Erstellung von Bibliotheken in CoDeSys V3

Dokument Version 2.0

INHALT

1 Einführung

Das Bibliothekskonzept in CoDeSys V3 unterscheidet sich grundlegend von dem in CoDeSys V2. Daher sind bei der Erstellung und Verwendung von Bibliotheken einige Punkte zu beachten.

Das Bibliothekskonzept in V3 hat folgende Eigenschaften:

Jede Bibliothek wird eindeutig durch ihren Namen, Ihre Version und Ihren Hersteller identifiziert.

Jede Bibliothek hat einen Namespace über den auf die Elemente (Bausteine und Variablen) der Bibliothek zugegriffen werden kann.

Die Version einer Bibliothek besteht aus 4 Stellen, wobei jede Stelle eine bestimmte Bedeutung besitzt (siehe Kapitel 2.3)

Bibliotheken werden in CoDeSys in einer Datenbank (genannt: Repository) abgelegt.

Für bessere Übersichtlichkeit im Repository kann jede Bibliothek einer *Kategorie* (Category) zugeordnet werden (siehe Kapitel 2.1).

Eine Bibliothek kann eine andere Bibliothek als sogenannte Kind-Bibliothek einbinden, wenn sie auf der Kind-Bibliothek aufbaut. Dann wird beim Laden der Bibliothek immer auch Ihre Kind-Bibliothek geladen.

Die Version einer Bibliothek die eingebunden wurde kann durch unterschiedliche Verfahren festgelegt werden (siehe Kapitel 2.2).

Bibliotheken können Funktionen und Bausteine enthalten, die im Laufzeitsystem in ANSI-C/C++ implementiert wurden (so genannte *„externe Bibliotheken“*). Siehe Kapitel 2.5.

Bibliotheken werden auch von CoDeSys zum Zwecke der Codegenerierung (z.B. Visualisierung, Symbolkonfiguration oder IO-Treiber in IEC) automatisch eingebunden (so genannte *„Implizite Bibliotheken“*). Diese werden im Bibliotheksverwalter grau hinterlegt und können nicht geändert werden. Siehe Kapitel 2.6.

Jede freigegebene Bibliothek kann intern durch ein Freigabe-Flag markiert werden. Danach wird bei jedem Versuch die Library zu ändern eine Warnung generiert. Siehe Kapitel 2.8.

Es sollte darauf geachtet werden, dass Bibliotheken nach der Definition von Schnittstellen und deren Implementierung getrennt werden sollten. Siehe Kapitel 2.7.

2 Bibliothekskonzept im Detail

Nachfolgend werden die wichtigsten Aspekte bei der Erstellung und Verwendung von Bibliotheken in CoDeSys V3 näher betrachtet.

2.1 Kategorien von Bibliotheken

Für bessere Übersichtlichkeit im Repository kann jede Bibliothek einer *Kategorie* zugeordnet werden. Jeder Kunde kann dabei eigene Kategorien erstellen.

Aktuell existieren folgende Kategorien:

2.1.1 System

Allgemeine für den Betrieb des Laufzeitsystemkomponenten notwendigen Bibliotheken, welche automatisch angezogen werden.

2.1.2 Internal

Bibliotheken welche von bestimmten PugIn‘s oder DeviceDescriptions benötigt werden und automatisch angezogen werden.

2.1.3 Application

Für den Programmierer hilfreiche Bibliotheken, welche dieser selbst je nach Bedarf in sein Projekt einfügen kann.

2.1.4 Target

Kategorie, welche spezielle hersteller- und zielsystemabhängige Bibliotheken gruppiert.

