Inhaltsverzeichnis

Allgemein 1
Was tut das Plugin? 1
Funktionen 1
Unterstützte Datenbankmotoren 1
Zweck & Anwendung 2
Installation 2
Anforderungen 2
Konfiguration 2
Übersicht 2
Datenbankspezifische Einstellungen 3
Tabellenspezifische Einstellungen 5
Columns 5
Fehlerdiagnose 6
Datenbank 6
Columns 7
Logdatei 7
Beispiel Variablen in SQL-Datenbank synchronisieren 7
Trigger 9
Beispielscript 10
Entities 12
Exchange 12
Channel 12
Tabelle 12
Spalte 13
Ordner & Dateien 13
Ordner 13
Dateien 13
Versionsinformation 13
Dieses Dokument 13
Plugin 13
Assembly 13

SQL Exchange Plugin

Allgemein

Das SQL Exchange Plugin erlaubt es Ihnen, CoDaBix® Datenpunktnodewerte bidirektional mit außenliegenden Datenbanken wie MyQSL, Oracle oder Microsoft SQL Server auszutauschen.

Was tut das Plugin?

Das Plugin lässt Sie:

- Datenbanken (Verbindungen) definieren
- Tabellen browsen oder definieren
- Tabellenspalten browsen oder definieren
- Spalten lesen und schreiben, indem es einen Read oder Write SQL Ausdruck spezifiziert

Funktionen

- Spalten einer Datenbankzeile lesen (ausgewählt von der *Read SQL Expression*)
- Spalten einer Datenbankzeile schreiben, entweder durch Einsetzen einer neuen Zeile oder durch Updaten einer existierenden Zeile (ausgewählt von der *Write SQL Expression*)
- Abbonieren von Spalten einer Datenbankzeile, wenn die entsprechenden CoDaBix® Variablennodes abboniert sind

Unterstützte Datenbankmotoren

- MySQL 5.5 oder höher
- Microsoft SQL Server 2008 oder höher
- Oracle
- SQLite

Zweck & Anwendung

- Eine Tabellenreihe anhängen oder eine existierende Reihe updaten
- Daten aus der neuesten Zeile order einer spezifischen Zeile einer Tabelle lesen
- Subscriptions erstellen, um benachrichtigt zu werden, wenn Daten in der Datenbank verändert werden
- Aufrufen von **Stored Procedures** über Methoden-Nodes in Codabix (wird derzeit nur für **Oracle**-Datenbanken unterstützt)

Installation

Dieses Plugin ist Bestandteil des CoDaBix® Setups. Bitte konsultieren Sie CoDaBix® Setup und erster Start für weitere Informationen darüber, wie dieses Plugin installiert und deinstalliert werden kann.

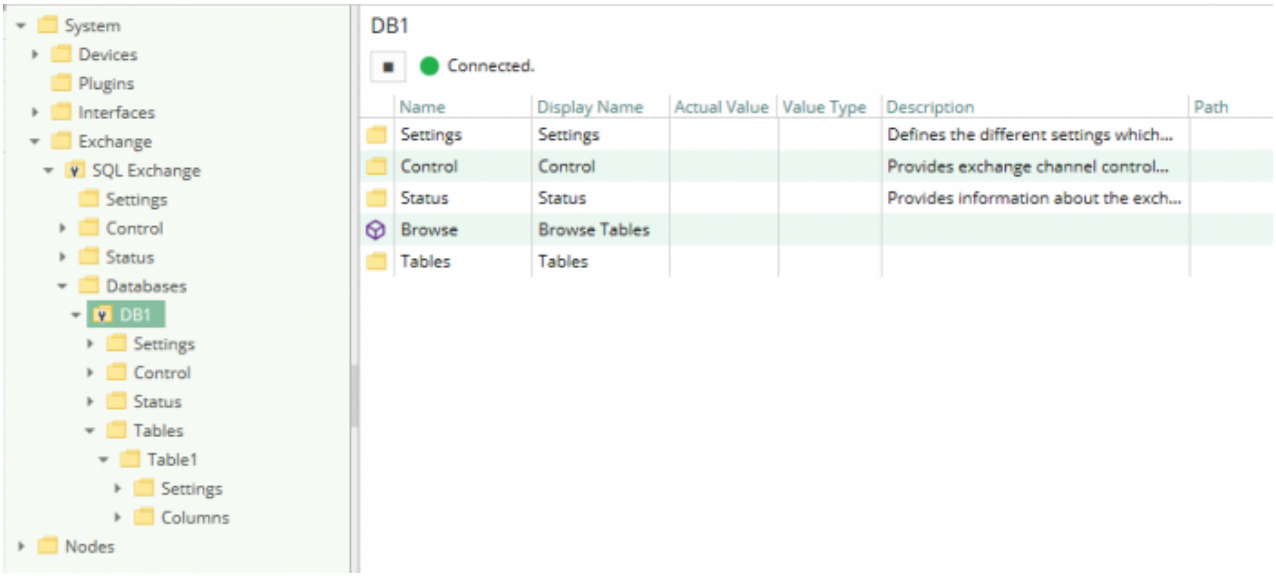
Anforderungen

- Die Maschine, die CoDaBix® betreibt, muss Zugang zu einem der unterstützten Datenbankmotoren haben.

