Codesys MQTT library – Raspberry Pi / Siemens Simatic IOT 2000 Implementation

Με την εφαρμογή του MODBUS RTU , το Gateway είναι πλέον ικανό να λαμβάνει τα δεδομένα από το PLC μέσω της σειριακής επικοινωνίας . Τώρα αυτό που απομένει είναι η αποστολή των δεδομένων αυτών μέσω ενός IoT πρωτοκόλλου στο άλλο PLC της διάταξης . Για να επιτύχουμε το σκοπό αυτό χρησιμοποιούμε το πρωτόκολλο MQTT.

Τι είναι MQTT;

Το MQTT ή αλλιώς Message Queue Telemetry Transport είναι ένα ελαφρύ και «συμπαγές» πρωτόκολλο ανταλλαγής μηνυμάτων για μεταφορά δεδομένων σε απομακρυσμένες τοποθεσίες όπου απαιτείται περιορισμένο εύρος ζώνης δικτύου. Αυτά τα πλεονεκτήματα επιτρέπουν την εφαρμογή αυτού του πρωτοκόλλου σε M2M συστήματα (Machine to Machine) και IIoT (Industrial Internet of Things).

Ο σχεδιασμός και η ιστορία πίσω από το MQTT

Το MQTT αναπτύχθηκε για πρώτη φορά το 1999 , αλλά με την εκθετική ανάπτυξη του Διαδικτύου των Αντικειμένων (Internet of Things - IoT) και την ανάγκη σύνδεσης και επικοινωνίας μεταξύ έξυπνων συσκευών (smart devices) χαμηλής κατανάλωσης , το MQTT εγκαθιδρύθηκε πρόσφατα . Το MQTT σχεδιάστηκε προκειμένου να είναι ένα low-overhead πρωτόκολλο το οποίο θα λαμβάνει υπόψιν το περιορισμένο εύρος ζώνης και την περιορισμένη επεξεργαστική ισχύ των συσκευών. Σχεδιάστηκε με την ικανότητα να τρέχει σε ενσωματωμένα συστήματα (embedded systems) όπου θα παρέχει αξιόπιστη επικοινωνία

Χαρακτηριστικά του MQTT

Πιο συγκεκριμένα είναι ένα πρωτόκολλο δημοσίευσης / εγγραφής (Publish / Subscribe).Αποτελείται από clients οι οποίοι επικοινωνούν μέσω ενός server συχνά αποκαλούμενο ως **Broker**. Ένας client μπορεί είτε να στέλνει δεδομένα (**Publisher**) ,είτε να δέχεται τα δεδομένα (**Subscriber**).

Τα δεδομένα οργανώνονται σε μια ιεραρχία θεμάτων (**topics**). Όταν ένας Publisher έχει νέα δεδομένα, αποστέλλει ένα μήνυμα ελέγχου με αυτά στον Broker και κατόπιν αυτός , τα διανέμει σε οποιοδήποτε Client που έχει κάνει subscribe στο συγκεκριμένο θέμα. Ο Publisher δεν χρειάζεται να έχει πληροφορίες σχετικά με τον αριθμό ή την τοποθεσία των Subscribers.

Επίπεδα ποιότητας υπηρεσιών (**Quality of Service – QoS**): Το MQTT μας παρέχει 3 επίπεδα QoS για την παράδοση μηνυμάτων. Κάθε επίπεδο προσφέρει μεγαλύτερη αξιοπιστία παράδοσης με μεγαλύτερο κόστος αντιστοίχα [7,11]. Τα επίπεδα περιγράφονται αναλυτικά παρακάτω.

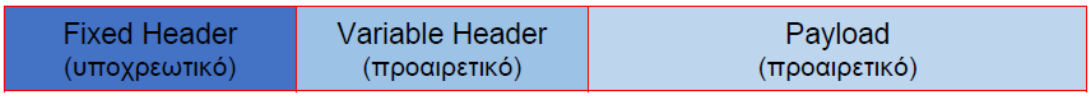
Ακόμη υπάρχει η δυνατότητα να ληφθούν μηνύματα σε ένα topic ακόμη και αν ο Client έκανε Subscribe μετά την αποστολή αυτών. Επίσης, μπορούμε να διατηρήσουμε τα Subscriptions ενός Client ακόμη και σε περίπτωση ξαφνικής

αποσύνδεσης, ώστε να συνεχίσει απρόσκοπτα μετά την επανασύνδεση του. Σε περιπτώσεις ξαφνικών αποσυνδέσεων έχουμε και την δυνατότητα
διαθήκης (will), η οποία εκτελεί κάποια συγκεκριμένη εργασία με την αποσύνδεση

Δομή των μηνυμάτων

Το μήνυμα που μεταδίδει το πρωτόκολλο αποτελείται από ένα MQTT πακέτο (packet) το οποίο ενθυλακώνεται από ένα TCP/IP packet. Στο σχήμα διακρίνουμε σε τρία μέρη την δομή ενός MQTT packet :

1. **Fixed Header** (Περιέχεται σε όλα τα μηνύματα)
2. **Variable Header** (Περιέχεται σε μερικά μηνύματα)
3. **Data, payload** (Περιέχεται σε μερικά μηνύματα)



Πιο συγκεκριμένα η σταθερή επικεφαλίδα (**Fixed Header**) αποτελείται επίσης από τρία μέρη :

- Είδος μηνύματος (**Message Type**): Παραδείγματος χάριν CONNECT, SUBSCRIBE,PUBLISH κ.ο.κ
- Συγκεκριμένες σημαίες για κάθε MQTT πακέτο (**Flags Specific to each MQTT packet**) : Αυτά τα 4 bits χρησιμοποιούνται ως βοηθητικές σημαίες ενώ η παρουσία και η κατάσταση αυτών εξαρτάται από το είδος του μηνύματος
- Υπόλοιπο μήκος (**Remaining Length**): Το τρέχον μήκος του μηνύματος (Variable Header + Data) . Το μέγεθός του είναι από 1 έως 4 bytes.

Είδος μηνύματος (Message Type)

Συνολικά, υπάρχουν 15 είδη μηνυμάτων στο πρωτόκολλο MQTT :

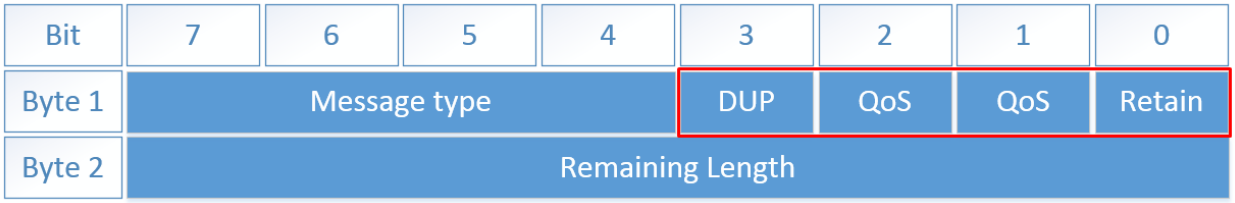
Είδος μηνύματος	Τιμή	Ροή μηνύματος	Περιγραφή
Reserved	0000 (0)	forbidden	Reserved
CONNECT	0001 (1)	**C* -> S****	Client request to connect to server
CONNACK	0010 (2)	**C <- S**	Connect acknowledgment
PUBLISH	0011 (3)	**C <- S, C -> S**	Publish message
PUBACK	0100 (4)	**C <- S, C -> S**	Publish acknowledgment
PUBREC	0101 (5)	**C <- S, C -> S**	Publish received
PUBREL	0110 (6)	**C <- S, C -> S**	Publish release
PUBCOMP	0111 (7)	**C <- S, C -> S**	Publish complete
SUBSCRIBE	1000 (8)	**C -> S**	Client subscribe request
SUBACK	1001 (9)	**C <- S**	Subscribe acknowledgment
UNSUBSCRIBE	1010 (10)	**C -> S**	Unsubscribe request
UNSUBACK	1011 (11)	**C <- S**	Unsubscribe acknowledgment