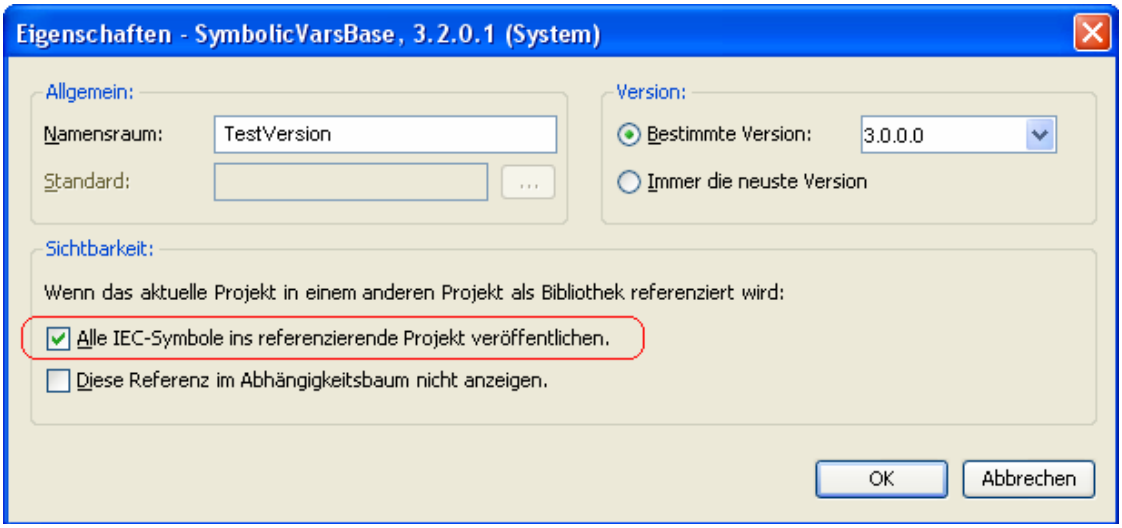
2.2 Namespaces

Jeder Toplevel-Bibliothek kann im Bibliotheksmanager ein Namespace zugewiesen werden. Der Default-Namespace wird aus dem Property 'DefaultNamespace' der Bibliothek übernommen. Ist dieses Property nicht gesetzt, so wird er aus Firmennamen und Bibliotheksnamen abgeleitet.

Namespaces müssen den Regeln für IEC-Identifier genügen, daher werden alle Zeichen, welche nicht Bestandteil eines IEC-Identifiers sein dürfen, durch Unterstriche '_' ersetzt. Außerdem wird ein Unterstrich '_' vorangestellt, sollte der Default-Namespace mit einer Ziffer beginnen. Kann ein Symbol vom Compiler nicht eindeutig aufgelöst werden, so erzeugt dieser einen Fehler. Die Eindeutigkeit wird dabei durch die Eindeutigkeit des Zugriffpfades <Namespace>.<Symbol> bestimmt. Bibliotheks-Namespaces können auch mehrstufig sein, d.h. bei geschachtelten Bibliotheken kann über eine Schachtelung der Namespaces auf die „unterste“ Bibliothek zugegriffen werden.

Beispiel:
Bibliothek Test mit Namespace Test referenziert die Bibliothek Test2 mit Namespace Test2. Beide enthalten eine Funktion aFUN. In einem Projekt wird die Verwendung des Symbols aFUN nur dann zu einer Fehlermeldung führen, wenn die Bibliothek Test die Symbole von Test2 auch veröffentlicht.

Das geschieht durch die folgende Einstellung in den Eigenschaften der Bibliotheksreferenz von Test2:



Wenn diese Einstellung getätigt wurde, dann wird die Verwendung des Symbols aFUN ohne Namespace zu einer Fehlermeldung führen, da das Symbol nicht eindeutig aufgelöst werden kann. Die Funktion aFUN aus Test kann mittels Test.aFUN verwendet werden. Die Funktion aFUN aus Test2 kann mittels Test.Test2.aFUN und mit Test2.aFUN referenziert werden, da dieser Namespace auch zu den veröffentlichten IEC Symbolen gehört.

Wenn die Symbole der Bibliothek nicht von der darüberliegenden Bibliothek veröffentlicht werden, dann können die Symbole von Test2 dennoch über eine Schachtelung der Namespaces angesprochen werden.

Die Namespaces von referenzierten Bibliotheken können nur in der sie verwendenden Bibliothek geändert werden, nicht jedoch im Projekt. Im Projekt können nur die Namespaces von Bibliotheken geändert werden, welche in diesem toplevel eingefügt sind.

