedef enum MLLightPoller::auth_status;
```

Members

Member name	Description
AUTH_EXPIRED	Authentication has expired.
AUTH_IN_USE	The user name is already authenticated.
AUTH_INVALID	Authentication is invalid.
AUTH_UNKNOWN	Authentication is unknown.
AUTH_VALID	Authentication is valid.
AUTH_VALID_BUT_EXPIRES_SOON	Authentication is valid but expires soon.

1.5.1.4 return_code Enumeration

Specifies possible return codes.

Syntax

```
public typedef enum MLLightPoller::return_code;
```

Members

Name	Description
AUTH_FAILED	Failed to authenticate the connection to the MobiLink server.
BAD_STREAM_NAME	Unrecognizable stream name.
BAD_STREAM_PARAM	Failed parsing a stream parameter.
COMMUNICATION_ERROR	Failed due to a communication error.
CONNECT_FAILED	Failed to connect to the MobiLink server.
CONTENT_OVERFLOW	The content buffer is too small.
KEY_NOT_FOUND	There is no push notification for the specified key from the specified Notifier.
NYI	For internal use only.
OK	Method ran successfully.
SUBJECT_OVERFLOW	The subject buffer is too small.

1.5.2 MLLPCreatePoller Method

Creates an instance of an MLLightPoller.

Syntax

```
extern MLLightPoller * MLLPCreatePoller()
```

Returns

A new MLLightPoller object.

Example

```
poller = MLLPCreatePoller();
```

Related Information

MLLPDestroyPoller Method [page 97]

1.5.3 MLLPDestroyPoller Method

Destroys an instance of an MLLightPoller.

Syntax

```
extern void MLLPDestroyPoller(
    MLLightPoller * poller
)
```

Parameters

poller

The MLLightPoller to destroy.

Remarks

This method accepts null MLLightPoller objects.

Example

```
MLLPDestroyPoller(poller);
```

Related Information

MLLPCreatePoller Method [page 96]

1.6 Server-initiated Synchronization System Procedures

Server-initiated synchronization system procedures add and delete rows in MobiLink system tables.

> i Note
>
> These system procedures are used for device tracking. If you use remote devices that support automatic device tracking, you do not need to use these system procedures. If you use remote devices that do not support automatic device tracking, you can configure manual device tracking using these system procedures.

In this section:

Related Information

Device Tracking Gateways [page 28]
MobiLink Server System Tables
MobiLink Server System Procedures
Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]

1.6.1 ml_delete_device System Procedure

Delete all information about a remote device when you are manually setting up device tracking.

Parameters

Item	Parameter	Description
1	@device	VARCHAR(255). Device name.

Remarks

This function is useful only if you are manually setting up device tracking.

Example

Delete a device record and all associated records that reference this device record:

```
CALL ml_delete_device('myOldDevice');
```

Related Information

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]

1.6.2 ml_delete_device_address System Procedure

Delete a device address when you are manually setting up device tracking.

Parameters

Item	Parameter	Description
1	@device	VARCHAR(255)

Item	Parameter	Description
2	@medium	VARCHAR(255)

Remarks

This system procedure is useful only if you are manually setting up device tracking.

Example

Delete an address record:

```
CALL ml_delete_device_address('myWindowsMobile', 'ROGERS AT&T');
```

Related Information

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]

1.6.3 ml_delete_listening System Procedure

Delete mappings between a MobiLink user and remote devices when you are manually setting up device tracking.

Parameters

Item	Parameter	Description
1	@name	VARCHAR(128)

Remarks

This system procedure is useful only if you are manually setting up device tracking.

Example

Delete a recipient record:

```
CALL ml_delete_listening('myULDB');
```

Related Information

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]

1.6.4 ml_set_device System Procedure

Add or alter information about remote devices when you are manually setting up device tracking. It adds or updates a row in the ml_device table.

Parameters

Item	Parameter	Description
1	@device	VARCHAR(255). User-defined unique device name.
2	@listener_version	VARCHAR(128). Optional remarks on MobiLink Listener version.
3	@listener_protocol	INTEGER. Use *0* for version 9.0.0 or *2* for post-9.0.0 MobiLink Listeners for Windows devices.
4	@info	VARCHAR(255). Optional device information.
5	@ignore_tracking	VARCHAR(1). Set to *y* to ignore tracking and stop it from overwriting manually entered data.
6	@source	VARCHAR(255). Optional remarks on the source of this record.

Remarks

The system procedures ml_set_device, ml_set_device_address, and ml_set_listening are used to override automatic device tracking by changing information in the MobiLink system tables ml_device, ml_device_address, and ml_listening.

This system procedure is useful only if you are manually setting up device tracking.

Example

For each device, add a device record:

```
CALL ml_set_device(
    'myWindowsMobile',
    'MobiLink Listeners for myWindowsMobile - 9.0.1',
    '1',
    'not used',
    'y',
    'manually entered by administrator'
);
```

Related Information

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]

ml_set_device_address System Procedure [page 102]

ml_set_listening System Procedure [page 104]

1.6.5 ml_set_device_address System Procedure

Add or alter information about remote device addresses when you are manually setting up device tracking. It adds or updates a row in the ml_device_address table.

Parameters

Item	Parameter	Description
1	@device	VARCHAR(255). Existing device name.

Item	Parameter	Description
2	@medium	VARCHAR(255). Network provider ID (must match a carrier's network_provider_id property).
3	@address	VARCHAR(255). Phone number of an SMS-capable device.
4	@active	VARCHAR(1). Set to *y* to activate this record to be used for sending push notifications.
5	@ignore_tracking	VARCHAR(1). Set to *y* to ignore tracking and stop it from overwriting manually entered data.
6	@source	VARCHAR(255). Optional remarks on the source of this record.

Remarks

The system procedures ml_set_device, ml_set_device_address, and ml_set_listening are used to override automatic device tracking by changing information in the MobiLink system tables ml_device, ml_device_address, and ml_listening.

This system procedure is useful only if you are manually setting up device tracking.

Example

For each device, add an address record for a device:

```
CALL ml_set_device_address(
    'myWindowsMobile',
    'ROGERS AT&T',
    '3211234567',
    'y',
    'y',
    'manually entered by administrator'
);
```

Related Information

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]
ml_set_device System Procedure [page 101]
ml_set_listening System Procedure [page 104]

1.6.6 ml_set_listening System Procedure

Add or alter mappings between MobiLink users and remote devices when you are manually setting up device tracking. It adds or updates a row in the ml_listening table.

Parameters

Item	Parameter	Description
1	@name	VARCHAR(128). MobiLink user name.
2	@device	VARCHAR(255). Existing device name.
3	@listening	VARCHAR(1). Set to *y* to activate this record to be used for DeviceTracker addressing.
4	@ignore_tracking	VARCHAR(1). Set to *y* to ignore tracking and stop it from overwriting manually entered data.
5	@source	VARCHAR(255). Optional remarks on the source of this record.

Remarks

The system procedures ml_set_device, ml_set_device_address, and ml_set_listening are used to override automatic device tracking by changing information in the MobiLink system tables ml_device, ml_device_address, and ml_listening.

This system procedure is useful only if you are manually setting up device tracking.

Example

For each remote database, add a recipient record for a device. This maps the device to the MobiLink user name.