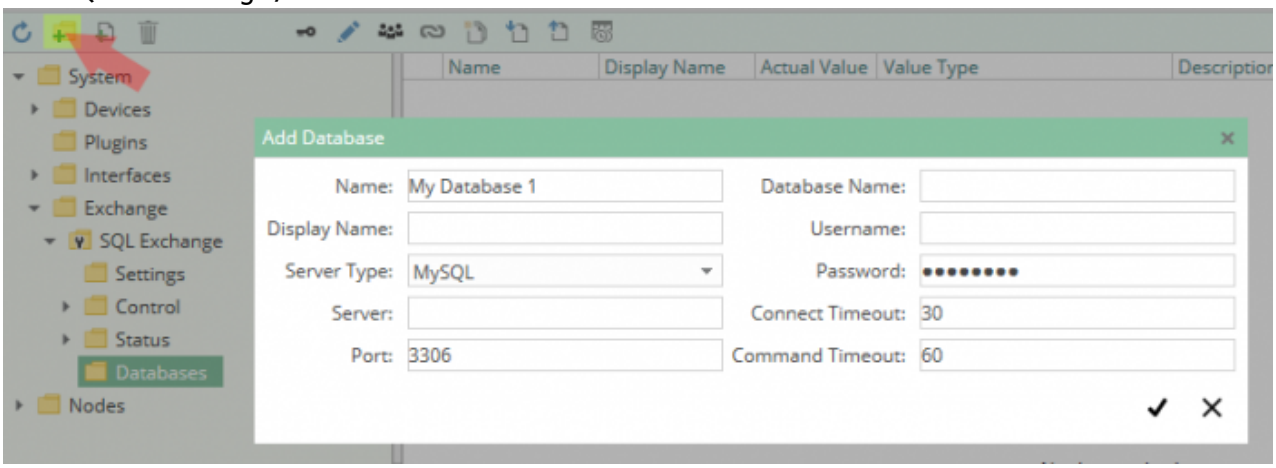
Konfiguration

Übersicht

Die gesamte SQL Exchange Pluginkonfiguration befindet sich unter dem Nodepfad `/System/Exchange/SQL Exchange`.



Der Nodebaum im oberen Bild zeigt den Standardnodebaum des SQL Exchange Plugins. Um eine oder mehrere SQL Exchange **Datenbanken** aufzusetzen, fügen Sie einen Folder Node unter dem Node `SQL Exchange/Databases` hinzu, oder machen Sie einen Rechtsklick auf den `SQL Exchange/Databases` Node und wählen Sie `Add Database` aus.