Είδος μηνύματος	Τιμή	Ροή μηνύματος	Περιγραφή
PINGREQ	1100 (12)	**C -> S**	PING request
PINGRESP	1101 (13)	**C <- S**	PING response
DISCONNECT	1110 (14)	**C -> S**	Client is disconnecting
Reserved	1111 (15)	Forbidden	Reserved

Όπου *C – Client, **S – Server

Σημαίες (Flags)

Τα πρώτα 4 πιο σημαντικά bits της σταθερής επικεφαλίδας χρησιμοποιούνται ως συγκεκριμένες σημαίες:



- **DUP**: Το bit Duplicate ενεργοποιείται όταν ένας Client ή ο Broker δεσμεύουν ένα πακέτο (χρησιμοποιείται στα πακέτα PUBLISH,SUBSCRIBE,UNSUBSCRIBE,PURBEL). Αν η σημαία έχει οριστεί τότε η κεφαλίδα μεταβλητής (Variable Header) πρέπει να περιέχει αναγνωριστικό μηνύματος (Message Identifier).
- **QoS**: Quality of Service (0,1,2)
- **RETAIN**: Κατά τη δημοσίευση δεδομένων με το Retain Flag ενεργοποιημένο, ο Broker θα το αποθηκεύσει. Όταν δημιουργηθεί μια νέα συνδρομή (subscription) σε αυτό το θέμα (topic), ο Broker θα στείλει αμέσως ένα μήνυμα με αυτή τη σημαία. Χρησιμοποιείται μόνο σε μηνύματα τύπου PUBLISH.

Variable Header

Η κεφαλίδα μεταβλητής υπάρχει σε μερικές κεφαλίδες και περιέχει τα παρακάτω δεδομένα :

- Αναγνωριστικό πακέτου (Packet Identifier): Υπάρχει σε όλα τα είδη μηνυμάτων εκτός από τα μηνύματα : CONNECT, CONNACK, PUBLISH(c QoS <1), PINGREQ, PINGRESP, DISCONNECT
- Όνομα πρωτοκόλλου (Protocol name) : Υπάρχει μόνο στα μηνύματα τύπου CONNECT

- Έκδοση πρωτοκόλλου (Protocol Version) : Υπάρχει μόνο στα μηνύματα τύπου CONNECT
- Σημαίες σύνδεσης (Connect Flags): Σημαίες που καθορίζουν τη συμπεριφορά του Client κατά την σύνδεση.

Δεδομένα & φορτίο (Data & Payload)

Το περιεχόμενο και η μορφή των δεδομένων που μεταδίδονται μέσω μηνυμάτων MQTT ορίζονται στη συσκευή. Το μέγεθος των δεδομένων μπορεί να υπολογιστεί αφαιρώντας το μήκος της κεφαλίδας μεταβλητής (Variable Header) από το υπόλοιπο μήκος

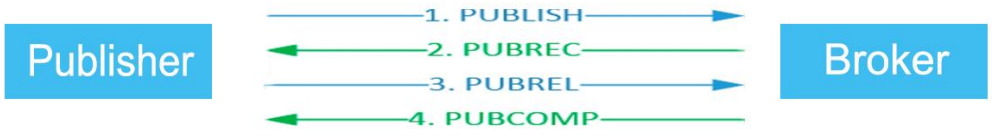
Quality of service στο πρωτόκολλο MQTT (QoS)

Όπως έχει αναφερθεί και προηγουμένως , το MQTT υποστηρίζει 3 επίπεδα Quality of service (QoS) κατά την αποστολή μηνυμάτων.

1. **QoS 0 At most once**
   Σε αυτό το επίπεδο ο Publisher στέλνει ένα μήνυμα στο Broker μια φορά χωρίς να περιμένει καμία απάντηση.

2. **QoS 1 At least once**
   Αυτό το επίπεδο εγγυάται την παράδοση του μηνύματος στον Broker, ωστόσο είναι δυνατή η επανάληψη μηνυμάτων από τον Publisher. Μόλις ληφθεί ένα αντίγραφο, ο Broker στέλνει αυτό το μήνυμα στου Subscribers ξανά και προωθεί την επιβεβαίωση παραλαβής μηνύματος στον Publisher.Εάν ο Publisher δεν λάβει ένα PUBACK μήνυμα από τον Broker ,θα επιχειρήσει να επαναφέρει αυτό το πακέτο ορίζοντας το bit DUP 1.

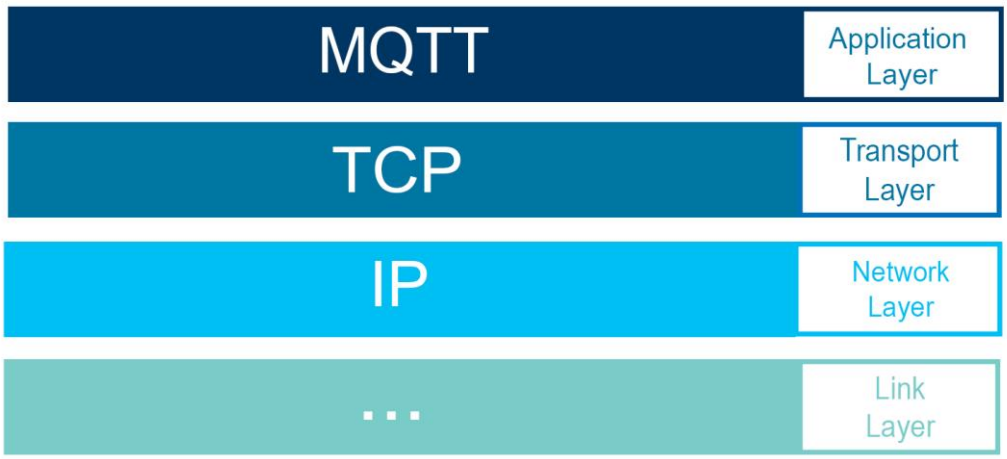
3. **QoS 2 Exactly once**
   Σε αυτό το επίπεδο, η παράδοση του μηνύματος στον Client είναι εγγυημένη χωρίς την πιθανότητα αντιγράφων.



Ο Publisher στέλνει ένα μήνυμα στο Broker . Το μήνυμα περιέχει το μοναδικό αναγνωριστικό πακέτου , QoS=2 και DUP=0. Ο Publisher αποθηκεύει το μήνυμα που δεν έχει αναγνωριστεί, εκτός εάν λάβει απάντηση PUBREC από τον Broker. Ο Broker απαντά με το μήνυμα PUBREC που περιέχει το ίδιο αναγνωριστικό πακέτου. Αφού ληφθεί το μήνυμα , ο Publisher στέλνει PUBREL με το ίδιο αναγνωριστικό πακέτου. Ο Broker πρέπει να αποθηκεύσει αντίγραφο του μηνύματος μέχρι να λάβει PUBREL. Μόλις ο Broker το λάβει, διαγράφει το αντίγραφο του μηνύματος και στέλνει στο Publisher ένα PUBCOMP μήνυμα σχετικά με την ολοκλήρωση της μετάδοσης.