```
CALL ml_set_listening(
     'myULDB',
     'myWindowsMobile',
     'y',
     'y',
     'manually entered by administrator'
);
```

Related Information

Setting up Device Tracking for 9.0.0 MobiLink Listeners [page 29]
ml_set_device System Procedure [page 101]
ml_set_device_address System Procedure [page 102]

1.6.7 ml_set_sis_sync_state System Procedure

Record the MobiLink synchronization state into the ml_sis_sync_state system table.

Parameters

Item	Parameter	Description
1	@remote_id	VARCHAR(128)
2	@subscription_id	VARCHAR(128)
3	@publication_name	VARCHAR(128)
4	@user_name	VARCHAR(128)
5	@last_upload	TIMESTAMP
6	@last_download	TIMESTAMP

Remarks

Call the ml_set_sis_sync_state system procedure in the publication_nonblocking_download_ack event to allow users to create a request_cursor event that references the ml_sis_sync_state table.

Example

Specify a publication_nonblocking_download_ack event script with the following script to record the synchronization state:

```
CALL ml_set_sis_sync_state(
    {ml s.remote_id},
    NULL,
    {ml s.publication_name},
    {ml s.username},
    NULL,
    {ml s.last_publication_download}
);
```

Related Information

publication_nonblocking_download_ack Connection Event

1.7 Advanced: Message Syntax

The following message syntax applies to lightweight polling (default), UDP gateways, and SYNC gateways.

```
message = [subject]content
```

Messages sent using the SMTP gateway have one of the following syntax structures:

- `message = sender[subject]content`
- `message = sender(subject)content`
- `message = sender{subject}content`
- `message = sender<subject>content`
- `message = sender'subject'content`
- `message = sender"subject"content`

The proper message syntax and `sender` email address syntax are dependent on your wireless carrier. To determine the message syntax, run the MobiLink Listener with message logging enabled, and with the verbosity level set to 2 using the dblsn -m and -v options. The message log contains the proper syntax when you initially run the MobiLink Listener.

When using a device tracking gateway, the message syntax depends on the subordinate gateway used to send the message. If you are using an SMTP subordinate gateway, the syntax is dependent on your public wireless carrier.

Remarks

Braces, chevrons, double quotation marks, parentheses, single quotation marks, and square brackets are reserved for internal use, and should not be used within **subject**.

In this section:

A SQL Anywhere consolidated database can use the sa_send_udp system procedure to send push notifications to a device through a UDP gateway. This method is an alternative to sending push notifications with Notifiers.

Related Information

Gateways and Carriers [page 27]
MobiLink Listener Keywords for Microsoft Windows [page 82]
-m dblsn Option [page 72]
-v dblsn Option [page 79]
How to Work with Push Requests [page 12]

1.7.1 Advanced: Sending a Push Notification Using the sa_send_udp System Procedure

A SQL Anywhere consolidated database can use the sa_send_udp system procedure to send push notifications to a device through a UDP gateway. This method is an alternative to sending push notifications with Notifiers.

Prerequisites

- A MobiLink Listener is set up on a device and listening for push notifications
- Microsoft Internet Explorer is installed on the device
- The following command was run on the device:

```
dblsn -l "message=RunBrowser;action='START iexplore.exe http://www.sap.com';"
```

- A SQL Anywhere consolidated database is running on the MobiLink server

Context

By appending a `1` to the end of your original message, and then using that message in the msg argument of an sa_send_udp system procedure, you send the original message to a MobiLink Listener.

Procedure

1. Run Interactive SQL and connect to your consolidated database using a command like the one below, replacing `consdb-source-name` with the ODBC name of your consolidated database.

   ```
   dbisql -c "dsn=consdb-source-name"
   ```
2. Execute the following statement to send the push notification:

   ```
   CALL sa_send_udp('device-ip-address', 5001, 'RunBrowser1')
   ```

   The first argument ensures that the push notification is sent to the correct device. Replace `device-ip-address` with the IP address of the device. If you are running the MobiLink Listener on the same computer as the MobiLink server, use **`localhost`**.

   The second argument is the port number. By default, MobiLink Listeners use port 5001 to listen for UDP.

   The third argument is the message to send with a **`1`** appended at the end. By appending a 1, which is a reserved server-initiated synchronization protocol, the *RunBrowser* message is sent to the device using a UDP gateway.

Results

When the system call is executed, the *RunBrowser* message is sent to the device, causing the device to run Microsoft Internet Explorer and load the SAP home page.

Related Information

sa_send_udp System Function

1.8 Server-initiated Synchronization Tutorials

Use the following tutorials to gain a better understanding of how to use server-initiated synchronization.

1. Tutorial: Configuring Server-initiated Synchronization Using Lightweight Polling [page 109]
   This tutorial demonstrates how to configure a SQL Anywhere consolidated and remote database for server-initiated synchronization. It is based on the sample code located in *%SQLANYSAMP17%*`\MobiLink\SIS_CarDealer_LP_DBLSN`.
2. Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]
   This tutorial demonstrates how to configure a SQL Anywhere consolidated and remote database for server-initiated synchronization. It is based on the sample code located in *%SQLANYSAMP17%*`\MobiLink\SIS_CarDealer`.

1.8.1 Tutorial: Configuring Server-initiated Synchronization Using Lightweight Polling

This tutorial demonstrates how to configure a SQL Anywhere consolidated and remote database for server-initiated synchronization. It is based on the sample code located in *%SQLANYSAMP17%*`\MobiLink\SIS_CarDealer_LP_DBLSN`.

Prerequisites

You require a basic knowledge of MobiLink event scripts.

The following software is required:

- SQL Anywhere 17

You must have the following roles and privileges on the consolidated database:

- SYS_AUTH_RESOURCE_ROLE compatibility role
- MONITOR system privilege

You must have the following roles and privileges on the remote database:

- SYS_REPLICATION_ADMIN_ROLE system role
- SYS_RUN_REPLICATION_ROLE system role

Context

Sample implementations of server-initiated synchronization are located in *%SQLANYSAMP17%*`\MobiLink`. All server-initiated synchronization sample directory names begin with the SIS_ prefix.

> i Note
>
> You can use SQL Central to administer remote databases, and then use server-initiated remote tasks (SIRT) as an alternative to server-initiated synchronization using lightweight polling.

In this tutorial, you:

- Set up a SQL Anywhere consolidated database for server-initiated synchronization.
- Configure server-side properties.
- Issue push requests to prompt a server-initiated synchronization.

In this section:

Lesson 1: Setting up the Consolidated Database [page 111]
In this lesson, you create a consolidated database named **`SIS_CarDealer_LP_DBLSN_CONDB`** with the scripts required for synchronization using the dbinit utility. You then use the SQL Anywhere 17 driver to define an ODBC data source for the **`SIS_CarDealer_LP_DBLSN_CONDB`** database.

Lesson 2: Generating a Database Schema [page 112]
: In this lesson, you generate a database schema, which includes a Dealer table, a non_sync_request table, a download_cursor synchronization script. This database schema satisfies the requirements for generating push requests.

Lesson 3: Creating a MobiLink Project [page 114]
: In this lesson, you connect to the consolidated database by creating a new MobiLink project.

Lesson 4: Configuring the Notifier [page 115]
: In this lesson, you configure a Notifier event to define how the Notifier creates push requests, and sends push notifications to devices.

Lesson 5: Starting the MobiLink Server [page 117]
: In this lesson, you start the MobiLink server with the Notifier so that push notifications can be sent to devices.

Lesson 6: Setting up a Remote Database [page 118]
: In this lesson, you create a SQL Anywhere remote database, create a synchronization publication, a user, and a subscription.

Lesson 7: Configuring the MobiLink Listener [page 120]
: In this lesson, you configure the MobiLink Listener by storing the MobiLink Listener options in a text file, and then running dblsn with the file name specified at the command line.

Lesson 8: Issuing Push Requests [page 121]
: In this lesson, you make a change to the Dealer table and prompt a server-initiated synchronization.

Lesson 9: Cleaning up [page 123]
: Remove tutorial materials from your computer.

Task overview: Server-initiated Synchronization Tutorials [page 108]

Next task: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Related Information

Server-initiated Synchronization [page 4]
Tutorial: Using Central Administration of Remote Databases
Server-initiated Remote Tasks (SIRT)

1.8.1.1 Lesson 1: Setting up the Consolidated Database

In this lesson, you create a consolidated database named **`SIS_CarDealer_LP_DBLSN_CONDB`** with the scripts required for synchronization using the dbinit utility. You then use the SQL Anywhere 17 driver to define an ODBC data source for the **`SIS_CarDealer_LP_DBLSN_CONDB`** database.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Create a new working directory to store the consolidated database. This tutorial assumes `c:\MLsis` as the working directory.
2. Create the SQL Anywhere consolidated database using the dbinit utility, and set the DBA user ID to DBA and password to passwd.
3. Change to the `c:\MLsis` directory and run the following command:

   ```
   dbinit -dba DBA,passwd SIS_CarDealer_LP_DBLSN_CONDB
   ```
4. To stat the consolidated database, run the following command.

   ```
   dbsrv17 SIS_CarDealer_LP_DBLSN_CONDB
   ```
5. Click *Start* > *Programs* > *SQL Anywhere 17* > *Administration Tools* > *ODBC Data Source Administrator*.
6. Click the *User DSN* tab, and then click *Add*.
7. In the *Create New Data Source* window, click *SQL Anywhere 17* and click *Finish*.
8. Perform the following tasks in the *ODBC Configuration For SQL Anywhere* window:
   a. Click the *ODBC* tab.
   b. In the *Data source name* field, type **`SIS_CarDealer_LP_DBLSN_CONDB`**.
   c. Click the *Login* tab.
   d. In the *User ID* field, type **`DBA`**.
   e. In the *Password* field, type **`passwd`**.
   f. From the *Action* dropdown list, choose *Connect to a running database on this computer*.
   g. In the *Server name* field, type **`SIS_CarDealer_LP_DBLSN_CONDB`**.
   h. Click *OK*.
9. Close the ODBC data source administrator.

   Click *OK* on the *ODBC Data Source Administrator* window.

Results

The consolidated database is created and an ODBC data source is defined.

Next Steps

Proceed to the next lesson.

Related Information

ODBC Data Sources

1.8.1.2 Lesson 2: Generating a Database Schema

In this lesson, you generate a database schema, which includes a Dealer table, a non_sync_request table, a download_cursor synchronization script. This database schema satisfies the requirements for generating push requests.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Click *Start* > *Programs* > *SQL Anywhere 17* > *Administration Tools* > *SQL Central*.
2. Perform the following tasks to connect to the consolidated database:
   a. Click *Connections* > *Connect with SQL Anywhere 17*.
   b. From the *Action* dropdown list, choose *Connect with an ODBC Data Source*.
   c. Click *ODBC Data Source name*, and click *Browse*.
   d. Select `SIS_CarDealer_LP_DBLSN_CONDB` and then click *OK*.
   e. Click *Connect*.
3. Connect to your database using Interactive SQL.

   You can start Interactive SQL from SQL Central or at a command prompt.

- To start Interactive SQL from SQL Central, right-click the **`SIS_CarDealer_LP_DBLSN_CONDB - DBA`** database and click *Open Interactive SQL*.
- To start Interactive SQL at a command prompt, run the following command:

  ```
  dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
  ```

4. Execute the following SQL statements to create and set up the Dealer and non_sync_request tables:

   ```
   CREATE TABLE Dealer (
       name           VARCHAR(10) NOT NULL PRIMARY KEY,
       rating         VARCHAR(5),
       last_modified  TIMESTAMP DEFAULT TIMESTAMP
   )
   CREATE TABLE non_sync_request(
       poll_key       VARCHAR(128)
   )
   ```

5. Insert data into the Dealer table using the following statements:

   ```
   INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
   INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
   INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
   INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
   INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
   INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
   INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
   INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
   INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'I');
   COMMIT;
   ```

6. Execute the following SQL statement to create the MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 17*\ with the location of your SQL Anywhere 17 installation.

   ```
   READ "C:\Program Files\SQL Anywhere 17\MobiLink\setup\syncsa.sql"
   ```

7. Run the following SQL script to specify a download_cursor synchronization script and record the synchronization state to the ml_sis_sync_state system table:

   ```
   CALL ml_add_table_script(
       'CarDealer',
       'Dealer',
       'download_cursor',
       'SELECT * FROM Dealer WHERE last_modified >= ?'
   );
   CALL ml_add_connection_script(
       'CarDealer',
       'publication_nonblocking_download_ack',
       'CALL ml_set_sis_sync_state(
           {ml s.remote_id},
           NULL,
           {ml s.publication_name},
           {ml s.username},
           NULL,
           {ml s.last_publication_download}
       )'
   );
   CALL ml_add_table_script(
       'CarDealer', 'Dealer', 'download_delete_cursor', '--{ml_ignore}'
   );
   COMMIT;
   ```

This script sets the ml_sis_sync_state to record download-only synchronization. Recording the synchronization state allows you to reference the ml_sis_sync_state system table from the request_cursor event. You specify the request_cursor event in the next lesson.

8. Close Interactive SQL.

Results

The database schema is created.

Next Steps

Proceed to the next lesson.

Related Information

Synchronization Scripts
SQL Anywhere Database Server Executable (dbsrv17, dbeng17)
CREATE TABLE Statement
download_cursor Table Event
publication_nonblocking_download_ack Connection Event

1.8.1.3 Lesson 3: Creating a MobiLink Project

In this lesson, you connect to the consolidated database by creating a new MobiLink project.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Click *Start* > *Programs* > *SQL Anywhere 17* > *Administration Tools* > *SQL Central* .

2. Click *Tools* > *MobiLink 17* > *New Project*.
3. In the *Name* field, type **`SIS_CarDealer_LP_DBLSN_CONDB_project`**.
4. In the *Location* field, type `C:\MLsis`, and then click *Next*.
5. In the *Database display name* field, type **`SIS_CarDealer_LP_DBLSN_CONDB`**.
6. Click *Edit*. The *Connect To A Generic ODBC Database* window appears.
7. In the *User ID* field, type **`DBA`**.
8. In the *Password* field, type **`passwd`**.
9. In the *ODBC Data Source* name field, click *Browse*, and then select **`SIS_CarDealer_LP_DBLSN_CONDB`**.
10. Click *OK*, and then click *Save*.
11. Check the *Remember the password* option, and then click *Next*.
12. On the *New Remote Database Schema* page, select only the *Dealer* table to synchronize. Click *Next*.
13. Click *Next* again and then click *Finish*. Click *OK*.

Results

The MobiLink project is created.

Next Steps

Proceed to the next lesson.

1.8.1.4 Lesson 4: Configuring the Notifier

In this lesson, you configure a Notifier event to define how the Notifier creates push requests, and sends push notifications to devices.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

The request_cursor event script detects push requests. Each push request determines what information is sent, and which device should receive the information.

Procedure

1. In the left pane of SQL Central under *MobiLink 17*, expand **`SIS_CarDealer_LP_DBLSN_CONDB_project`**, *Consolidated Databases* and then **`SIS_CarDealer_LP_DBLSN_CONDB - DBA`**.
2. Right-click *Notification*, and then click *New* > *Notifier*.
3. In the *What do you want to name the new notifier* field, type **`CarDealerNotifier`**.
4. Click *Finish*.
5. In the right pane, select **`CarDealerNotifier`**, and then click *Properties*.
6. Click the *Events* tab and click *request_cursor* from the *Events* list.
7. Type the following SQL statement in the provided text field:

```
SELECT ml_sis_sync_state.remote_id + '.sync' FROM ml_sis_sync_state
WHERE
(
    EXISTS (SELECT 1 FROM Dealer
        WHERE last_modified >= ml_sis_sync_state.last_download)
)
```

8. Click *OK* to save the Notifier event.

Results

A Notifier is created and configured.

Next Steps

Proceed to the next lesson.

Related Information

request_cursor Event [page 46]
ml_set_sis_sync_state System Procedure [page 105]

1.8.1.5 Lesson 5: Starting the MobiLink Server

In this lesson, you start the MobiLink server with the Notifier so that push notifications can be sent to devices.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

Connect to your consolidated database.

Run the following command:

```
mlsrv17 -notifier -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB" -o serverOut.txt -v+ -dl
-zu+ -x tcpip
```

The following table describes the mlsrv17 options used in this lesson. The options -o and -v provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v is typically not used in production.

Option	Description
-notifier	Starts all enabled Notifiers for server-initiated synchronization.
-c	Specifies a connection string.
-o	Specifies the message log file `serverOut.txt`.
-v+	Specifies what information is logged. Using -v+ sets maximum verbose logging.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and protocol options for MobiLink clients.

Results

The MobiLink server messages window appears. The Notifier indicates that it is ready to receive push notification requests from devices.

Next Steps

Proceed to the next lesson.

Related Information

MobiLink Server
MobiLink Server Options
-notifier mlsrv17 Option
-c mlsrv17 Option
-o mlsrv17 Option
-v mlsrv17 Option
-zu mlsrv17 Option
-x mlsrv17 Option

1.8.1.6 Lesson 6: Setting up a Remote Database

In this lesson, you create a SQL Anywhere remote database, create a synchronization publication, a user, and a subscription.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Create your MobiLink client database using the dbinit command line utility.

   From the `c:\MLsis` directory, run the following command:

   ```
   dbinit -dba DBA,passwd SIS_CarDealer_LP_DBLSN_REM
   ```
2. Start your MobiLink client database using the dbsrv17 command line utility.

   Run the following command:

   ```
   dbsrv17 SIS_CarDealer_LP_DBLSN_REM
   ```
3. Connect to your MobiLink client database using Interactive SQL.

Run the following command:

```
dbisql -c "SERVER=SIS_CarDealer_LP_DBLSN_REM;UID=DBA;PWD=passwd"
```

4. Create the Dealer table in the remote database.

   Execute the following SQL statements in Interactive SQL:

```
CREATE TABLE Dealer (
    name           VARCHAR(10) NOT NULL PRIMARY KEY,
    rating         VARCHAR(5),
    last_modified  TIMESTAMP DEFAULT TIMESTAMP
)
COMMIT;
```

5. Create your MobiLink synchronization user, publication, and subscription.

   Execute the following SQL statements in Interactive SQL:

```
CREATE PUBLICATION CarDealer(TABLE DEALER WHERE 0=1)
CREATE SYNCHRONIZATION USER test_mluser OPTION ScriptVersion='CarDealer'
CREATE SYNCHRONIZATION SUBSCRIPTION TO CarDealer FOR test_mluser
SET OPTION public.ml_remote_id = 'remote_id';
COMMIT;
```

Results

A SQL Anywhere remote database, synchronization publication, user, and subscription are all created.

Next Steps

Proceed to the next lesson.

Related Information

MobiLink Clients
Publications
Script Versions
CREATE TABLE Statement
CREATE PUBLICATION Statement [MobiLink] [SQL Remote]
CREATE SYNCHRONIZATION USER Statement [MobiLink]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

1.8.1.7 Lesson 7: Configuring the MobiLink Listener

In this lesson, you configure the MobiLink Listener by storing the MobiLink Listener options in a text file, and then running dblsn with the file name specified at the command line.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Run the following command to synchronize with the MobiLink server and create the `SIS_CarDealer_LP_DBLSN_REM.rid` file:

   ```
   dbmlsync -c "SERVER=SIS_CarDealer_LP_DBLSN_REM;UID=DBA;PWD=passwd" -e sa=on -
   o rem1.txt -v+
   ```

   The MobiLink Listener can use the $remote_id action variable to define a poll key, which the MobiLink server uses to identify the device. This variable is retrieved from the remote ID file, `SIS_CarDealer_LP_DBLSN_REM.rid`, which is created during the initial synchronization with the MobiLink server. You must synchronize with the MobiLink server to make use of the remote ID file.
2. Click *Shut Down* on the SQL Anywhere MobiLink client window.
3. Create a MobiLink Listener command file by creating a text file with the following contents:

   ```
   # Verbosity level
   -v2
   # Show notification messages in console and log
   -m
   # Truncate, then write output to dblsn.txt
   -ot dblsn.txt
   # Remote ID file (defining the scope of $remote_id)
   -r SIS_CarDealer_LP_DBLSN_REM.rid
   # Message handlers
   # Signal dbmlsync to launch, sync and then shutdown
   -l "poll_connect='tcpip(host=localhost)';
       poll_notifier=CarDealerNotifier;
       poll_key=$remote_id.sync;
       action='run dbmlsync.exe -c
   SERVER=SIS_CarDealer_LP_DBLSN_REM;UID=DBA;PWD=passwd -e sa=on -o rem1.txt -v
   +';"
   ```
4. This tutorial assumes `c:\MLsis` as the working directory for server-side components. Save the text file as `mydblsn.txt` in this directory.
5. Start the MobiLink Listener.

   At a command prompt, navigate to `c:\MLsis` or to the directory where you saved your MobiLink Listener command file.

Start the MobiLink Listener by running the following command:

```
dblsn @mydblsn.txt
```

The *MobiLink Listener for Windows* window appears, indicating the MobiLink Listener is sleeping.

Results

The MobiLink Listener is configured.

Next Steps

Proceed to the next lesson.

Related Information

Listeners [page 18]
MobiLink Listener Utility for Windows (dblsn) [page 61]
@data dblsn Option [page 66]
Action Variables [page 22]

1.8.1.8 Lesson 8: Issuing Push Requests

In this lesson, you make a change to the Dealer table and prompt a server-initiated synchronization.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

In this lesson, you make a change to the Dealer table in the consolidated database so that the information can be downloaded into the remote database when the MobiLink Listener polls for push notifications. You then prompt a server-initiated synchronization by inserting a poll key value into the consolidated database. The

Notifier runs the request_cursor event, detects the poll key in the non_sync_request table, then sends a push notification to the MobiLink Listener. When the MobiLink Listener receives the push notification, it synchronizes with the MobiLink database and updates the remote database.

Procedure

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=SIS_CarDealer_LP_DBLSN_CONDB"
   ```
2. Execute the following SQL statements:

   ```
   UPDATE Dealer
       SET RATING = 'B' WHERE name = 'Geo';
   COMMIT;
   ```
3. Wait a few seconds for the synchronization to occur.

   The MobiLink Listener should poll the consolidated database, download the push notification, then update the Dealer table on the remote database.
4. Confirm that the Dealer table on the remote database was updated.

   Execute the following SQL statement:

   ```
   SELECT * FROM Dealer
   ```

   The rating for **`Geo`** should now be **`B`**.

Results

A change is made in the consolidated database and server-initiated synchronization is initiated.

Next Steps

Proceed to the next lesson.

Related Information

How to Work with Push Requests [page 12]
INSERT Statement
UPDATE Statement
DELETE Statement

1.8.1.9 Lesson 9: Cleaning up

Remove tutorial materials from your computer.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Close Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related ODBC data sources.
    a. Start the ODBC Data Source Administrator.

       At a command prompt, type the following command:

       ```
       odbcad32
       ```
    b. Remove the **`SIS_CarDealer_LP_DBLSN_CONDB`** data source.
4. Navigate to the directory containing your consolidated and remote databases, `c:\MLsis\`, and delete all the files.

Results

The tutorial materials are removed from your computer.

Next Steps

Proceed to the next lesson.

1.8.2 Tutorial: Configuring Server-initiated Synchronization Using Gateways

This tutorial demonstrates how to configure a SQL Anywhere consolidated and remote database for server-initiated synchronization. It is based on the sample code located in *%SQLANYSAMP17%*`\MobiLink\SIS_CarDealer`.

Prerequisites

You require a basic knowledge of MobiLink event scripts.

The following software is required:

- SQL Anywhere 17

You must have the following roles and privileges on the consolidated database:

- SYS_AUTH_RESOURCE_ROLE compatibility role
- MONITOR system privilege

You must have the following roles and privileges on the remote database:

- SYS_REPLICATION_ADMIN_ROLE system role
- SYS_RUN_REPLICATION_ROLE system role

Context

Several sample implementations of server-initiated synchronization are located in *%SQLANYSAMP17%*`\MobiLink`. All server-initiated synchronization sample directory names begin with the SIS_ prefix.

In this tutorial, you:

- Set up a SQL Anywhere consolidated database for server-initiated synchronization.
- Configure server-side properties.
- Issue push requests to prompt a server-initiated synchronization.

1. Lesson 1: Setting up the Consolidated Database [page 125]
   In this lesson, you create a consolidated database named **`MLconsolidated`** with the scripts required for synchronization using the dbinit utility. You then define an ODBC data source for the database.
2. Lesson 2: Generating a Database Schema [page 127]
   In this lesson, you generate the database schema, which includes a Dealer table and a download_cursor synchronization script. A table and stored procedure is used to generate server-initiated synchronization push requests.
3. Lesson 3: Creating a Table to Store Push Request [page 129]
   In this lesson, you create a push request table for storing push requests. The Notifier sends a message to a device when it detects a push request.

4. Lesson 4: Creating a MobiLink Project [page 130]
   In this lesson, you connect to the consolidated database by creating a new MobiLink project.
5. Lesson 5: Configuring the Notifier [page 131]
   In this lesson, you configure three Notifier events to define how the Notifier creates push requests, transmits the requests to MobiLink Listeners, and deletes expired requests.
6. Lesson 6: Configuring Gateways and Carriers [page 134]
   Gateways are the mechanisms for sending messages.
7. Lesson 7: Starting the MobiLink Server [page 136]
   In this lesson, you start the MobiLink server with the Notifier so that push notifications can be sent to devices.
8. Lesson 8: Setting up a Remote Database [page 137]
   In this lesson, you create a SQL Anywhere remote database, create a synchronization publication, a user, and a subscription.
9. Lesson 9: Configuring the MobiLink Listener [page 139]
   In this lesson, you configure the MobiLink Listener by storing the MobiLink Listener options in a text file, and then running dblsn with the file name specified at the command line.
10. Lesson 10: Issuing Push Requests [page 141]
   You can issue push requests by populating the PushRequest table directly, or making a change in the Dealer table.
11. Lesson 11: Cleaning up [page 143]
   Remove tutorial materials from your computer.