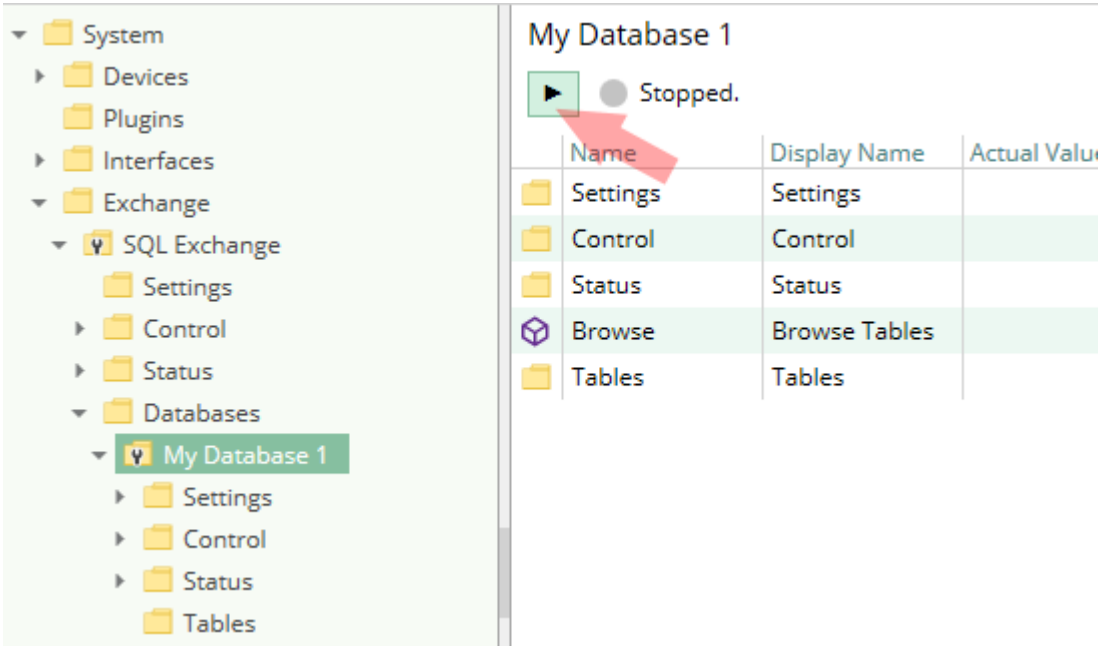


Datenbankspezifische Einstellungen

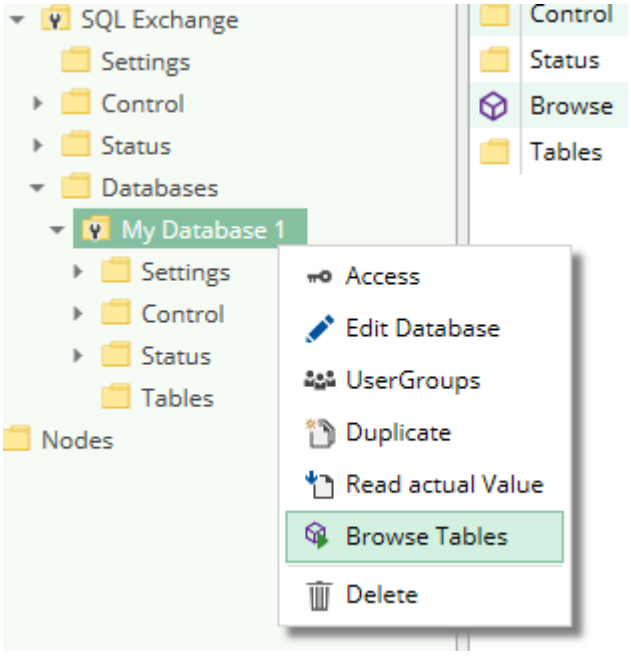
Name	Typ	Beschreibung
Server Type	Enum	Definiert den Servertyp, zu dem die Datenbankverbindung hergestellt wird. Gültige Werte: `MySQL`, `Oracle`, `MSSQL`, `ODBC`, `SQLite`
Server	String	Der Hostname oder die IP Adresse, zu der die Verbindung hergestellt werden soll. Für **SQLite** ist dies der Dateipfad zur Datenbank; andere Verbindungsproperties wie `Port` werden hierbei ignoriert.
Port	Integer	Der Port, zu dem die Verbindung hergestellt werden soll. Bei der Verwendung von `MSSQL` können Sie `0` als Port angeben, in welchem Fall eine Verbindung zur standardmäßigen SQL-Server-instanz hergestellt werden soll.
Database Name	String	Der Name der Datenbank bzw. des Schemas. Für **ODBC** ist dies der Data Source Name (DSN) und ist das einzige Feld, dass angegeben werden muss. Die DSN kann unter Windows in der App **ODBC-Datenquellen** konfiguriert werden (Benutzer-DSN oder System-DSN). Beachten Sie, dass bei Codabix (x86) die 32-Bit- und bei Codabix (x64) die 64-Bit-ODBC-Datenquellen-App verwendet werden muss.

Name	Typ	Beschreibung
Username	String	Der Benutzername für die Verbindung.
Password	Password	Das Passwort.
Connect Timeout	Integer	Der Timeout in Sekunden, nachdem ein Versuch zu verbinden abgebrochen wird.
Command Timeout	Integer	Der Timeout in Sekunden, nach dem ein anhaltender Befehl abgebrochen wird.

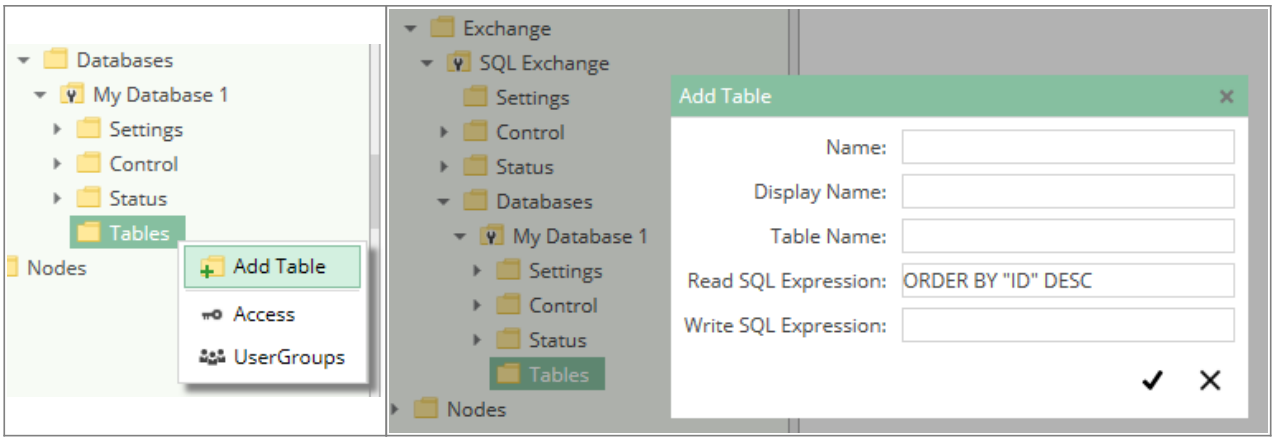
Nachdem Sie „Save“ geklickt haben, wird die Datenbanknode erstellt. Sie können Sie starten, indem Sie die Datenbanknode auswählen und den Startbutton klicken:



Um eine oder mehrere Tabellen (`Tables`) für die Datenbank zu erstellen, können Sie einen Rechtsklick auf die Datenbanknode machen und dann den `Browse Tables` Menüeintrag verwenden:



Alternativ können Sie einen Rechtsklick auf den `Tables` Node machen und `Add Table` auswählen (um eine existierende Tabelle zu bearbeiten, machen Sie einen Rechtsklick auf den `Table` Node und wählen Sie `Edit Table` aus):

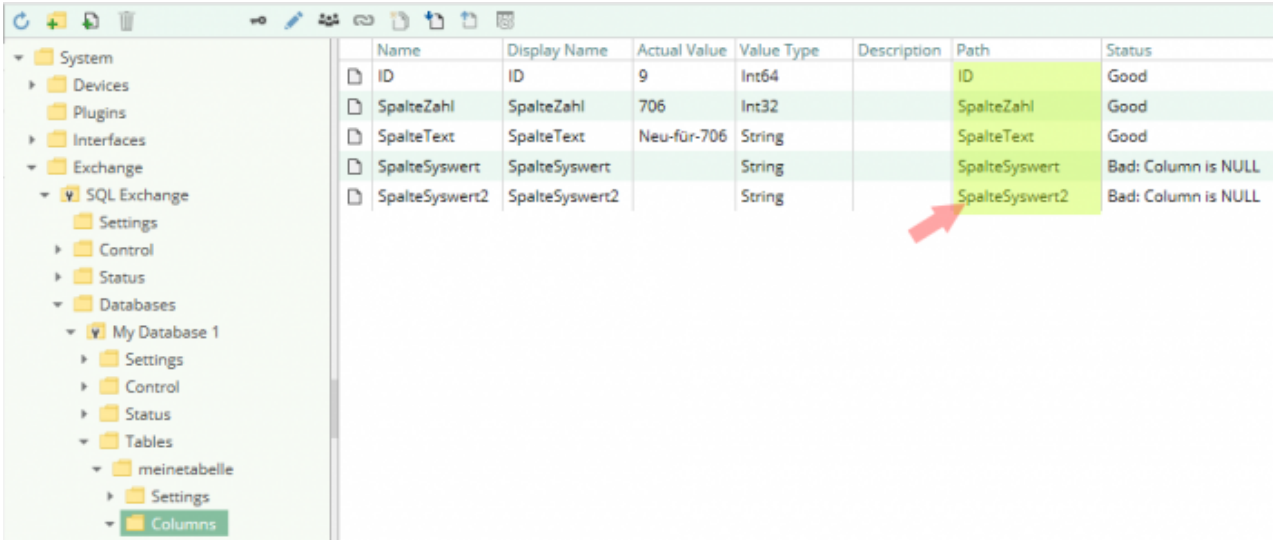


Tabellenspezifische Einstellungen

Name	Typ	Beschreibung
Table Name	String	Der name der Tabelle in der Datenbank.
Read SQL Expression	String	Ein SQL String (z.B. ein `WHERE`-Satz oder ein `ORDER BY`-Satz) der angewandt wird, wenn die Spalten gelesen werden. Die erste Reihe, die von der Datenbank zurückgeliefert wird, wird gelesen. Sie können einen `WHERE`-Satz spezifizieren (z.B. `WHERE "ID" = 123`), um nur eine spezifische Spalte zu lesen. Standard ist `ORDER BY "ID" DESC`, um Reihen absteigend nach ID zu ordnen, damit die Reihe mit der höchsten ID benutzt wird.
Write SQL Expression	String	Ein SQL String (z.B. ein `WHERE`-Satz), der angewandt wird, wenn die Spalten geschrieben werden. Sie können einen `WHERE`-Satz spezifizieren (z.B. `WHERE "ID" = 123`), um eine existierende Reihe zu updaten. Der Standardwert ist ein leerer String, was bedeutet, dass eine neue Reihe jedes Mal dann eingefügt wird, wenn Spaltennodes geschrieben werden.

Columns

Jeder Node innerhalb der `Columns` Node einer `Table` Node kann mithilfe der `Path` Eigenschaft einer Spalte (`Column`) einer Tabelle zugewiesen werden.



Syntax:

`<ColumnName>`	Nur der Spaltenname ist spezifiziert. Wenn aus ihm gelesen/in ihn geschrieben wird, werden die Read SQL or Write SQL Ausdrücke der Tabelle benutzt. Beispiel: `MyColumn1`

`<ColumnName>;` `<ReadWriteExpression>`	Der Spaltenname ist gemeinsam mit einem Ausdruck spezifiziert, der beim Lesen oder Schreiben aus der/in die Spalte benutzt wird. Beispiel: `MyColumn1; WHERE ID = 123`

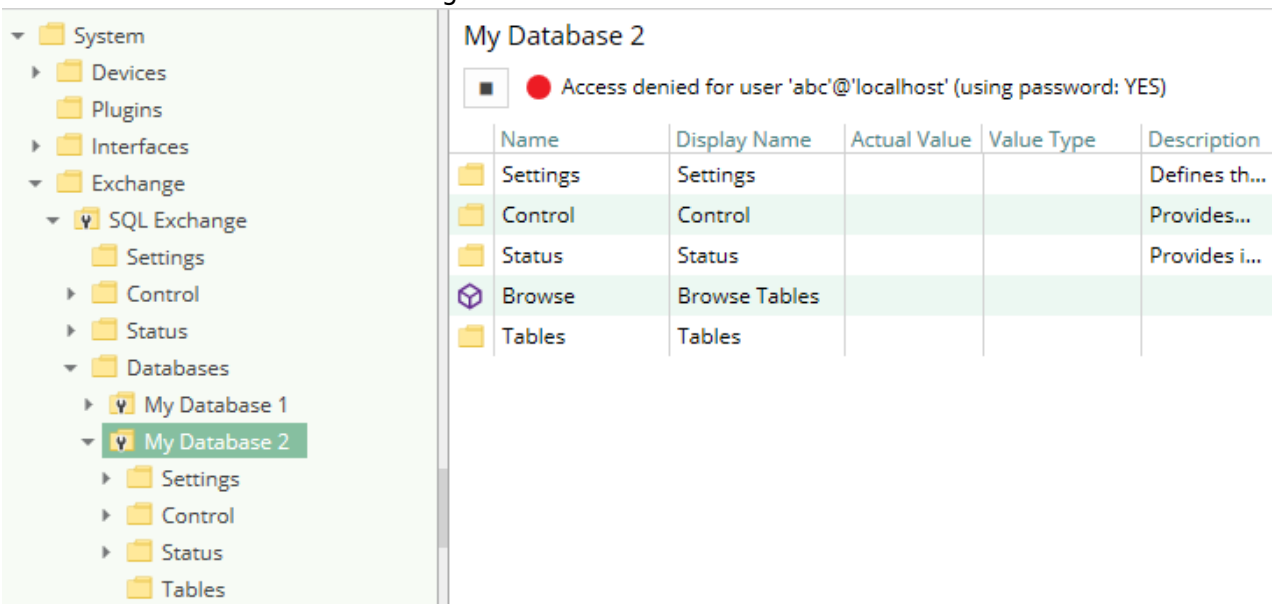
`<ColumnName>;`
`<ReadExpression>;`
`<WriteExpression>`
	Der Spaltenname ist gemeinsam mit einem Ausdruck, der beim Lesen aus der Spalte benutzt wird und einem anderen Ausdruck, der beim Schreiben in die Spalte benutzt wird, spezifiziert.
	Beispiel: `MyColumn1; WHERE ID = 123; WHERE ID = 456`

Fehlerdiagnose

Das SQL Exchange Plugin liefert je nach zu untersuchender Schicht verschiedene Statusinformationen. Generell werden die kanalbasierten Diagnoseinformationen durch den Verbindungsstatus des Channels zum Datenbankserver produziert. Die variablenbasierten Diagnoseinformationen werden während des Lese- / Schreibzugruffs auf die verschiedenen Spalten produziert.

Datenbank

Um den Status der verschiedenen Datenbankkanäle zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf das folgende Bild:



Das obige Bild zeigt das Bedienfeld des Datenbankkanals, das alle statusrelevanten Informationen anzeigt. Das Bedienfeld aktualisiert automatisch seine Statusinformation, wenn ein neuer Status verfügbar ist.

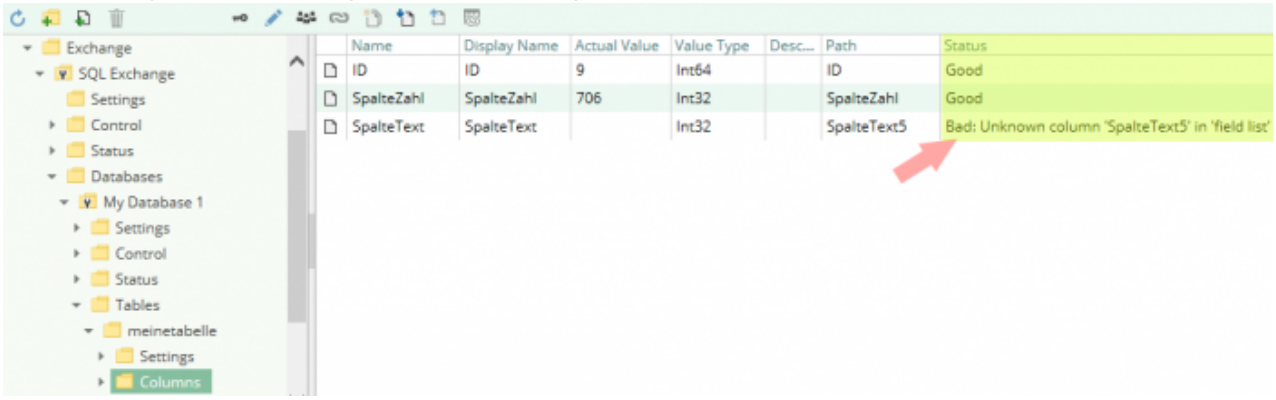
Statuskreis

Farbe	Bedeutung
	Der Datenbankkanal ist gestoppt. Klicken Sie den ► Button, um ihn zu starten.

Farbe	Bedeutung
	Der Datenbankkanal startet oder stoppt gerade oder wartet auf den Verbindungsaufbau.
	Der Datenbankkanal läuft und es wurde erfolgreich eine Verbindung hergestellt. Sie können ihn durch Klick auf den ■ Button stoppen.
	Der Datenbankkanal läuft, aber die Verbindung ist momentan fehlerhaft. Bitte überprüfen Sie den Statustext für weitere Informationen.

Columns

Um den Status der verschiedenen Tabellenspalten zu überwachen und zu diagnostizieren, werfen Sie einen Blick auf die in CoDaBix® angezeigte `Status`-Eigenschaft der Spalte. Benutzen Sie den Button „Read actual Value“, um die Werte aus der Datenbank auszulesen und das Ergebnis in den Spaltennodes zu speichern.



Logdatei

Alle datenbankbezogenen Statusinformationen werden auch in die datenbankspezifische Logdatei im `[LoggingFolder]` protokolliert. Jede Logdatei wird nach dem Namensschema `SQL Exchange.<DatabaseName>.log` benannt. Der Inhalt einer solchen Logdatei kann wie folgt aussehen:

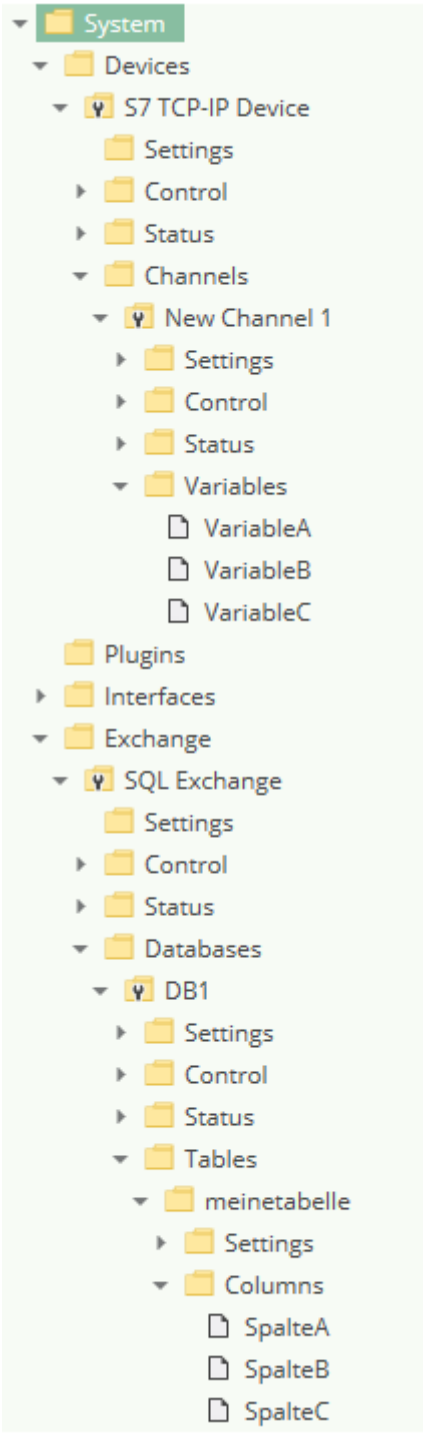
```
...
[14:55:34 25.07.2017] - Error (Severity=High): Code=[-1], Text=[Access
denied for user 'abc'@'localhost' (using password: YES)], Details=[]
...
```

Beispiel Variablen in SQL-Datenbank synchronisieren

Unter diesem Beitrag finden Sie ein Beispielscript das zum Synchronisieren von Werten von einer Quelle (z.B. S7 Device Plugin oder OPC UA Client Device) in ein Ziel (z.B. SQL Exchange Plugin) verwendet werden kann.

Das Beispielscript enthält oben eine Liste an Nodes, wo jeweils die Quell-Node und Ziel-Node angegeben ist.
Für das Beispielscript könnten die Nodes im S7-Plugin und im SQL Exchange Plugin so

aussehen („VariableA“ aus S7 wird in „SpalteA“ aus SQL geschrieben. „VariableB“ wird in „SpalteB“ geschrieben usw.):



Informationen zum Einrichten des SQL Exchange Plugins für den Zugriff auf eine SQL-Datenbank finden Sie weiter oben beschrieben.

Für Ihr CoDaBix müssen Sie dann die Nodeliste im Script (Zeile 6-10) entsprechend anpassen, so dass diese nach diesesm Muster aussieht:

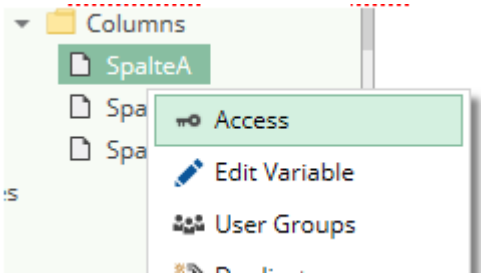
```
    const syncNodePaths: [string, string][] = [
        "/Pfad/zur/QuellNode_1", "/Pfad/zur/ZielNode_1"],
        "/Pfad/zur/QuellNode_2", "/Pfad/zur/ZielNode_2"],
```

```
    ];
```

sodass es immer eine Quelle und ein Ziel gibt.

Hier ist die Quelle eine SPS-Variable und das Ziel eine Datenbank-Spalte, dies kann aber auch umgekehrt erfolgen, sodass z.B. aus einer Datenbankzeile Werte gelesen und in die SPS geschrieben werden.

Den kompletten Node-Path einer Node bekommen Sie, wenn Sie diese auswählen und dann rechtsklicken und auf „Access“ gehen:



Trigger

Für das Auslösen der Synchronisierung (d.h. das Übertragen der Werte von den Quell-Nodes in die Ziel-Nodes) gibt es verschiedene Möglichkeiten.

Eine einfache Möglichkeit ist, dieses in einem regelmäßigen Intervall auszulösen. Dies ist in dem Beispielscript aktuell implementiert (siehe Zeile 29-37, „Option A“).
Hier wird in einer Schleife immer 2 Sekunden gewartet und dann durch den Aufruf von „await synchronizeNodesAsync(syncNodes)“ die Synchronisierung gestartet.

Eine weitere Möglichkeit ist, eine separate Trigger-Node zu verwenden und die Synchronisierung nur dann auszulösen, wenn die Triggernode einen bestimmten Wert annimmt (z.B. „1“).
Dies ist im Script als „Option B“ in den Zeilen 40-48 beschrieben. Um diese Option zu verwenden, müssten Sie den Code von Option A auskommentieren und den Code von Option B aktivieren. Dieser Code erstellt eine Subscription, wodurch der Wert in einem bestimmten Intervall (z.B. 500 ms) gelesen wird und bei einer Wertänderung dann das ValueChanged-Event ausgelöst wird.

Hierzu müssen Sie dann den Pfad zur Trigger-Node angeben. Aktuell würde dann immer, wenn sich der Wert der Trigger-Node ändert, die Synchronisierung gestartet werden.

Soll die Synchronisierung nur ausgelöst werden, wenn die Trigger-Node einen bestimmten Wert wie „1“ annimmt, können Sie innerhalb des ValueChanged-EventListeners den aktuellen Wert der Trigger-Node abfragen, z.B.:

```
    let triggerNode = codabix.findNode("path/to/TriggerNode", true);
    triggerNode.addValueChangedEventListener(e => {
        if (e.newValue.value == 1)
            runtime.handleAsync(synchronizeNodesAsync(syncNodes));
    });
    // Read the Trigger node value every 500 ms.
    codabix.subscribeNodes([triggerNode], 500);
```

Um dieses Script zu verwenden, gehen Sie bitte im CoDaBix-Fenster auf den Menüpunkt „Script Interface“ und legen dort ein neues Script an. Wenn Sie auf dieses anschließend rechtsklicken, können Sie im Kontextmenü den Script-Editor öffnen (dies geht auch über einen Klick auf den Button in der Symbolleiste).

Nun löschen Sie bitte den vorhandenen Template-Code und fügen dann den Code aus der Textdatei ein. Wenn das Script dann mit dem aktuellen Code aktiv werden soll, markieren Sie die Checkbox „Go Live“ und klicken dann auf den Speichern-Button.

Beispielscript

SampleScriptSyncNodes.ts

```
runtime.handleAsync(async function () {

    // A list of node pairs with the source node and destination
node, which should be synchronized.
    const sourceNodePath = "/System/Devices/S7 TCP-IP
Device/Channels/New Channel 1/Variables/";
    const destNodePath = "/System/Exchange/SQL
Exchange/Channels/DB1/Tables/meinetabelle/Columns/";
    const syncNodePaths: [string, string][] = [
        [sourceNodePath + "VariableA", destNodePath + "SpalteA"],
        [sourceNodePath + "VariableB", destNodePath + "SpalteB"],
        [sourceNodePath + "VariableC", destNodePath + "SpalteC"]
    ];

    // Find the specified nodes.
    const syncNodes: [codabix.Node, codabix.Node][] = [];
    for (let syncNodePath of syncNodePaths) {
        let sourceNode = codabix.findNode(syncNodePath[0], true);
        let destNode = codabix.findNode(syncNodePath[1], true);

        syncNodes.push([sourceNode, destNode]);
    }


    // Option A:
    // Run a loop where we synchronize the source nodes into the
destination nodes every 2000 ms.
    while (true) {
        // Synchronize the nodes
        await synchronizeNodesAsync(syncNodes);

        // Wait 2000 ms
        await timer.delayAsync(2000);
    }

    // Option B:
    // Synchronize the nodes only when the value of the trigger
node has changed.
```

```
    // For this, we create a subscription specifying the interval,
and then use the value changed event.
    // let triggerNode = codabix.findNode("path/to/TriggerNode",
true);
    // triggerNode.addValueChangedEventListener(e => {
    //     codabix.scheduleCallback(() =>
runtime.handleAsync(synchronizeNodesAsync(syncNodes)));
    // }, true);
    // // Read the Trigger node value every 500 ms.
    // codabix.subscribeNodes([triggerNode], 500);



    async function synchronizeNodesAsync(syncNodes: [codabix.Node,
codabix.Node][]) {
        // Synchronize the source nodes into the destination
nodes.
        let readRequest = syncNodes.map((value => value[0]));
        logger.log("Reading " + readRequest.length + " node
values...");
        let values = await
codabix.readNodeValuesAsync(readRequest);

        // Check if every node has a value and a "Good" status,
and build the write request.
        let nodesToWrite: {
            node: codabix.Node,
            value: codabix.NodeValue
        }[] = [];

        for (let i = 0; i < values.length; i++) {
            if (values[i] == null)
                logger.logWarning("Node '" + readRequest[i] + "'
doesn't have a value; please check the device!");
            else if (values[i]!.status.isBad)
                logger.logWarning("Node '" + readRequest[i] + "'
has status '" + values[i]!.status.statusText + "'; please check
the device!");

            nodesToWrite.push({
                node: syncNodes[i][1],
                value: values[i] || new codabix.NodeValue(null)
            });
        }

        // Write the values.
        logger.log("Writing node values...");
        await codabix.writeNodeValuesAsync(nodesToWrite);
    }
```

```
} ());
```

Entities

Wie jedes Exchange Plugin erweitert das SQL Exchange Plugin das grundlegende CoDaBix Exchange Modell.

Exchange

Der Exchangetyp `SqlExchange` des Plugins definiert auch den `SqlExchangeChannel` und erweitert somit die grundlegenden `CodabixExchange` und `CodabixExchangeChannel` Entities. Während der `SqlExchange` lediglich eine Konkretisierung des `CodabixExchange` repräsentiert, erweitert der `SqlExchangeChannel` den `CodabixExchangeChannel` mit SQL Tabellenentities.

Channel

Jeder Channel wird von einem Channel Worker behandelt, der eine physische Verbindung zur Datenbank herstellt. Zum Zweck der Fehlerdiagnose untersucht der Worker die Datenbankverbindung alle 10 Sekunden, um den Statuscode des `Channels` und die Beschreibung zu aktualisieren, damit Verbindungsausfälle aufgespürt werden.

Standardmäßig liest der Worker keine Werte. Wenn ein Client oder Plugin einen synchronen Lesevorgang des `Channels` anfordert, liest der Channel Worker die Variablen in CoDaBix (z.B. unter Verwendung der CoDaBix Webkonfigurations-Funktion „Read actual value“) aus der Datenbank und schreibt sie dann in die entsprechenden CoDaBix Nodes.

Ähnlich schreibt der Channel Worker die Werte in eine Datenbank, die ein Client oder Plugin in die Variablen des Channels schreibt.

Um eine Datenbankvariable stetig gelesen zu bekommen, müssen Sie den Node in der Webkonfiguration bearbeiten und „History Options“ auf `Yes` stellen (was eine interne Subscription erstellt), oder Sie können z.B. einen OPC UA Client verbunden mit einem OPC UA Server Plugin benutzen und eine Subscription für die S7 Variablennodes anlegen. In diesem Fall liest der Channel Worker die Variablen in einem gleichmäßigen Intervall aus der Datenbank und, falls der Wert einer der Variablen sich verändert hat, schreibt den neuen Wert in den entsprechenden CoDaBix® Node.

Tabelle

Jede Tabellenentity repräsentiert die Informationen, die benötigt werden, um auf eine Datenbanktabelle zuzugreifen. Auf Tabellenlevel können Sie spezifizieren, wie die Zeile ausgewählt wird, aus der die Werte gelesen und in die die Werte geschrieben werden.

Spalte

Jede Spaltenentity repräsentiert eine einzige Datenbanktabellenspalte.

Ordner & Dateien

Ordner

Inhalt	Pfad	Zweck / Verwendung
AssemblyFolder	<CodabixInstallDir>/plugins/SqlExchangePlugin/	Beinhaltet die Plugin Assembly Datei.
ConfigFolder	<CodabixDataDir>/plugins/SqlExchangePlugin/	Beinhaltet die Plugin Konfigurationsdatei.
LoggingFolder	<CodabixDataDir>/log/	Beinhaltet die Plugin Log Dateien.

Dateien

Typ	Pfad	Zweck / Verwendung
Assembly	[AssemblyFolder]/CoDaBix.SqlExchangePlugin.dll	Die Plugin Assembly Datei.
Logging	[LoggingFolder]/SQL Exchange.<DatabaseName>.log	Die Log Datei.

Versionsinformation

Dieses Dokument

Datum	2018-06-14
Version	1.1

Plugin

Name	SQL Exchange Plugin
Node	/System/Exchange/SQL Exchange
Version	1.0.5

Assembly

Name	CoDaBix.SqlExchangePlugin.dll
Datum	2019-02-04
Version	1.0.5.0

From:
https://www.codabix.de/ - **CoDaBix®**

Permanent link:
https://www.codabix.de/de/plugins/exchange/sqlexchangeplugin

Last update: **2021/07/30 13:39**