2.3 Versionierung von Bibliotheken

Grundsätzlich erhält jede Bibliothek bei der Freigabe eine eindeutige Version. Die Strategie für Vergabe der Version wir nachfolgend erklärt:

Bei **kompatibler Änderung** im Sinne einer Erweiterung oder Fehlerbehebung wird die letzte Stelle (patch version) der Versionsnummer hoch gezählt.

Bei **inkompatibler Änderung** im Sinne einer Erweiterung: vorletzte Stelle (service pack version) der Versionsnummer hoch zählen.

Bei Inkompatibler Änderung im Sinne von Schnittstellenänderung: zweite Stelle (sub version) der Versionsnummer hoch zählen.

Damit ergibt sich:
V3. <sub version>. <service pack version>. <patch version>

2.4 Verfahren für das Einbinden einer Bibliotheksversion

Eine Bibliothek wird durch ihren Namen, ihre Version und ihren optionalen Namespace eindeutig identifiziert. Bei der Version gibt es nun mehrere Möglichkeiten hier beim Einbinden Einfluss zu nehmen (so genannte „Version Constraints“).

Die Version einer Bibliothek kann durch folgende unterschiedliche Bedingungen eingeschränkt bzw. erweitert werden.

2.4.1 Bestimmte Version (Specific Version)

Es wird hier immer die exakte Version einer Bibliothek geladen, z.B. 3.2.0.1. Das dient dazu, dass auch bei Vorhandensein neuerer und älterer Versionen derselben Bibliothek immer nur genau diese Version geladen und verwendet wird. Damit ist auch nach dem Laden eines Projektes keine neuer Compile-Vorgang notwendig, da sich die Bibliothek ja nicht geändert hat.

2.4.2 Immer neueste Version (Newest Version)

Hier wird immer die neueste Version einer Bibliothek verwendet.

Es sind zum Beispiel die Versionen 3.1.3.1, 3.1.3.2, 3.2.0.0 und 3.2.0.1 installiert. Dann wird immer 3.2.0.1 eingebunden, auch wenn zum Zeitpunkt der Einbindung der Bibliothek nur die Version 3.1.3.1 als neueste Bibliothek existierte.

Dieses Verfahren ist für Schnittstellen-Bibliotheken zwingend vorgeschrieben! Dies dient dazu, dass sich alle verwendeten Bibliotheken (die diese Schnittstelle implementieren) auf die neueste Version einigen und nur diese eine Version geladen wird. Ansonsten könnten bei gemischten Versionen Compilefehler auftreten, da dieselbe Schnittstelle dann in unterschiedlichen Versionen vorliegt und diese damit inkompatibel sind. Daher ist auch die Trennung von Schnittstellen und deren Implementierung in unterschiedliche Bibliotheken aufzuteilen (siehe Kapitel 2.7)!

2.4.3 Platzhalter

In der Gerätebeschreibung einer Steuerung kann eine Bibliothek als Platzhalter definiert werden. Wenn man nun in einem Projekt oder Bibliothek diesen Platzhalter einbindet, dann wird die eigentliche Version aus der Gerätebeschreibung entnommen. Damit wird also immer nur eine bestimmte Version einer Bibliothek verwendet, die die Gerätebeschreibung vorgibt.

WICHTIG:
Alle Bibliotheken, die externe Implementierungen enthalten (siehe Kapitel 2.5), müssen grundsätzlich über Platzhalter eingebunden werden, damit die Bibliothek immer mit der Implementierung im Laufzeitsystem übereinstimmt!

2.5 Externe Bibliotheken

Bibliotheken, die im Laufzeitsystem in ANSI-C implementiert sind, werden als externe Bibliotheken bezeichnet. Wobei in CoDeSys V3 nicht mehr zwischen internen und externen Bibliotheken unterschieden wird. (Weitere Informationen hierzu für OEMs: Literaturverzeichnis [1], Kapitel 9.4.)

Dafür kann in den Objekteigenschaften eines Objekts jedes Objekt einzeln als intern oder extern deklariert werden. Für externe Objekte wird pro extern zu linkender Funktion ein Eintrag in der Liste der externen Referenzen erzeugt. Dieser Eintrag enthält den Namen der Funktion ohne Namespace, die Adresse des Funktionszeigers sowie einen Checkcode über die Schnittstelle (CRC der Signatur). Im LZS implementierte Funktionen müssen also

eindeutige Namen haben! Damit kann also jede Funktion oder jeder Baustein in einer Bibliothek einzeln im Laufzeitsystem implementiert werden.

Der Name in der Bibliothek und der Name im Laufzeitsystem müssen nicht notwendigerweise gleich sein. Typischerweise ist er das, es kann aber zum Beispiel sein, dass der Name einer externen Funktion mit einem IEC-Bezeichner kollidiert. Dann kann man die externe Funktion mit dem Attribut „external_name“ dekorieren:

```
{attribute 'external_name' := 'name_on_rts'}
FUNCTION ExtFun : BOOL
VAR_INPUT
END_VAR
VAR
END_VAR
```

In diesem Fall wird die Funktion nicht mit einer Funktion „ExtFun“ auf dem LZS identifiziert sondern mit einer Funktion „name_on_rts“.

Da es im Laufzeitsystem nur eine Implementierung einer Funktion oder eines Bausteins gibt, darf nur eine Version einer Bibliothek mit genau dieser Implementierung im Projekt verwendet werden! Das wird typischerweise über das Platzhalter-Konzept ermöglicht (siehe 2.4.3). Daher muss eine externe Bibliothek typischerweise in der Gerätebeschreibung hinterlegt werden.

Dies kann entfallen, wenn sich an der Schnittstelle einer Bibliothek in den verschiedenen Versionen keine Änderung ergeben hat.

2.5.1 Versionsprüfung von Bibliotheken gegenüber dem Laufzeitsystem

Jede Bibliotheks-Funktion bzw. eine Bibliotheks-Methode eines Bausteins, die im Laufzeitsystem implementiert ist, werden im Laufzeitsystem auf Versionskompatibilität geprüft. Jede Funktion bzw. Methode eines Baustein erbt dabei die Version ihrer Bibliothek.

Wenn nun die ersten beiden Stellen dieser vierstelligen Version nicht übereinstimmen, dann wird der Download abgelehnt und eine entsprechende Meldung in CoDeSys erzeugt. Daher müssen hier die Version der (externen) Bibliothek und die Implementierung im Laufzeitsystem in den ersten beiden Stellen zusammenpassen.

2.5.2 Signaturprüfung von Bibliotheken gegenüber dem Laufzeitsystem

Jede Bibliotheks-Funktion bzw. eine Bibliotheks-Methode eines Bausteins, die im Laufzeitsystem implementiert ist, besitzt eine definierte Schnittstelle. Da im Laufzeitsystem aber nur der Name der Funktion bzw. der Methode geprüft wird, können sich hier schwerwiegende Probleme bis zum Absturz ergeben, wenn diese Schnittstellen nicht zusammenpassen.

Aus diesem Grund wird zu jeder Funktion bzw. Methode eine Checksumme über die Schnittstelle mit zum Laufzeitsystem übertragen. Gegen diese CRC kann dann das Laufzeitsystem die entsprechende Implementierung prüfen.

Die Checksumme der Schnittstelle einer Funktion bzw. Methode beinhaltet Datentyp des Rückgabewerts sowie Namen und Datentypen der Parameter.

Da die Berechnung der Checksumme nur in CoDeSys erfolgen kann, gibt es in CoDeSys eine Möglichkeit alle extern implementierten Funktionen bzw. Methoden einer Bibliothek automatisch in eine Datei für das Laufzeitsystem exportieren zu können. Näheres dazu im folgenden Abschnitt.

2.5.3 Export in m4-Dateien

In CoDeSys V3 gibt es die Möglichkeit alle extern implementierten Funktionen bzw. Methoden einer Bibliothek automatisch in eine Datei für das Laufzeitsystem zu exportieren. Damit sind Inkompatibilitäten zwischen der Bibliothek und der Implementierung im Laufzeitsystem nahezu ausgeschlossen.

In dieser erzeugten Datei (genannt m4-Datei) befinden sich alle extern implementierten Funktionen und Methoden mit ihrem Namen (entweder wie in CoDeSys definiert oder über das Attribut „external_name“ festgelegt), ihrer Schnittstelle, der Version der Bibliothek und der Checksumme der Schnittstelle. Diese Datei muss dann mittels einem m4-Compiler zu einer ANSI-C Schnittstellen Header Datei übersetzt werden. Zusätzlich besteht die Möglichkeit in CoDeSys auch einen C-Source Rahmen für die Implementierung erzeugen zu lassen.

Die Schnittstellen Header Datei (<Bibliotheksnamen>Itf.h) und die Source Datei (<Bibliotheksnamen>.c) können dann im Laufzeitsystem als Basis für die Implementierung einer Komponente verwendet werden, die die Implementierung der externen Funktionen bzw. Bausteine enthält. Diese Komponente kann dann ins Laufzeitsystem eingebunden werden.

2.6 Implizit geladene Bibliotheken

Interne Komponenten von CoDeSys, die IEC-Code erzeugen (z.B. Symbolkonfiguration oder Visualisierung) benötigen dafür auch sehr häufig Elemente aus Bibliotheken. Diese Bibliotheken werden daher von CoDeSys in jedem Projekt automatisch d.h. implizit eingebunden. Diese Bibliotheken werden grau im Bibliotheksverwalter angezeigt.

Bibliotheken können aber auch implizit aus einer Gerätebeschreibung angezogen werden. Dies wird zum Beispiel für IO-Geräte verwendet, die beim Einbinden auch eine Bibliothek mitbringen. Beim Einbinden wird dann meist auch eine Instanz eines IO-Treibers aus dieser Bibliothek erzeugt.

2.7 Trennung von Schnittstelle und Implementierung

Es sollte bei der Erstellung von Bibliotheken darauf geachtet werden, dass ein Interface, das in mehreren Bibliotheken implementiert wird, in eine separate Bibliothek verlagert wird.

Dann können alle Bibliotheken, die ein gemeinsames Interface implementieren, diese Bibliothek mit Newest Constraint einbinden (siehe Kapitel 2.4.2). Ansonsten können Übersetzungsfehler die Folge sein, wenn in einem Projekt zwei unterschiedliche Versionen dieser Schnittstellenbibliothek eingebunden sind, denn der CoDeSys Compiler erkennt hier eine Inkompatibilität einer Schnittstelle, die in zwei unterschiedlichen Versionen vorliegt.

2.8 Freigabe von Bibliotheken

In der Projektinformation einer Bibliothek dient ein Freigabeflag dazu, eine freigegebene Bibliothek zu erkennen. Diese Information kann auch automatisiert ausgelesen werden (z.B. Rep-Tool).

Nach dem Setzen dieses Flags wird bei jedem Versuch die Library zu ändern eine Warnung generiert.

Literaturverzeichnis

[1] CoDeSys SP 3.x Runtime System Overview.pdf (OEM-Dokumentation)

Änderungshistorie

Version	Beschreibung	Datum
0.1	Erstellung	23.04.2008
0.2	Review und entspr. Überarbeitung Aus Punkt 1 in der Einführung Punkte 1 und 2 gemacht. Kapitel 2.2 umgeschrieben.	02.05.2008
0.3	Review und entspr. Überarbeitung In Kapitel 2.5 zu den externen Funktionen noch eine Bemerkung zum Attribut „external_name“ zugefügt.	25.06.2008
0.4	Formaler Review und entspr. Überarbeitung (u.a. dts.Abb)	25.06.2008
1.0	Freigabe	25.06.2008
1.1	2.3 und 2.4.1 angepasst	15.07.2008
1.2	Review -> In Ordnung	21.08.2008
2.0	Freigabe nach formalem Review	21.08.2008