Task overview: Server-initiated Synchronization Tutorials [page 108]

Previous task: Tutorial: Configuring Server-initiated Synchronization Using Lightweight Polling [page 109]

Related Information

Server-initiated Synchronization [page 4]

1.8.2.1 Lesson 1: Setting up the Consolidated Database

In this lesson, you create a consolidated database named `MLconsolidated` with the scripts required for synchronization using the dbinit utility. You then define an ODBC data source for the database.

Prerequisites

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Create a new working directory to store the consolidated database.

   This tutorial assumes `c:\MLsis` as the working directory.
2. Create the SQL Anywhere consolidated database using the dbinit utility.
3. Run the following command:

   ```
   dbinit -dba DBA,passwd MLconsolidated
   ```
4. Start the consolidated database using the dbsrv17 utility.

   Run the following command:

   ```
   dbsrv17 MLconsolidated
   ```
5. Click *Start* > *Programs* > *SQL Anywhere 17* > *Administration Tools* > *ODBC Data Source Administrator*.
6. Click the *User DSN* tab, and then click *Add*.
7. In the *Create New Data Source* window, click *SQL Anywhere 17*, and then click *Finish*.
8. Perform the following tasks in the *ODBC Configuration For SQL Anywhere* window:
   a. Click the *ODBC* tab.
   b. In the *Data source name* field, type **`sis_cons`**.
   c. Click the *Login* tab.
   d. In the *User ID* field, type **`DBA`**.
   e. In the *Password* field, type **`passwd`**.
   f. From the *Action* dropdown list, choose *Connect to a running database on this computer*.
   g. In the *Server name* field, type **`MLconsolidated`**.
   h. Click *OK*.
9. Close the ODBC data source administrator.

   Click *OK* on the *ODBC Data Source Administrator* window.

Results

The consolidated database is created and an ODBC data source is defined.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Next task: Lesson 2: Generating a Database Schema [page 127]

Related Information

ODBC Data Sources

1.8.2.2 Lesson 2: Generating a Database Schema

In this lesson, you generate the database schema, which includes a Dealer table and a download_cursor synchronization script. A table and stored procedure is used to generate server-initiated synchronization push requests.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Click *Start* > *Programs* > *SQL Anywhere 17* > *Administration Tools* > *SQL Central*.
2. Perform the following tasks to connect to the consolidated database:
    a. Click *Connections* > *Connect with SQL Anywhere 17*.
    b. In the *User ID* field, type **`DBA`**.
    c. In the *Password* field, type **`passwd`**.
    d. From the *Action* dropdown list, click *Connect with an ODBC Data Source*.
    e. Click *ODBC Data Source name*, and click *Browse*.
    f. Select **`sis_cons`**, and then click *OK*.
    g. Click *Connect*.
3. Connect to your database using Interactive SQL.

    You can start Interactive SQL from SQL Central or at a command prompt.

    - To start Interactive SQL from SQL Central, right-click the **`MLconsolidated - DBA`** database and click *Open Interactive SQL*.
    - To start Interactive SQL at a command prompt, run the following command:

    ```
    dbisql -c "dsn=sis_cons"
    ```
4. Run the following SQL statement to create and set up the Dealer table:

    ```
    CREATE TABLE Dealer (
        name VARCHAR(10) NOT NULL PRIMARY KEY,
        rating VARCHAR(5),
        last_modified TIMESTAMP DEFAULT TIMESTAMP
    ```

```
)
```

5. Insert data into the Dealer table using the following statements:

```
INSERT INTO Dealer(name, rating) VALUES ('Audi', 'a');
INSERT INTO Dealer(name, rating) VALUES ('Buick', 'b');
INSERT INTO Dealer(name, rating) VALUES ('Chrysler', 'c');
INSERT INTO Dealer(name, rating) VALUES ('Dodge', 'd');
INSERT INTO Dealer(name, rating) VALUES ('Eagle', 'e');
INSERT INTO Dealer(name, rating) VALUES ('Ford', 'f');
INSERT INTO Dealer(name, rating) VALUES ('Geo', 'g');
INSERT INTO Dealer(name, rating) VALUES ('Honda', 'h');
INSERT INTO Dealer(name, rating) VALUES ('Isuzu', 'I');
COMMIT;
```

6. Run the following SQL script to create the MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 17* with the location of your SQL Anywhere 17 installation.

```
READ "C:\Program Files\SQL Anywhere 17\MobiLink\setup\syncsa.sql"
```

7. Run the following SQL script to specify a download_cursor synchronization script and record the synchronization:

```
CALL ml_add_table_script(
    'sis_ver1',
    'Dealer',
    'download_cursor',
    'SELECT * FROM Dealer WHERE last_modified >= ?'
);
CALL ml_add_table_script(
    'sis_ver1', 'Dealer', 'download_delete_cursor', '--{ml_ignore}'
);
COMMIT
```

   Do not close Interactive SQL.

Results

The database schema is generated, which includes a Dealer table and a download_cursor synchronization script, and MobiLink system tables and stored procedures are installed.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 1: Setting up the Consolidated Database [page 125]

Next task: Lesson 3: Creating a Table to Store Push Request [page 129]

Related Information

Synchronization Scripts
SQL Anywhere Database Server Executable (dbsrv17, dbeng17)
CREATE TABLE Statement
download_cursor Table Event

1.8.2.3 Lesson 3: Creating a Table to Store Push Request

In this lesson, you create a push request table for storing push requests. The Notifier sends a message to a device when it detects a push request.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. You should be connected to your database in Interactive SQL from the previous lesson.

   If you are not connected to your database, you can start Interactive SQL from SQL Central or at a command prompt.

   - To start Interactive SQL from SQL Central, right-click the **MLconsolidated - DBA** database and click *Open Interactive SQL*.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=sis_cons"
     ```
2. Run the following SQL statements in Interactive SQL:

   ```
   CREATE TABLE PushRequest (
       req_id INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
       mluser VARCHAR(128),
       subject VARCHAR(128),
       content VARCHAR(128),
       resend_interval VARCHAR(30) DEFAULT '20s',
       time_to_live VARCHAR(30) DEFAULT '1m',
       status VARCHAR(128) DEFAULT 'created'
   )
   COMMIT;
   ```
3. Close Interactive SQL.

Results

A push request table is created for storing push requests.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 2: Generating a Database Schema [page 127]

Next task: Lesson 4: Creating a MobiLink Project [page 130]

Related Information

Push Requests [page 9]
Server-initiated Synchronization [page 4]
Server-initiated Synchronization Components [page 6]

1.8.2.4 Lesson 4: Creating a MobiLink Project

In this lesson, you connect to the consolidated database by creating a new MobiLink project.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. From *SQL Central*, click *Tools* > *MobiLink 17* > *New Project*.
2. In the *Name* field, type **`sis_cons_project`**.
3. In the *Location* field, type `C:\MLsis`, and then click *Next*.