Σύνδεση Client-Server

Στο μοντέλο αναφοράς πέντε επιπέδων TCP/IP, το MQTT πρωτόκολλο βρίσκεται στο επίπεδο εφαρμογής (που περιλαμβάνει τα επίπεδα 5-7 του OSI μοντέλου), δηλαδή σε παραπάνω επίπεδο από τα πρωτόκολλα TCP/IP, επομένως κάθε Client και Server απαιτούν TCP/IP stack για να μπορούν να συνδεθούν όπως παρατηρούμε στο παρακάτω σχήμα



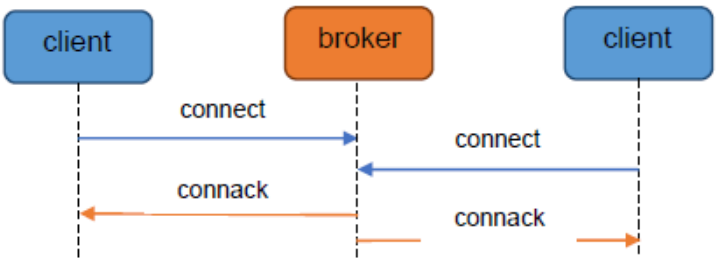
Ο **MQTT Client** έχει τις παρακάτω δυνατότητες:

- Σύνδεση (Connect)
- Δημοσίευση (Publish)
- Συνδρομή (Subscribe)
- Διαγραφή συνδρομής (Unsubscribe)
- Αποσύνδεση (Disconnect)

Ο MQTT Broker βρίσκεται στο επίκεντρο κάθε πρωτοκόλλου δημοσίευσης/συνδρομής (Publish/Subscribe). Ανάλογα με την εφαρμογή , ένας Broker μπορεί να χειριστεί έως και χιλιάδες ταυτόχρονα συνδεδεμένους MQTT Clients. Ο Broker είναι υπεύθυνος για τη λήψη όλων των μηνυμάτων , το φιλτράρισμά τους, το προσδιορισμό του MQTT Client που έχει εγγραφεί σε κάθε

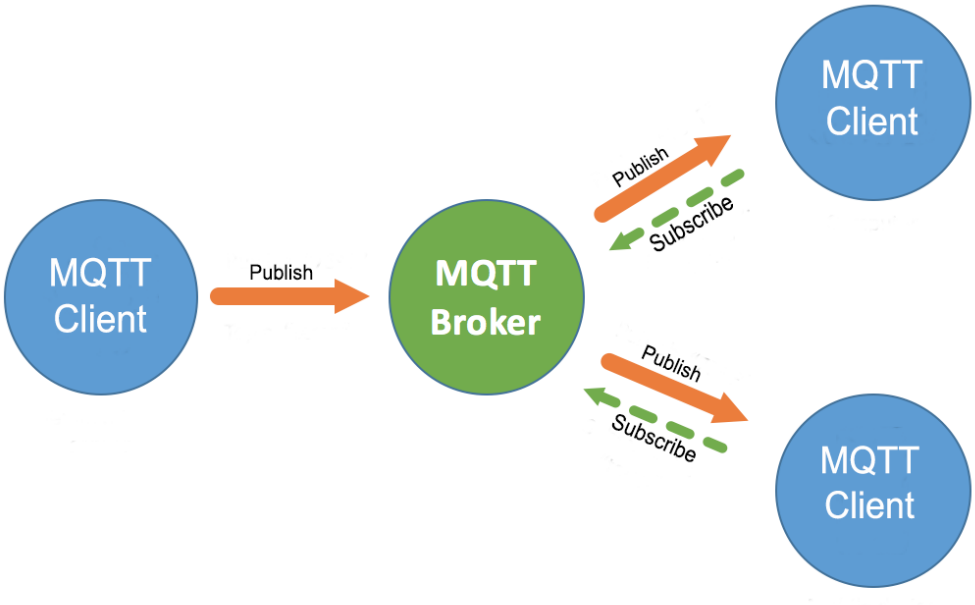
μήνυμα και την αποστολή του μηνύματος σε αυτούς τους εγγεγραμμένους πελάτες. Επίσης είναι υπεύθυνος για την εξασφάλιση της ορθής αποστολής και αποθήκευσης μηνυμάτων σε περίπτωση αποτυχίας, την αποθήκευση των Persisted Clients, και την εφαρμογή των απαιτούμενων Security features. Επομένως καταλαβαίνουμε ότι σε ένα scalable δίκτυο έχει μεγάλη σημασία η ικανότητα του Broker να διαχειρίζεται τον όγκο των δεδομένων με αξιοπιστία.

Είναι πολύ σημαντικό να αναφερθεί ότι όλοι οι Clients συνδέονται αποκλειστικά και μόνο με τον Broker χωρίς την δυνατότητα να δουν ή να συνδεθούν με άλλους Clients. Αυτό είναι πρωταρχικής σημασίας για την απλότητα των Clients σε υπολογιστικό επίπεδο και για το scalability του συστήματος.



Μοντέλο Publish-Subscribe

Το Publish/Subscribe είναι μοντέλο επικοινωνίας όπου οι αποστολείς των μηνυμάτων (εκδότες/Publishers) δεν αποστέλλουν τα μηνύματα απευθείας στους παραλήπτες (συνδρομητές/Subscribers).Ο Publisher αποστέλλει ένα μήνυμα ελέγχου το οποίο περιλαμβάνει τα δεδομένα στον Broker και αυτός με την σειρά του τα αποστέλλει στου MQTT Clients που έχουν κάνει εγγραφή στο συγκεκριμένο θέμα (topic). Έτσι, ούτε οι παραλήπτες ξέρουν ποιοι είναι οι αποστολείς. Η χρήση του Publish/Subscribe μοντέλου ενδείκνυται για την επεκτασιμότητα που παρέχει στο σύστημά μας, όπως και για την ανοχή σε αλλαγές του δικτύου μας.



