bachmann.
SolutionCenter

C / C++ Developer

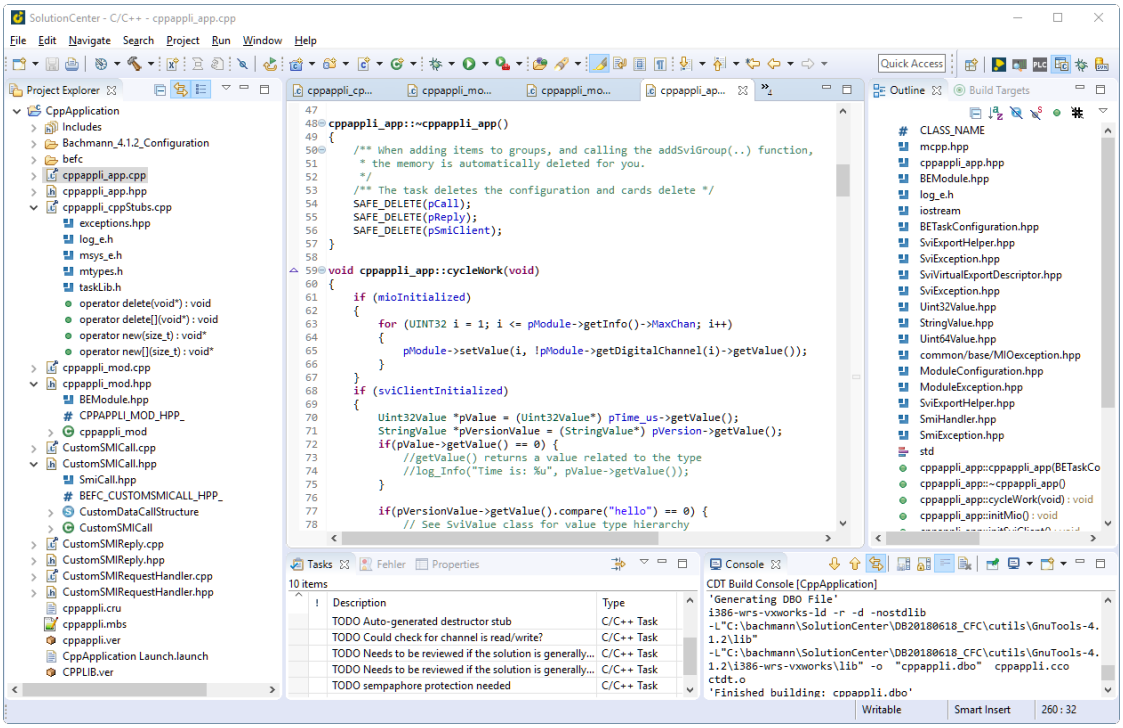
Moderne Automatisierungssysteme lösen komplexe Aufgabenstellungen, die weit über das reine Steuern einer Anlage hinausgehen. Die prozedurale oder objektorientierte Programmierung in den Sprachen C bzw. C++ bietet die Möglichkeit verschiedenste Aufgabenstellungen effizient und strukturiert zu lösen. Dabei unterstützt der C/C++ Developer den Anlagenprogrammierer in allen Phasen der Software-Entwicklung. Die systemnahe Programmierung ermöglicht eine hohe Flexibilität bei höchster Performance.

Synergien im SolutionCenter

Durch die Integration des C/C++ Developer in das SolutionCenter können Synergien genutzt und damit Mehrwerte geschaffen werden.

Features

- C/C++ Projekte können durch Verlinken in ein Solution-Projekt mit Device-Konfigurationen und Vorlagen zu einer Gesamtlösung verbunden und gemeinsam verwaltet werden.
- Die Bearbeitung der C/C++ Projekte erfolgt in einer vordefinierten Perspektive, deren Editoren und Ansichten den Bedürfnissen des Anwenders entsprechend angeordnet werden können.
- Die integrierte Versionsverwaltung mittels GIT oder SVN ermöglicht dem Benutzer die einfache Nachverfolgbarkeit der Code-Änderungen und stellt die Basis für ein Release-Management sowie die Zusammenarbeit in Teams dar.
- Die Verwendung des integrierten Scripting-Frameworks oder andere Erweiterungen in Form von Plug-ins, die auch direkt aus dem Eclipse Marketplace installiert werden können, erleichtern die Projektierung und Programmierung.
- Durch die Verwendung der SolutionCenter Scripting-API können moderne Software-Entwicklungsmethoden wie Continuous Integration, Model-Driven-Development und Test-Driven-Development zum Einsatz gebracht werden.
- Vordefinierte Projekt-Vorlagen ermöglichen einen schnellen Einstieg in die Applikationsentwicklung. Dabei können einzelne Features ausgewählt werden, welche dann gleich kompiliert, auf die Steuerung übertragen und ausgeführt werden können.