4. In the *Database display name* field, type **`sis_cons`**.
5. Click *Edit*. The *Connect To A Generic ODBC Database* window appears.
6. In the *User ID* field, type **`DBA`**.
7. In the *Password* field, type **`passwd`**.
8. In the *ODBC Data Source name* field, click *Browse*, and then select **`sis_cons`**.
9. Click *OK*, and then click *Save*.
10. Check the *Remember the password* option, and then click *Next*.
11. Accept the defaults on the *New Remote Database Schema* page and click *Next*.
12. Click *Next* and then click *Finish*. Click *OK*.

Results

The MobiLink project is created.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 3: Creating a Table to Store Push Request [page 129]

Next task: Lesson 5: Configuring the Notifier [page 131]

1.8.2.5 Lesson 5: Configuring the Notifier

In this lesson, you configure three Notifier events to define how the Notifier creates push requests, transmits the requests to MobiLink Listeners, and deletes expired requests.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

The Notifier detects changes in the consolidated database and creates push requests using the begin_poll event. In this case, the begin_poll script populates the PushRequest table if changes occur in the Dealer table and when a remote database is not up-to-date.

The request_cursor script fetches push requests. Each push request determines what information is sent in the message, and which remote databases receive the information.

The request_delete Notifier event specifies cleanup operations. Using this script, the Notifier can automatically remove implicitly dropped and expired requests.

Procedure

1. In the left pane of SQL Central under *MobiLink 17*, expand **`sis_cons_project`**, *Consolidated Databases* and then **`sis_cons`**.
2. Right-click *Notification*, and then click *New* › *Notifier*.
3. In the *What do you want to name the new notifier* field, type **`CarDealerNotifier`**.
4. Click *Finish*.
5. Enter the begin_poll event script.
    a. In the right pane, select **`CarDealerNotifier`**, and then click *File* › *Properties*.
    b. Click the *Events* tab.
    c. Choose *begin_poll* from the *Events* list.
    d. Type the following SQL statements in the provided text field:

```
--
-- Insert the last consolidated database
-- modification date into @last_modified
--
DECLARE @last_modified timestamp;
SELECT MAX(last_modified) INTO @last_modified FROM Dealer;
--
-- Delete processed requests if the mluser is up-to-date
--
DELETE FROM PushRequest
    FROM PushRequest AS p, ml_user AS u, ml_subscription AS s
    WHERE p.status = 'processed'
        AND u.name = p.mluser
        AND u.user_id = s.user_id
        AND @last_modified <= GREATER(s.last_upload_time,
s.last_download_time);
--
-- Insert new requests when a device is not up-to-date
--
INSERT INTO PushRequest(mluser, subject, content)
SELECT u.name, 'sync', 'ignored'
    FROM ml_user as u, ml_subscription as s
    WHERE u.name IN (SELECT name FROM ml_listening WHERE listening = 'y')
        AND u.user_id = s.user_id
        AND @last_modified > greater(s.last_upload_time,
s.last_download_time)
        AND u.name NOT LIKE '%-dblsn'
        AND NOT EXISTS(SELECT * FROM PushRequest
            WHERE PushRequest.mluser = u.name
```

```
                 AND PushRequest.subject = 'sync')
```

In the first major section of the begin_poll script, processed requests from the PushRequest table are eliminated if a device is up to date:

```
@last_modified <= GREATER(s.last_upload_time, s.last_download_time)
```

@last_modified is the maximum modification date in the consolidated database Dealer table. The expression greater(s.last_upload_time, s.last_download_time) represents the last synchronization time for a remote database.

You can also delete push requests directly using the request_delete event. However, the begin_poll event, in this case, ensures that expired or implicitly dropped requests are not eliminated before a remote database synchronizes.

The next section of code checks for changes in the last_modified column of the Dealer table and issues push requests for all active MobiLink Listeners (listed in the ml_listening table) that are not up to date:

```
@last_modified > GREATER(s.last_upload_time, s.last_download_time)
```

When populating the PushRequest table, the begin_poll script sets the subject to 'sync'.

6. Enter the request_cursor script.
   a. Click *request_cursor* from the *Events* list.
   b. Type the following SQL statement in the provided text field:

```
SELECT
    p.req_id,
    'Default-DeviceTracker',
    p.subject,
    p.content,
    p.mluser,
    p.resend_interval,
    p.time_to_live
    FROM PushRequest AS p
```

The PushRequest table supplies rows to the request_cursor script.

The order and values in the request_cursor result set is significant. The second parameter, for example, defines the default gateway Default-DeviceTracker. A device tracking gateway keeps track of how to reach users and automatically selects UDP or SMTP to connect to remote devices.

7. Enter the request_delete script.
   a. Click *request_delete* from the *Events* list.
   b. Type the following SQL statement in the provided text field:

```
UPDATE PushRequest SET status='processed' WHERE req_id = ?
```

Instead of deleting the row, this request_delete script updates the status of a row in the PushRequest table to 'processed'.

8. Click *Apply* and then click *OK* to save the Notifier events.

Results

Three Notifier events are defined.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 4: Creating a MobiLink Project [page 130]

Next task: Lesson 6: Configuring Gateways and Carriers [page 134]

Related Information

Device Tracking Gateways [page 28]
request_delete Event [page 48]
begin_poll Event [page 42]
request_cursor Event [page 46]
request_delete Event [page 48]

1.8.2.6 Lesson 6: Configuring Gateways and Carriers

Gateways are the mechanisms for sending messages.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

You can define either a supported gateway or a device tracking gateway. The MobiLink server keeps track of how to reach clients when you specify a device tracking gateway, and automatically chooses the most appropriate gateway.

Procedure

You use a default device tracking gateway for the purpose of this tutorial, so configuration is not necessary.

Results

The default gateway is used.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 5: Configuring the Notifier [page 131]

Next task: Lesson 7: Starting the MobiLink Server [page 136]

Related Information

Gateways as an Alternative to Lightweight Pollers [page 27]
Gateways and Carriers [page 27]
Device tracking Gateway Properties [page 57]

1.8.2.7 Lesson 7: Starting the MobiLink Server

In this lesson, you start the MobiLink server with the Notifier so that push notifications can be sent to devices.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

Connect to your consolidated database.

Run the following command:

```
mlsrv17 -notifier -c "dsn=sis_cons" -o serverOut.txt -v+ -dl -zu+ -x tcpip
```

The following table describes the mlsrv17 options used in this lesson. The options -o and -v provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v is typically not used in production.