Topics

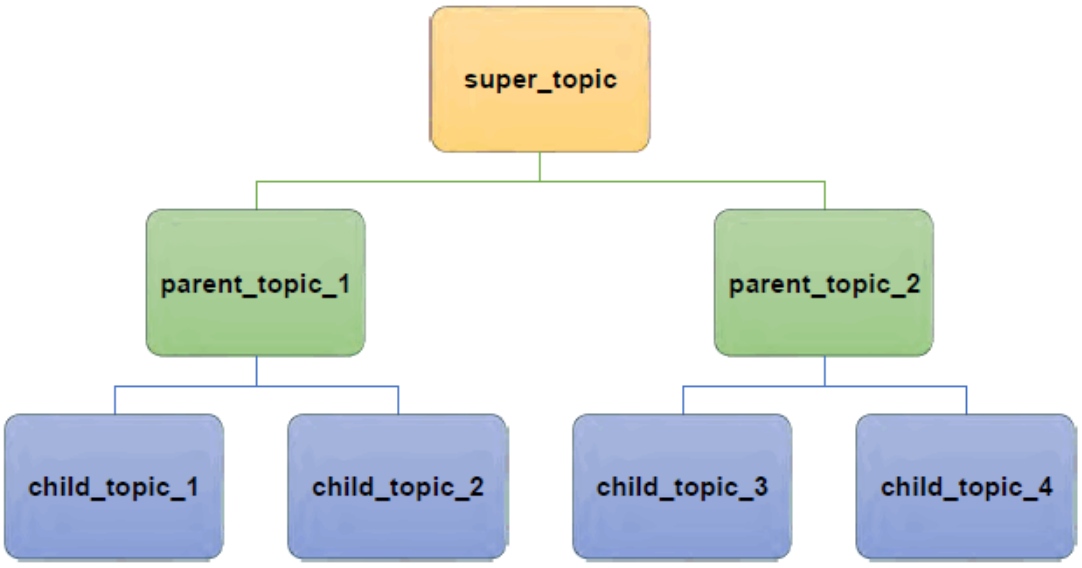
Το topic είναι ένα UTF-8 string που χρησιμοποιείται από τον Broker στο φιλτράρισμα των μηνυμάτων για τους κατάλληλους MQTT Clients. Μπορούμε να έχουμε πολλαπλά θεματικά επίπεδα σε ένα topic για τη διευκόλυνσή μας, με κάθε επίπεδο να ξεχωρίζει στο string από ένα διαχωριστικό «/» .

parent_topic / child_topic

Γενικά είναι καλή πρακτική να ξεκινάμε με ένα super topic «home». Συνίσταται να μην ξεκινάμε ποτέ με διαχωριστική γραμμή γιατί αυτό δημιουργεί ένα super topic χωρίς όνομα [11].

Παρατηρούμε ότι κάθε parent topic μπορεί να έχει πολλαπλά child topics, ανάλογα με τον σχεδιασμό του συστήματός μας. Έτσι δημιουργούμε ένα δέντρο θεμάτων (**topic tree**) το οποίο όμως βρίσκεται μόνο στον σχεδιασμό μας και δεν αποθηκεύεται στο σύστημα (Σχήμα 3.6-1). Ένας Broker βλέπει μόνο το παραπάνω format, δηλαδή ένα string διαχωρισμένο με slashes.

Στο σχήμα δείχνουμε την ιεραρχία των θεμάτων σε ένα δέντρο.



Το super_topic είναι ένα topic που χρησιμοποιούμε για την ολοκληρωμένη ιεράρχηση του δέντρου μας. Δεν το χρησιμοποιούμε αλλά η βιβλιογραφία προτείνει την χρήση του για το scalability του δέντρου μας [13]. Ονόματα που μπορούμε να χρησιμοποιήσουμε για αυτό το topic είναι home ή root ή και τον χώρο στον οποίο βρίσκεται η εφαρμογή μας όπως university, factory. Για να αναφερθούμε στα 4 child topics γράψουμε [13]:

super_topic / parent_topic_1 / child_topic_1

super_topic / parent_topic_1 / child_topic_2

super_topic / parent_topic_2 / child_topic_3

super_topic / parent_topic_2 / child_topic_4

Πολλές φορές θέλουμε να αναφερθούμε σε πολλαπλά topics. Το MQTT μας δίνει την δυνατότητα να χρησιμοποιήσουμε τρία σύμβολα (+, #, $), τα οποία στην βιβλιογραφία ονομάζονται **wildcards** [11,12] προκειμένου να αναφερθούμε σε πολλά topic με μία έκφραση. Το σύμβολο της πρόσθεσης **+** λέγεται **μονοεπίπεδο wildcard** γιατί μας επιτρέπει να αναφερθούμε σε ένα ολόκληρο θεματικό επίπεδο. Για παράδειγμα:

super_topic / + / child_topic_1

Στο παραπάνω παράδειγμα αναφερόμαστε σε όλα τα parent_topics των θεμάτων με όνομα child_topic_1, δηλαδή στα:

super_topic / parent_topic_1 / child_topic_1

super_topic / parent_topic_2 / child_topic_1

super_topic / parent_topic_3 / child_topic_1

...

Επίσης, το σύμβολο της δίεσης # λέγεται **πολυεπίπεδο wildcard** και μας δίνει την δυνατότητα να αναφερθούμε σε όλα τα επίπεδα που βρίσκονται στο ίδιο θεματικό επίπεδο και χαμηλότερα, για παράδειγμα:

super_topic / #

Σύμφωνα με το topic tree της εικόνας αναφερόμαστε στα εξής topics:

super_topic / parent_topic_1 / child_topic_1

super_topic / parent_topic_1 / child_topic_2

super_topic / parent_topic_2 / child_topic_3

super_topic / parent_topic_2 / child_topic_4

Παρατηρούμε ότι με την δίεση αναφερόμαστε σε όλα τα topics που βρίσκονται στο επίπεδο 2 και χαμηλότερα [12].

Εδώ σημειώνουμε επίσης την ύπαρξη του συμβόλου $ που χρησιμοποιείται μόνο για topics εσωτερικής χρήσης του Broker και debugging και όχι για αποστολή μηνυμάτων [12].

Από τα παραπάνω καταλαβαίνουμε την τεράστια σημασία του σχεδιασμού του topic tree σε ένα MQTT project. Συνήθως χρησιμοποιούμε πρακτικές που διευκολύνουν το σωστό σχεδιασμό. Οι κυριότερες σύμφωνα με τα [12,13] είναι:

- Δεν χρησιμοποιούμε κενά στα ονόματα των θεμάτων γιατί μεγαλώνει η πιθανότητα λάθους στην αναγραφή αυτών στον κώδικά μας.
- Χρησιμοποιούμε σύντομα και περιεκτικά ονόματα για να διευκολύνουμε την ανάγνωσή και κατανόηση τους.
- Δεν ξεχνάμε την πιθανότητα της ανάγκης πρόσθεσης νέων topic στο δέντρο. Άρα, σχεδιάζουμε πάντα με αυτό υπόψιν. Η χρήση super topic όπως πχ. home/ μας διευκολύνει σε αυτό και γ' αυτό την προτιμούμε.
- Συνδυάζουμε τα αντικείμενα του project με τις θεματικές ενότητες (επίπεδα).

super_topic / device_1 / sensor_1

super_topic / device_1 / sensor_2

...

- Συνδυάζουμε λειτουργείες των αντικειμένων [11].

super_topic / door_1 / open

super_topic / door_1 / close

...

Διαθήκη Client και επιμονή μηνυμάτων

Σε περίπτωση αποσύνδεσης ενός Client μπορούμε να έχουμε ορίσει από πριν ένα μήνυμα που θα σταλεί σε αυτή την περίπτωση. Αυτό μας διευκολύνει στο να εντοπίσουμε πότε και πού έγινε η αποσύνδεση αλλά και να φροντίσουμε την συνέχεια της λειτουργείας του συστήματος μετά την αποσύνδεση. Επίσης μπορούμε να διευκολύνουμε την αυτόματη επανασύνδεση και συνέχιση του αποσυνδεδεμένου Client [7,11].

Ο Broker έχει την ευθύνη να αποσυνδέει αυτομάτως όλους τους Clients μετά από μια περίοδο απραξίας. Ο Client έχει την ευθύνη να καταγράφει τον χρόνο από την τελευταία φορά που αλληλεπίδρασε με τον Broker. Όταν ξεπεραστεί ένα όριο χρόνου που ορίζουμε εμείς, τότε ο Client στέλνει PINGREQ στον Broker (ο οποίος απαντά με PINGRESP) για να αποφύγει την αυτόματη αποσύνδεσή του από τον Broker. Στην περίπτωση αποσύνδεσης τότε ο Broker στέλνει το προκαθορισμένο μήνυμα διαθήκης του Client [7,11].

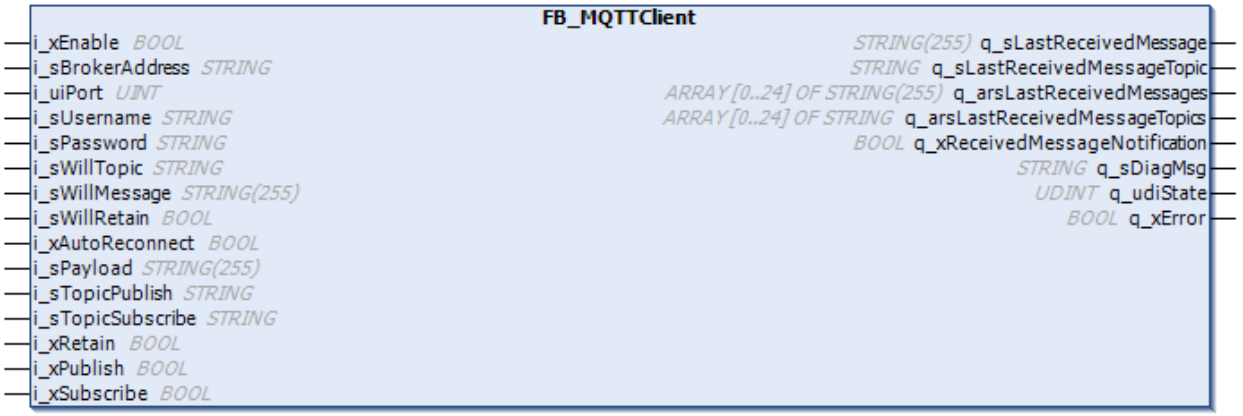
Παράλληλα, μπορούμε να διασφαλίσουμε ότι μηνύματα με αποδέκτη τον αποσυνδεδεμένο Client θα γίνουν αποδεκτά ετεροχρονισμένα με την επανασύνδεση του. Όπως αναφέρθηκε παραπάνω, με το flag RETAIN ένα μήνυμα

χαρακτηρίζεται ως επίμονο και αποθηκεύεται στον Broker μέχρι να ληφθεί από όλους τους αποδέκτες ακόμη και αν αυτοί είναι αποσυνδεδεμένοι. Επομένως, με την επανασύνδεση ενός Client μπορεί να λάβει μηνύματα που είχαν αποσταλεί όσο ήταν εκτός λειτουργίας. Αυτή η λειτουργία μπορεί να χρησιμοποιηθεί και για την αρχικοποίηση ενός νέου Client μέσω μηνυμάτων [7,11].

Υλοποίηση MQTT στο SIMATIC IOT2020

Η υλοποίηση του πρωτοκόλλου MQTT (όπως και του MODBUS RTU) έγινε με τη χρήση της πλατφόρμας Codesys.Η Rossmann Engineering έχει δημιουργήσει για το Codesys μια βιβλιοθήκη η οποία υλοποιεί το MQTT με σκοπό τη δημοσίευση των μεταβλητών ενός PLC σε έναν MQTT Broker.

Για να το επιτύχουμε αυτό είναι ανάγκη η χρήση ενός μόνο Function Block (FB) προγραμματισμένο σε γλώσσα του προτύπου IEC 61131-3.



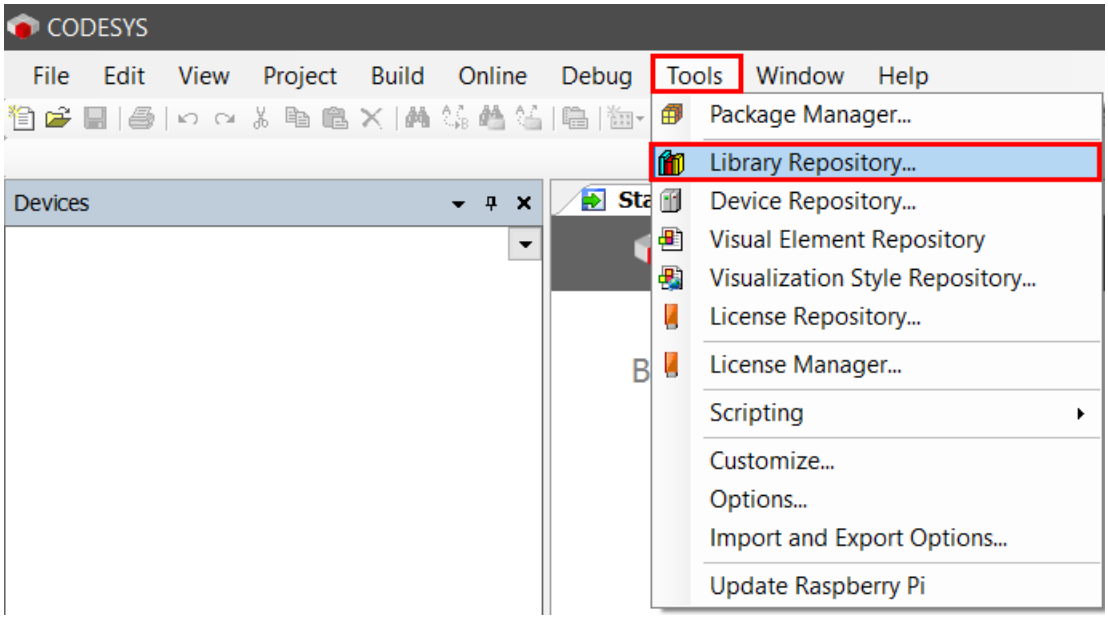
Το οποίο έχει τις παρακάτω παραμέτρους εισόδου-εξόδου :

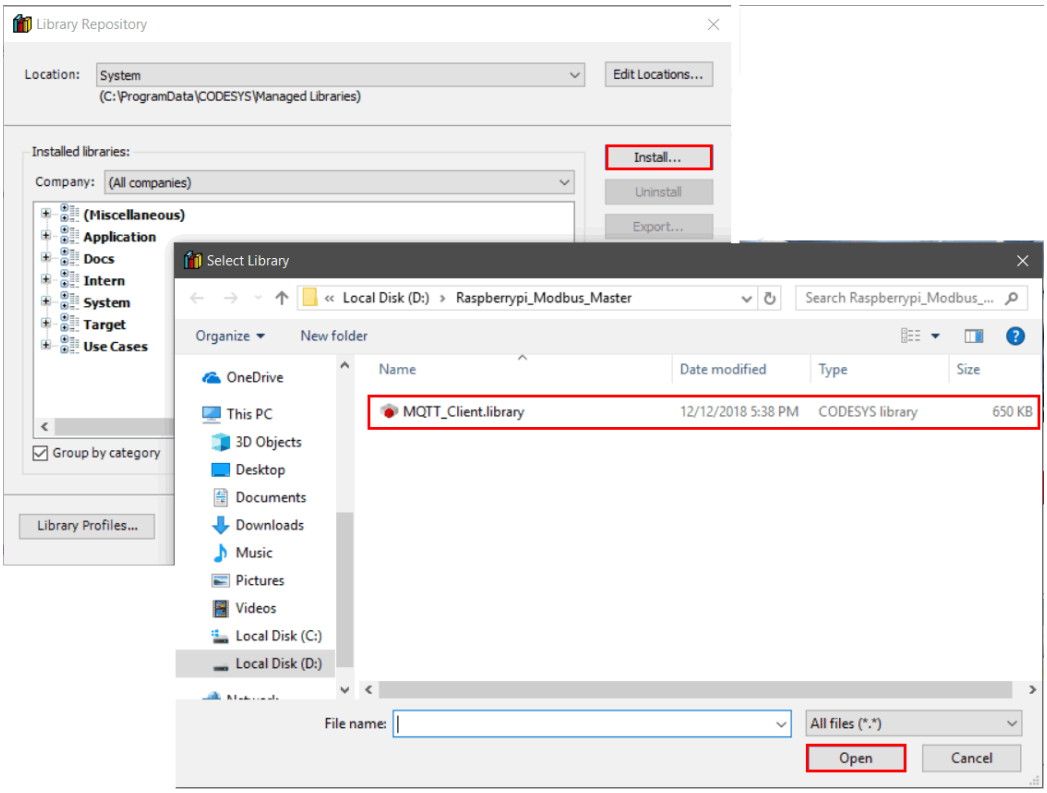
Όνομα	Τύπος δεδομένων	Αρχική Τιμή	Σχόλιο
i_xEnable	BOOL	TRUE	Ενεργοποίηση του Function Block. Όταν η κατάσταση της εισόδου i_xEnable είναι 1 τότε η συσκευή συνδέεται στον MQTT Broker
i_sBrokerAddress	STRING	'www.mqtt-dashboard.com'	IP Διεύθυνση (Παράδειγμα: 192.168.178.101) ή Διεύθυνση Ιστοσελίδας (www.mqtt-dashboard.com) of the MQTT-Broker

i_uiPort	UINT	1883	Θύρα (Port) του MQTT-Broker
i_sUsername	STRING	''	(Προαιρετικό) Username (Αλλάζουμε μόνο την αρχική τιμή αν απαιτείται)
i_sUsername	STRING	''	(Προαιρετικό) Password (Αλλάζουμε μόνο την αρχική τιμή αν απαιτείται)
i_sWillTopic	STRING	''	(Προαιρετικό) Το θέμα του μηνύματος διαθήκης (Το αφήνουμε ανέγγιχτο αν δεν απαιτείται)
i_sWillMessage	STRING	''	(Προαιρετικό) Μήνυμα Διαθήκης (Το αφήνουμε ανέγγιχτο αν δεν απαιτείται)
i_sWillRetain	BOOL	FALSE	Διατήρηση τελευταίου μηνύματος διαθήκης, αν απαιτείται Standard: FALSE – αλλιώς το αφήνουμε ανέγγιχτο
i_xAutoReconnect	BOOL	TRUE	Αυτόματη προσπάθεια επανασύνδεσης μετά από μια εξαίρεση
i_sPayload	STRING	'Hello MQTT-Broker from CoDeSys'	Το Payload προς δημοσίευση (Publish)
i_sTopicPublish	STRING	'CoDeSys'	Θέμα του μηνύματος προς δημοσίευση
i_sTopicSubscribe	STRING	'CoDeSys'	Θέμα συνδρομής
i_xRetain	BOOL	TRUE	Retain Flag
i_xPublish	BOOL	FALSE	Δημοσίευση του μηνύματος στον MQTT Broker
i_xSubscribe	BOOL	FALSE	Συνδρομή στο Θέμα Συνδρομής (που έχουμε

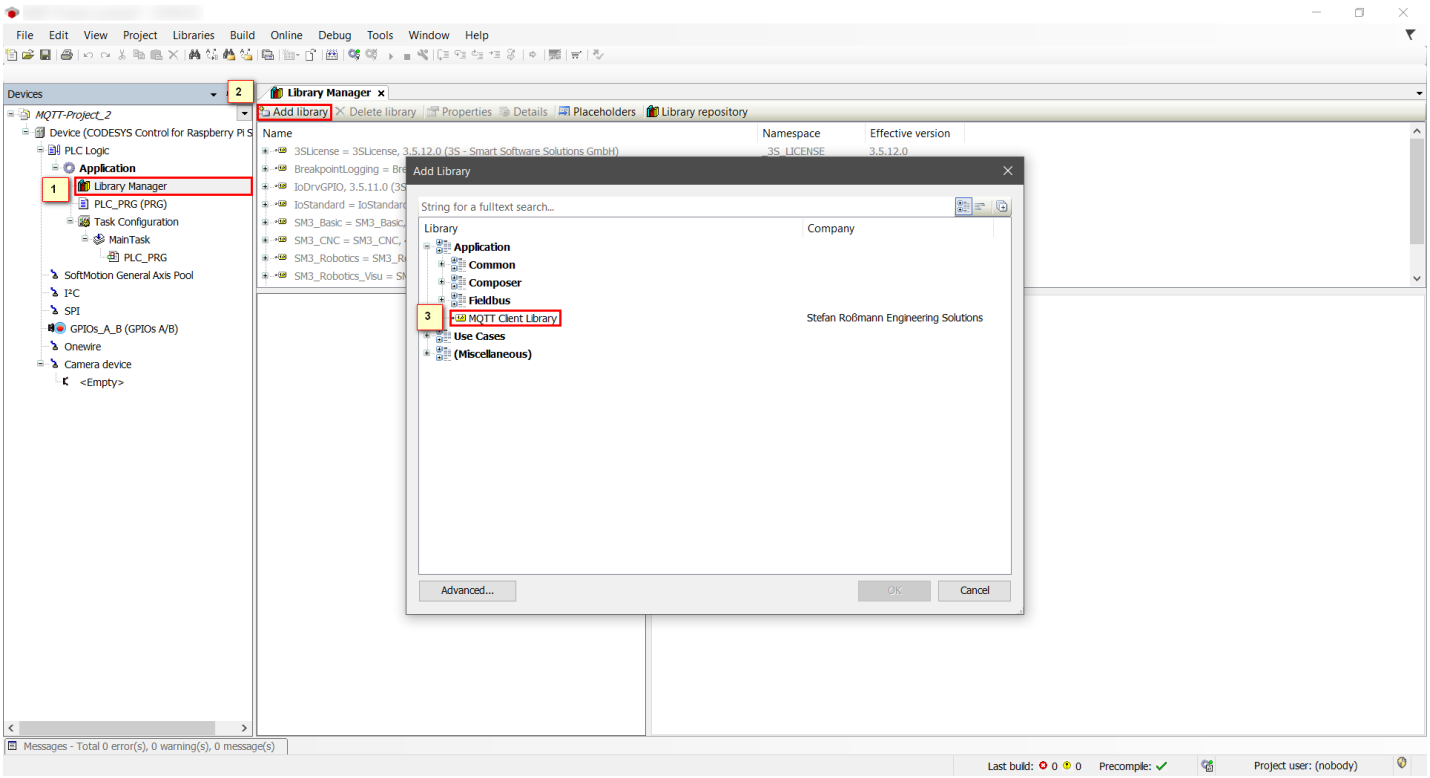
			εισάγει στην είσοδο i_sTopicSubscribe)
q_stLastReceivedMessage	STRING		Μήνυμα που ελήφθη από ένα θέμα
q_stLastReceivedMessageTopic	STRING		Τελευταίο θέμα μηνύματος που ελήφθη.
q_arsLastReceivedMessages	ARRAY[0..24] OF STRING		Τελευταία 25 ληφθέντα μηνύματα
q_arsLastReceivedMessagesTopic	ARRAY[0..24] OF STRING		Τελευταία 25 ληφθέντα θέματα
q_xReceivedMessageNotification	BOOL		Ειδοποίηση λήψης του μυνήματος
q_sDiagMsg	String		Diag Message – Τωρινή Κατάσταση
q_udiState	UDINT		Τωρινή κατάσταση του Function block
q_xError	BOOL		Error Flag - Η Εξαίρεση θα αναγνωριστεί με παλμική άνοδο στη μεταβλητή i_xEnable

Αρχικά εγκαταστήσαμε την παραπάνω βιβλιοθήκη στο Codesys και συγκεκριμένα στο Library Repository.

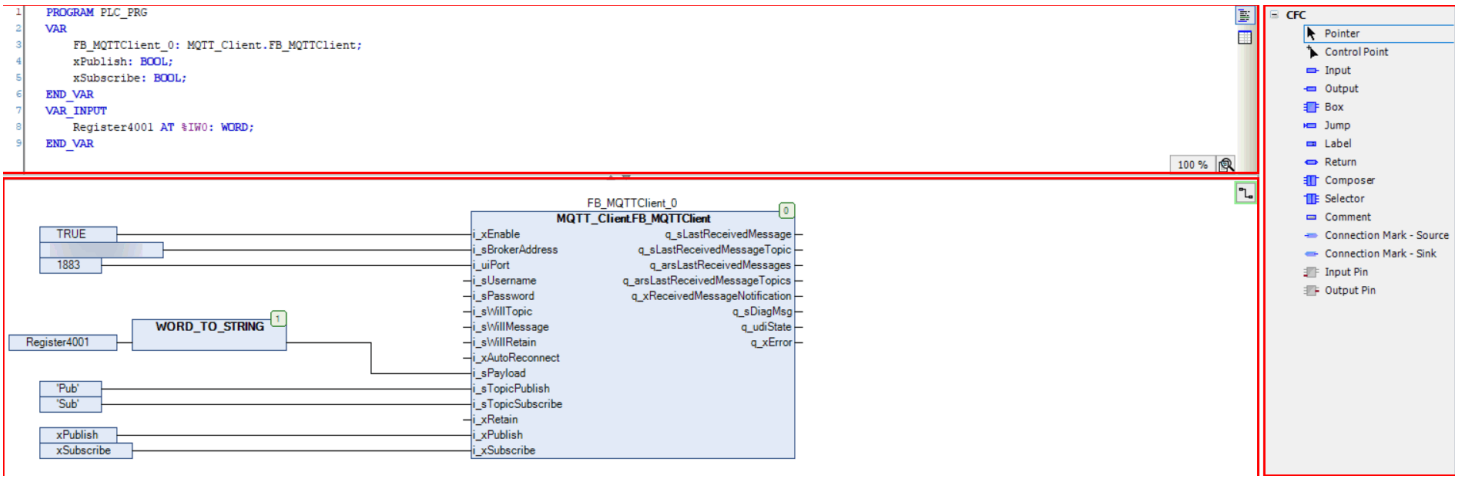




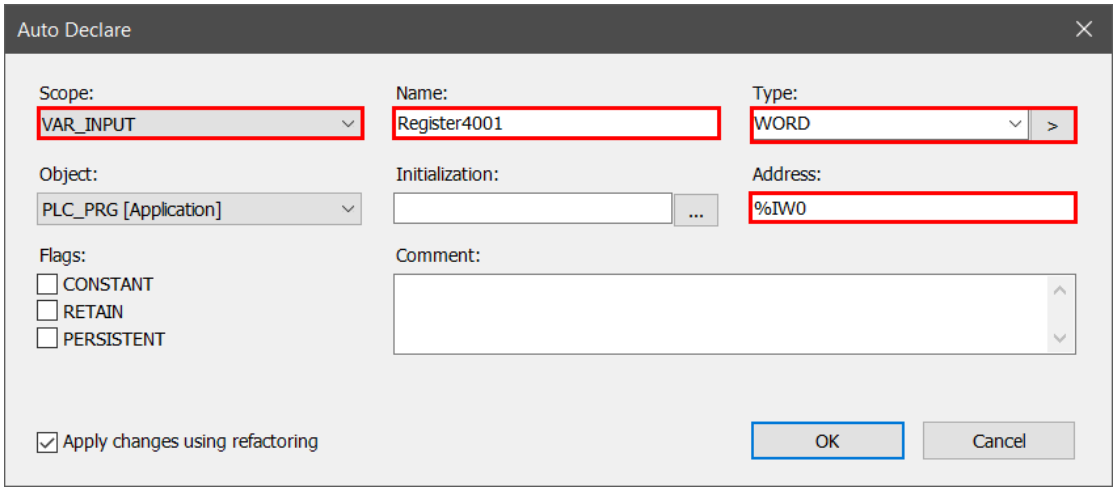
Αφ' ότου εγκαταστήσαμε την βιβλιοθήκη, δημιουργήσαμε ένα καινούργιο Standard Project σε γλώσσα **CFC** (**Continuous Function Chart**). Ύστερα, εισήγαμε τη βιβλιοθήκη στο project σύμφωνα με την παρακάτω εικόνα:



Στη συνέχεια χρησιμοποιώντας το Menu που βρίσκεται στα δεξιά ,εισήγαμε το απαιτούμενο Function Block (MQTT_ClientFB_MQTTClient) και ορίσαμε τις παραμέτρους εισόδου αυτού.



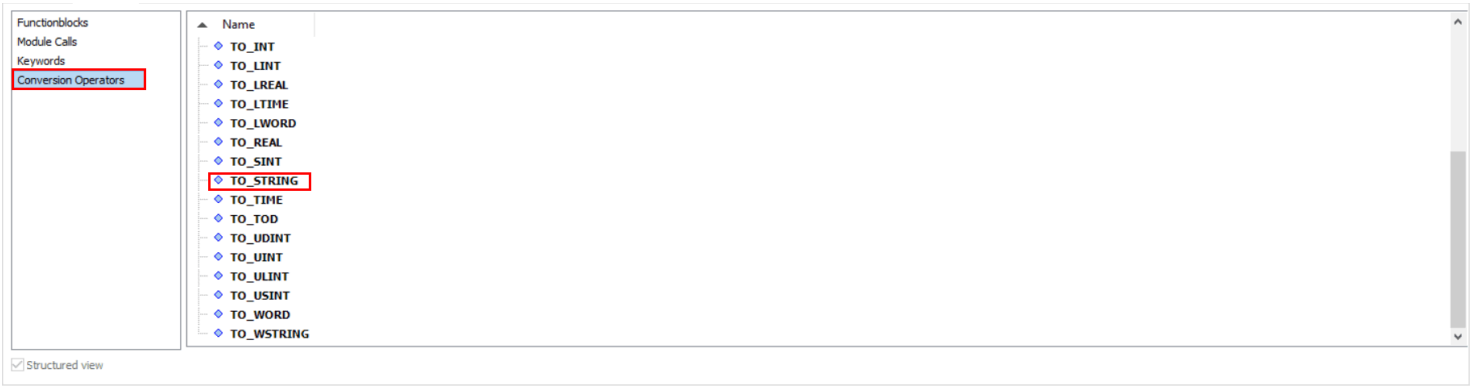
Σύμφωνα με την παραπάνω εικόνα, αρχικά ορίσαμε την παράμετρο i_xEnable μόνιμα στην κατάσταση 1 (TRUE) ούτως ώστε το Gateway να συνδέεται απευθείας στον MQTT Broker κατά την εκτέλεση του προγράμματος. Ακόμη ορίσαμε την IP διεύθυνση του MQTT Broker (i_xBrokerAddress) αλλά και την θύρα (port) που χρησιμοποιεί το πρωτόκολλο MQTT (1883). Τέλος δημιουργήσαμε δύο λογικές (Boolean) μεταβλητές (x_Publish και x_Subscribe) για την δημοσίευση και την συνδρομή μηνυμάτων ενώ παράλληλα ορίσαμε και τα θέματα δημοσίευσης και συνδρομής (Pub και Sub αντίστοιχα). Το μόνο που μας έμενε ήταν να ορίσουμε το περιεχόμενο του μηνύματος. Αυτό που επιθυμούμε είναι το Gateway να διαβάζει δεδομένα από το PLC μέσω του πρωτοκόλλου MODBUS RTU. Συνεπώς έπρεπε να ορίσουμε μια είσοδο η οποία θα έχει ως διεύθυνση τη διεύθυνση του δεδομένου που θέλουμε να διαβάσουμε από το PLC . Αυτό γίνεται εισάγοντας αρχικά από το menu μια είσοδο (Input) και στη συνέχεια με διπλό κλικ πάνω στο τετράγωνο πλαίσιο ,εμφανίζεται το παρακάτω παράθυρο .



Τα δεδομένα από το PLC θα πρέπει να οριστούν στο πρόγραμμα ως **VAR_INPUT** ενώ ο τύπος τους είναι **WORD**. Στο πλαίσιο Address γράφουμε την

διεύθυνση της μεταβλητής που θέλουμε να διαβάσουμε ενώ στο πλαίσιο Name δίνουμε μια ονομασία για αυτήν.

Όμως απ' ό,τι παρατηρούμε, τα δεδομένα του PLC είναι τύπου WORD ενώ το Function Block δέχεται ως είσοδο μόνο δεδομένα τύπου STRING. Άρα το μόνο που μας έμενε ήταν να μετατρέψουμε τα δεδομένα που λαμβάναμε μέσω του MODBUS RTU σε δεδομένα τύπου STRING. Το Codesys μας δίνει την δυνατότητα να χρησιμοποιήσουμε έτοιμα blocks – μετατροπείς με την βοήθεια των οποίων μπορούμε να μετατρέψουμε τα δεδομένα μας σε οποιοδήποτε τύπο επιθυμούμε.

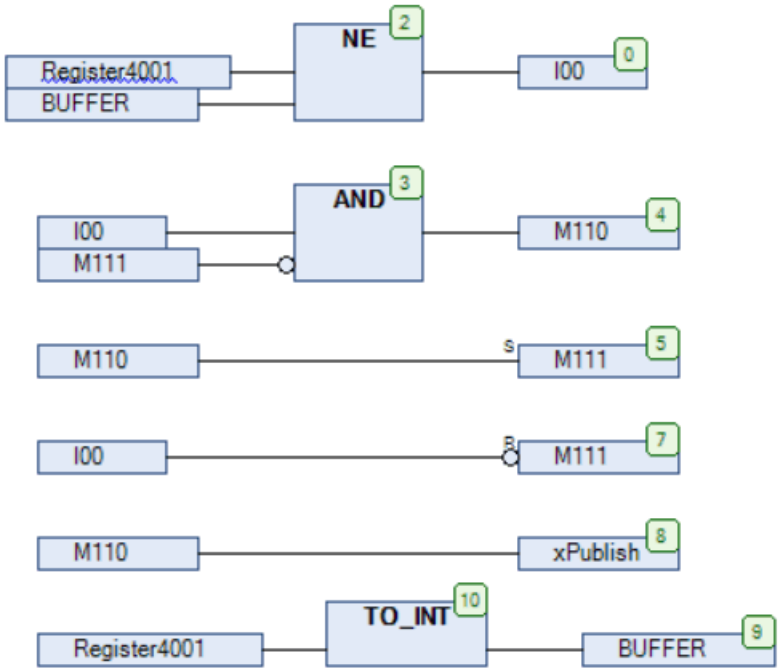


Για να μειώσουμε την διακίνηση των δεδομένων στο δίκτυο (network traffic), γράψαμε ένα μικρό κομμάτι κώδικα με τον οποίο κάθε MQTT Client (στη προκειμένη περίπτωση κάθε PLC) δημοσιεύει δεδομένα μόνο όταν υπάρχει αλλαγή τιμής σε αυτά . Με αυτό το τρόπο το κάθε PLC δέχεται μια νέα τιμή μόνο όταν είναι απαραίτητο βελτιώνοντας παράλληλα και την απόδοση του δικτύου.

```
STL
      1.  L   buffer
      2.  L   Register4001
      3.  <>
      4.  =   I0.0
      5.  A   I0.0
      6.  AN  M11.1
      7.  =   M11.0
      8.  A   M11.0
      9.  S   M11.1
      10. AN  I0.0
      11. R   M11.1
      12. A   M11.0
      13. =   xPublish
      14. L   Register4001
      15. T   buffer
```

Στις πρώτες τέσσερις γραμμές του κώδικα ,εξετάζουμε αν η τιμή που διαβάζουμε από το PLC (μέσω του MODBUS RTU) είναι διαφορετική από την τιμή του buffer. Θα πρέπει σε αυτό το σημείο να επισημάνουμε ότι έχουμε αρχικοποιήσει την μεταβλητή buffer με την τιμή -1. Ο λόγος που επιλέχθηκε αυτή η τιμή είναι ότι δεν υπάρχει στη διάταξη μας κάποιο στοιχείο (είσοδοι, έξοδοι και αισθητήρες) που μπορεί να πάρει αρνητικές τιμές. Αν οι τιμές των μεταβλητών Register4001 και buffer είναι διαφορετικές τότε ενεργοποιείται η είσοδος I0.0. Από την εντολή 5 έως την εντολή 11 δημιουργούμε έναν κρουστικό παλμό καθώς για την δημοσίευση ενός μηνύματος από τον MQTT Client (PLC) χρειαζόμαστε παλμική άνοδο του RLO (Result of Logic Operation) . Έτσι όταν έρχεται νέα τιμή στη μεταβλητή Register4001 ο Client κάνει Publish αυτή στο άλλο PLC. Τέλος ενημερώνουμε την τιμή του buffer με την τωρινή τιμή του καταχωρητή.

Η γραφή του παραπάνω προγράμματος έγινε στην πλατφόρμα Codesys χρησιμοποιώντας την γλώσσα CFC .



Η ορθή λειτουργία ολόκληρου του προγράμματος (τόσο της υλοποίησης του MQTT όσο και της δημοσίευσης μηνυμάτων μόνο σε αλλαγές τιμής των δεδομένων) εξετάστηκε με τη χρήση του προγράμματος MQTT.fx το οποίο μπορεί να λειτουργήσει είτε ως MQTT Publisher είτε ως MQTT Subscriber.