SolutionCenter - C/C++ - cppappli_app.cpp
File Edit Navigate Search Project Run Window Help
Project Explorer
CppApplication
Includes
Bachmann_4.1.2_Configuration
befc
cppappli_app.cpp
cppappli_app.hpp
cppappli_cppStubs.cpp
exceptions.hpp
log_e.h
msys_e.h
mtypes.h
taskLib.h
operator delete(void*) : void
operator delete[](void*) : void
operator new(size_t) : void*
operator new[](size_t) : void*
cppappli_mod.cpp
cppappli_mod.hpp
BEModule.hpp
CPPAPPLI_MOD_HPP_
cppappli_mod
CustomSMICall.cpp
CustomSMICall.hpp
SmiCall.hpp
BEFC_CUSTOMSMICALL_HPP_
CustomDataCallStructure
CustomSMICall
CustomSMIReply.cpp
CustomSMIReply.hpp
CustomSMIRequestHandler.cpp
CustomSMIRequestHandler.hpp
cppappli.cru
cppappli.mbs
cppappli.ver
CppApplication Launch.launch
CPPLIB.ver
cppappli_cp...
cppappli_mo...
cppappli_mo...
cppappli_ap...
cppappli_app::~cppappli_app()
{
/** When adding items to groups, and calling the addSviGroup(..) function,
* the memory is automatically deleted for you.
*/
/** The task deletes the configuration and cards delete */
SAFE_DELETE(pCall);
SAFE_DELETE(pReply);
SAFE_DELETE(pSmiClient);
}
void cppappli_app::cycleWork(void)
{
if (mioInitialized)
{
for (UINT32 i = 1; i <= pModule->getInfo()->MaxChan; i++)
{
pModule->setValue(i, !pModule->getDigitalChannel(i)->getValue());
}
}
if (sviClientInitialized)
{
Uint32Value *pValue = (Uint32Value*) pTime_us->getValue();
StringValue *pVersionValue = (StringValue*) pVersion->getValue();
if(pValue->getValue() == 0) {
//getValue() returns a value related to the type
//log_Info("Time is: %u", pValue->getValue());
}
if(pVersionValue->getValue().compare("hello") == 0) {
// See SviValue class for value type hierarchy
Outline
Build Targets
CLASS_NAME
mcpp.hpp
cppappli_app.hpp
BEModule.hpp
log_e.h
iostream
BETaskConfiguration.hpp
SviExportHelper.hpp
SviException.hpp
SviVirtualExportDescriptor.hpp
SviException.hpp
Uint32Value.hpp
StringValue.hpp
Uint64Value.hpp
common/base/MIOexception.hpp
ModuleConfiguration.hpp
ModuleException.hpp
SviExportHelper.hpp
SmiHandler.hpp
SmiException.hpp
std
cppappli_app::cppappli_app(BETaskCo
cppappli_app::~cppappli_app()
cppappli_app::cycleWork(void) : void
cppappli_app::initMio() : void
Tasks
Fehler
Properties
10 items
Description
Type
TODO Auto-generated destructor stub
C/C++ Task
TODO Could check for channel is read/write?
C/C++ Task
TODO Needs to be reviewed if the solution is generally...
C/C++ Task
TODO Needs to be reviewed if the solution is generally...
C/C++ Task
TODO sempaphore protection needed
C/C++ Task
Console
CDT Build Console [CppApplication]
'Generating DBO File'
i386-wrs-vxworks-ld -r -d -nostdlib
-L"C:\bachmann\SolutionCenter\DB20180618_CFC\cutils\GnuTools-4.1.2\lib"
-L"C:\bachmann\SolutionCenter\DB20180618_CFC\cutils\GnuTools-4.1.2\i386-wrs-vxworks\lib" -o "cppappli.dbo" cppappli.cco ctdt.o
'Finished building: cppappli.dbo'
Writable
Smart Insert
260 : 32

Effizientes Programmieren und Verwalten

Die übersichtlichen und intuitiven Editoren erlauben eine moderne und effiziente Arbeitsweise.

- Dabei erhöht das benutzerdefinierte Syntax-Highlighting die Lesbarkeit von Code der Sprachen C und C++.
- Durch die durchgängig verfügbare Code-Navigation und Aufrufhierarchie können auch in komplexen Anwendungen Zusammenhänge leicht erkannt werden.
- Eine effiziente Programmierung wird insbesondere durch die kontextabhängige Autoverollständigung und die vordefinierten und erweiterbaren Code- und Dateivorlagen begünstigt.
- Durch die Verwaltung der Projekte in einem Arbeitsbereich werden alle Änderungen an den Quelldateien in einem lokalen Protokoll gespeichert. Diese können zu einem späteren Zeitpunkt wieder abgerufen werden, unabhängig davon, ob das Projekt versionsverwaltet ist.
- Direkt bei der Eingabe wird der Programmierer durch eine umfangreiche Validierung auf Programmierfehler sofort hingewiesen.
- Informationen zu Schnittstellen und Dokumentation von Klassen, Funktionen oder Variablen können, ohne den Editor zu verlassen, über Tooltipps abgerufen und angezeigt werden.
- Das projektweite Ändern von Bezeichnern ist einfach und durchgängig möglich, genauso wie die Formatierung des Quellcodes.

Modernes Engineering

Der C/C++ Developer ist offen, flexibel und transparent gestaltet, um das Engineering auf eine neue Ebene zu bringen.

- Alle Teile des Quellcodes werden in Textform im Projekt gespeichert, was dem Benutzer die Möglichkeit der Generierung von ganzen Quelldateien und Konfigurationen eröffnet. Wiederkehrende Arbeitsschritte können so automatisiert und Copy & Paste-Fehler vermieden werden.
- Die Strukturierung der Programmteile ist frei wählbar und ermöglicht somit einen modularen Aufbau des Anwendungsprogramms.
- Die Programmierung erfolgt unabhängig von der Zielplattform, welche erst zum Ausführungszeitpunkt ausgewählt werden muss. Nach dem Kompilieren kann das erstellte Anwendungsprogramm direkt auf eine M200-Steuerung übertragen und ausgeführt werden.
- Für Aufgaben können benutzerdefinierte Tags gesetzt werden, um die offenen Punkte besser im Blick zu behalten.

Höhere Qualität durch einfache Fehlersuche

Gerade bei der Entwicklung neuer Anwendungsprogramme und bei einer Fehlersuche ist das Debug-Framework des C/C++ Developer äußerst hilfreich.

- Mittels Breakpoint kann ein Anwendungsprogramm an einer beliebigen Stelle angehalten und in der Folge schrittweise ausgeführt werden. Breakpoints können gemeinsam verwaltet und gesamthaft aktiviert bzw. deaktiviert werden.
- Breakpoints können mit Bedingungen versehen werden, um das Anwendungsprogramm unter bestimmten Umständen anhalten zu können.
- Bei angehaltenem Anwendungsprogramm wird der Stack-Frame angezeigt, über welchen auch zu den aufrufenden Funktionen navigiert werden kann.
- Das gleichzeitige Debuggen mehrerer Anwendungsprogramme und mehrerer Tasks ist möglich.
- Die Disassembly-Ansicht erlaubt das Überwachen und Debuggen im Assembler-Code. Gleichzeitig wird auch der Quellcode dargestellt.

C/C++ Developer

Allgemein	
Integration	• Verwaltung mehrerer C/C++ Projekte in einem Arbeitsbereich • Verwaltung aller Devices und C/C++ Projekte in einem Arbeitsbereich • Verlinkung von C/C++ Projekten in Solution-Projekte für die gemeinsame Verwaltung
Darstellung	• Vordefinierte Perspektive mit Ansichten zur Bearbeitung von C/C++ Projekten • Übersichtliche, frei wählbare Anordnung der Ansichten und Editoren
Versionsverwaltung	• SVN (Subversion) • GIT (lokal und remote)
Automatisierung	• Ausführen von Skripten im SolutionCenter (JavaScript, Python) • Vordefinierte Skriptfunktionen für das Erstellen und Übertragen von C/C++ Anwendungsprogrammen
Erweiterbarkeit	• Erweiterungen (Plug-ins) über Eclipse Marketplace installierbar • Verwendung selbst erstellter Eclipse-Plug-ins
Skalierbarkeit	• Erstellen des Anwendungsprogramms erfolgt unabhängig von der Zielplattform • Übertragung und Ausführung auf allen M200-CPUs
Struktur	• Zusammenfassen von Projekten in Arbeitssets • Ordnerstruktur im Projekt frei wählbar • Voller Zugriff auf alle Quellcode-Dateien
Vorlagen	• Vordefinierte Projektvorlagen mit funktionsfähigen Code-Snippets • Vorlagen für PLC-Bibliotheken • Vorlagen für C-Komponenten • Benutzerdefinierte Projektvorlagen
Editoren	
Sprachen	• C nach IEC 9899:1999 (C99) • C++ nach IEC 14882:2011 (C++11) • C++ nach IEC 14882:2014 (C++14)
Code-Navigation	• Referenzen finden (im Projekt) • Aufruf-Hierarchie • Aufrufer-Hierarchie (Hierarchieebenen erweiterbar) • Deklaration öffnen
Autovervollständigung	• Kontextabhängig (Deklaration, Methoden) • Vordefinierte Quellcode-Vorlagen • Benutzerdefinierte Quellcode-Vorlagen
Syntax Highlighting	• Vordefiniert • Frei konfigurierbar
Änderungsverfolgung	• Vergleich mit lokal gespeicherten Versionen (lokales Protokoll) • Vergleich mit Quellcode aus SVN oder GIT
Änderungen	• Projektweites Ändern von Bezeichnern • Formatieren des Quellcodes
Informationen	• Kontextabhängige Informationen im Tooltipp auf Bezeichnern
Validierung	• Beschreibung der Systemfunktionen in der Dokumentation • Darstellung der Probleme direkt im Quellcode und in der Fehler-Ansicht • Syntaktische und semantische Prüfung des Quellcodes • Statische Code-Analyse
Aufgaben	• Dokumentation offener Punkte in Kommentaren • Darstellung aller Aufgaben in eigener Ansicht • Definition benutzerdefinierte Tags

Konfiguration	
Modulkonfiguration	• Definition über Konfigurationsbeschreibung (CRU) • Editor für das Bearbeiten der Modulkonfiguration
Build-Konfiguration	• Definition mehrerer Build-Konfigurationen in einem Projekt • Gleichzeitiges Erstellen mit mehreren Build-Konfigurationen
Kompilieren	
Toolchain	• GCC 4.1.2 (bereits enthalten) • GCC 5.5 (bereits enthalten) • MinGW • Cygwin
Build	• Manuell • Automatisch • Pre-/Post-Build-Steps definierbar • Paralleler Build auf mehreren Prozessorkernen
Online gehen	
Installation	• Zielsteuerung permanent oder dynamisch wählbar • Installation auf Zielsteuerung temporär oder permanent • Name der Modulinstanz frei wählbar • Verbinden auf laufendes Anwendungsprogramm (Attach)
Initialisierung	• Definierter Debug-Start möglich
Deploy-Konfigurationen	• Mehrere Konfigurationen pro Projekt möglich • Kombinierte Ausführung von mehreren Konfigurationen möglich (Launch Groups) • Verwaltung der Favoriten
Debuggen - Fehlersuche	
Schrittweise Abarbeitung	• Anhalten des Anwendungsprogramms mittels Breakpoint • Schritt hinein/über/zurück • Bedingte Breakpoints • Assembler-Stepping
Überwachen	• Variablenwerte (bei angehaltenem Anwendungsprogramm) • Disassembly-Ansicht mit Assembler-Instruktionen und Quellcode
Darstellung	• Umschaltung des Formats von INT-Werten (dec, hex, bin, oct) • Aktueller Stack-Frame (bei angehaltenem Anwendungsprogramm)
Manipulieren	• Setzen von Variablenwerten