Option	Description
-notifier	Starts all enabled Notifiers for server-initiated synchronization.
-c	Specifies a connection string.
-o	Specifies the message log file `serverOut.txt`.
-v+	Specifies what information is logged. Using -v+ sets maximum verbose logging.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and protocol options for MobiLink clients.

Results

The MobiLink server messages window appears. The Notifier indicates that it is ready to receive push notification requests from devices.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 6: Configuring Gateways and Carriers [page 134]

Next task: Lesson 8: Setting up a Remote Database [page 137]

Related Information

MobiLink Server
MobiLink Server Options
-notifier mlsrv17 Option
-c mlsrv17 Option
-o mlsrv17 Option
-v mlsrv17 Option
-zu mlsrv17 Option
-x mlsrv17 Option

1.8.2.8 Lesson 8: Setting up a Remote Database

In this lesson, you create a SQL Anywhere remote database, create a synchronization publication, a user, and a subscription.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Create your MobiLink client database using the dbinit command line utility.

   Run the following command:

   ```
   dbinit -dba DBA,passwd remote1
   ```

2. Start your MobiLink client database using the dbsrv17 command line utility.

   Run the following command:

   ```
   dbsrv17 remote1
   ```

3. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "SERVER=remote1;UID=DBA;PWD=passwd"
   ```

4. Create the Dealer table.

   Execute the following SQL statements in Interactive SQL:

   ```
   CREATE TABLE Dealer (
       name            VARCHAR(10) NOT NULL PRIMARY KEY,
       rating          VARCHAR(5),
       last_modified   TIMESTAMP DEFAULT TIMESTAMP
   )
   COMMIT;
   ```

5. Create your MobiLink synchronization user, publication, and subscription.

   Execute the following SQL statements in Interactive SQL:

   ```
   CREATE PUBLICATION car_dealer_pub (table Dealer);
   CREATE SYNCHRONIZATION USER sis_user1;
   CREATE SYNCHRONIZATION SUBSCRIPTION
       TO car_dealer_pub
       FOR sis_user1
       OPTION scriptversion='sis_ver1';
   COMMIT;
   ```

Results

A SQL Anywhere remote database, synchronization publication, user, and subscription are created.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 7: Starting the MobiLink Server [page 136]

Next task: Lesson 9: Configuring the MobiLink Listener [page 139]

Related Information

MobiLink Clients
Publications
Script Versions
CREATE TABLE Statement
CREATE PUBLICATION Statement [MobiLink] [SQL Remote]
CREATE SYNCHRONIZATION USER Statement [MobiLink]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

1.8.2.9 Lesson 9: Configuring the MobiLink Listener

In this lesson, you configure the MobiLink Listener by storing the MobiLink Listener options in a text file, and then running dblsn with the file name specified at the command line.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Create a MobiLink Listener command file by creating a text file with the following contents:

```
#----------------------------------
# Verbosity level
-v2
# Show notification messages in console and log
-m
# Polling interval, in seconds
-i 3
# Truncate, then write output to dblsn.txt
-ot dblsn.txt
# MobiLink address and connect parameter for dblsn
-x "host=localhost"
# Enable device tracking and specify the MobiLink user name.
-t+ sis_user1
# Message handlers
# Synchronize using dbmlsync
-l "subject=sync;
action='start dbmlsync.exe
 -c SERVER=remote1;UID=DBA;PWD=passwd
 -o dbmlsyncOut.txt
';"
```

2. This tutorial assumes `c:\MLsis` as the working directory for server-side components. Save the text file as `mydblsn.txt` in this directory.

3. Start the MobiLink Listener.

   At a command prompt, navigate to `c:\MLsis` or the directory where you saved your MobiLink Listener command file.

   Start the MobiLink Listener by running the following command:

   ```
   dblsn @mydblsn.txt
   ```

Results

The *MobiLink Listener for Windows* window appears, indicating the MobiLink Listener is sleeping.

When tracking information is uploaded to the consolidated database, a new entry appears in the MobiLink server messages window. This information relays the successful initial communication between the MobiLink Listener and the MobiLink server.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 8: Setting up a Remote Database [page 137]

Next task: Lesson 10: Issuing Push Requests [page 141]

Related Information

Listeners [page 18]
MobiLink Listener Utility for Windows (dblsn) [page 61]
@data dblsn Option [page 66]

1.8.2.10 Lesson 10: Issuing Push Requests

You can issue push requests by populating the PushRequest table directly, or making a change in the Dealer table.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

For server-initiated synchronization, you can issue push requests by populating the PushRequest table directly, or making a change in the Dealer table. In the latter case, the Notifier begin_poll script detects the change in the Dealer table and populate the PushRequest table. In either case, the PushRequest table supplies rows to the Notifier request_cursor script, which determines how remote devices receive messages.

Procedure

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=sis_cons"
   ```
2. Execute the following SQL statements:

   ```
   INSERT INTO PushRequest(mluser, subject, content)
       VALUES ('sis_user1', 'sync', 'not used');
   COMMIT;
   ```
3. Wait a few seconds for the synchronization to occur.

   When populated, the PushRequest table supplies rows to the Notifier's request_cursor script. The request_cursor script determines what information is sent in the message, and which remote devices receive the information.
4. If the dbmlsync windows is still open, click *Shut down* to close it.
5. Execute the following SQL statements to make a change in the **`Dealer`** table on the consolidated database, prompting server-initiated synchronization:

   ```
   UPDATE Dealer
       SET RATING = 'B' WHERE name = 'Geo';
   COMMIT;
   ```
6. Wait a few seconds for the synchronization to occur.

In this case, the Notifier begin_poll script detects changes in the dealer table and populates the PushRequest table appropriately. As before, once the PushRequest table is populated, the Notifier request_cursor script determines what information is sent in the message, and which remote devices receive the information.

7. Confirm that the Dealer table on the remote database was updated.

   Execute the following SQL statement:

   ```
   SELECT * FROM Dealer
   ```

   The rating for **`Geo`** should now be **`B`**.

Results

A push request is inserted directly into the PushRequest table, prompting server-initiated synchronization.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 9: Configuring the MobiLink Listener [page 139]

Next task: Lesson 11: Cleaning up [page 143]

Related Information

How to Work with Push Requests [page 12]
INSERT Statement
UPDATE Statement

1.8.2.11 Lesson 11: Cleaning up

Remove tutorial materials from your computer.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Close Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related ODBC data sources.
    a. Start the ODBC Data Source Administrator.

       At a command prompt, type the following command:

       ```
       odbcad32
       ```
    b. Remove the **`sis_cons`** data source.
4. Navigate to the directory containing your consolidated and remote databases, `c:\MLsis\`, and delete all the files.

Results

The tutorial materials are removed from your computer.

Task overview: Tutorial: Configuring Server-initiated Synchronization Using Gateways [page 124]

Previous task: Lesson 10: Issuing Push Requests [page 141]

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP