

**Tutorial**

**Verwendung von Makros und der**

**Web-Service-Schnittstelle**

**im Offboard Diagnostic Information System**

**Engineering**

|  |  |
| --- | --- |
| Erstellt durch: | Offboard Diagnostic Information System -Entwicklungsteam |

Änderungshistorie

| **Dokumenten- Version** | **API-Version** | **Datum** | **Autor** | **Beschreibung der Änderungen** |
| --- | --- | --- | --- | --- |
| 0.1 | 0.1 | 02.12.2010 | T-Systems International GmbH | Ersterstellung |
| 0.5 | 0.5 | 15.12.2010 | T-Systems International GmbH | Beschreibung der Rückgabeobjekte hinzugefügt. |
| 0.7 | 0.7 | 06.01.2011 | T-Systems International GmbH | Beschreibungen im Kapitel 5 hinzugefügt und Formatierungen angepasst. |
| 0.8 | 0.8 | 06.01.2011 | T-Systems International GmbH | Kleine Korrekturen sowie Kapitel Versionierung |
| 0.8 | 0.8 | 10.01.2011 | T-Systems International GmbH | Rückgabetypen der Schnittstellenmethoden hinzugefügt. |
| 0.9 | 0.9 | 14.01.2011 | T-Systems International GmbH | sendRawServiceFunctional |
| 0.9 | 0.9 | 08.02.2011 | T-Systems International GmbH | DiagResultStatusImpl und DiagResultEventMemoryEntryImpl um eine Methode erweitert. |
| 0.91 | 0.91 | 10.03.2011 | T-Systems International GmbH | Beschreibung von Speicherzellen lesen / schreiben hinzugefügt |
| 0.92 | 0.92 | 14.03.2011 | T-Systems International GmbH | Rückgabestrukturen für negative Antworten hinzugefügt,  DiagResultImpl und DiagResultNegativeResponseImpl.  Rückgabestruktur zum Lesen von Speicherzellen ergänzt, DiagResultMemoryCellsImpl |
| 2.1.0 | 2.1.0 | 18.03.2011 | T-Systems International GmbH | Anhebung der Version auf 2.1.0 um den Bezug auf die Version der WSDL herzustellen.  2.1 Methoden der Schnittstellen – writeByteCoding  und 4.2.8 DiagResultCodingImpl  Änderung der Ein- und Ausgabparameter für Binärkodierung auf Datentyp String, damit die kurze Kodierung KWP als Integer-Wert übertragen wird. |
| 2.2.0 | 2.2.0 | 01.04.2011 | T-Systems International GmbH | 2.2 Schnittstellenerweiterung um die Methode ‚configureSetting’ zum generischen Konfigurieren des Testers. |
| 2.2.1 | 2.2.1 | 13.04.2011 | T-Systems International GmbH | 3.2.2 Anpassung der festgelegten Admin-Einstellungen |
| 2.2.2 | 2.2.2 | 19.04.2011 | T-Systems International GmbH | Beschreibung DiagResultTableImpl |
| 2.2.3 | 2.2.3 | 27.04.2011 | T-Systems International GmbH | Abbildungsdatei der Namen von Identifikationsdaten |
| 2.4.0 | 2.4.0 | 29.04.2011 | T-Systems International GmbH | Entfernt: Kapitel 3.7 Testbarkeit  Hinzugefügt: Kapitel 3.7 Zustandsdiagramm  Kapitel 1: Hinweis auf Referenzierte Version eingefügt. |
| 2.4.1 | 2.4.1 | 12.05.2011 | T-Systems International GmbH | Starten als Webservice-Server über Startparameter (Kapitel 5.2.1 Start Offboard Diagnostic Information System Engineering als Web-Service Server) |
| 2.4.2 | 2.4.2 | 10.06.2011 | T-Systems International GmbH | Erweiterung Rückgabestruktur getDiagnosticProtocol um KWP2000 |
| 2.4.3 | 2.4.3 | 17.06.2011 | T-Systems International GmbH | Ergänzungen exit Befehl |
| 2.5.0 | 2.5.0 | 05.07.2011 | T-Systems International GmbH | Kapitel 1: Anhebung der referenzierten Schnittstellenversion  Kapitel 2.1 Erweiterungen / Änderungen – Neuerstellung  Kapitel 4.1 Anpassung der Dokumentation für connectToEcu, closeConnection und openConnection. |
| 2.5.1 | 2.5.1 | 15.07.2011 | T-Systems International GmbH | Schnittstellenbeschreibung für Funktion echo() ergänzt |
| 2.6.0 | 2.6.0 | 19.08.2011 | T-Systems International GmbH | Kapitel 1: Anhebung der referenzierten Schnittstellenversion  Kapitel 2.1, 2.2, 4.1 Beschreibung der neuen Funktion readIdentificationData  Kapitel 3.2.1 Start Offboard Diagnostic Information System Engineering als Web-Server jetzt auch unter „localhost“ |
| 2.6.1 | 2.6.1 | 10.10.2011 | T-Systems International GmbH | getEcuAddress: Aus der Dokumentation entfernt, da sie nicht angeboten wird  setCommunicationTraceLevel: Argumente mit sprechenden Namen versehen und Beschreibung erweitert  setTraceLevel: Dokumemtation verfeinert |
| 2.6.2 | 2.6.2 | 18.10.2011 | T-Systems International GmbH | Attribut errorId zu DiagException hinzugefügt. Kann z.B. genutzt werden, um einen Verbindungsabbruch zum Fahrzeug zu erkennen. Aufnahme von DiagException in die Dokumentation. |
| 2.7.0 | 2.7.0 | 02.11.2011 | T-Systems International GmbH | Administrationseinstellungen für Bauzustand dokumentiert |
| 2.7.0 | 2.7.0 | 18.11.2011 | T-Systems International GmbH | Neue Funktionen:  getRollModeState(), startRollMode(), stopRollMode(),  getFlashSessions()  Eingefügt:  Kapitel 4.2.28: DiagResultRollModeImpl  Kapitel 4.2.29: DiagResultRollModeEntryImpl  => Kapitel 4.2.28: DiagException ist jetzt 4.2.30 |
| 2.7.0 | 2.7.0 | 21.11.2011 | T-Systems International GmbH | Neue Funktionen:  setCommParameter  resetCommParameter  setBaudrate  createProtocolBZD  createProtocolVDS  getAvailableProtocols  loginService42  sendToService42Carport  logoutService42 |
| 2.7.0 | 2.7.0 | 24.11.2011 | T-Systems International GmbH | Neue Funktion: getProjectEcuAddressList |
| 3.0.0 | 3.0.0 | 09.02.2012 | T-Systems International GmbH | Zusammenführung der Dokumentationen für die Makro- und Web-Service Schnittstelle mit Übernahme der Historie der Web-Service Schnittstelle |
| 3.0.0 | 3.0.0 | 24.01.2012 | T-Systems International GmbH | Neue Funktionen Flashen:  createFlashSessionDescriptor  flashProgrammingParallel  Neue Strukturen / Rückgabestrukturen:  IDiagFlashSessionDescriptor  IDiagResultFlashProgramming  IDiagResultFlashSession |
| 3.0.0 | 3.0.0 | 09.02.2012 | T-Systems International GmbH | Änderungen / Erweiterungen im Rahmen Multilinkfähigkeit:  Erweiterterung configureSetting(String, String) um die neuen Schlüssel Multilink.MaxNumberOfLogicalLinks und Multilink.ConsiderRestrictionsOfVRTVPT zur Konfiguration des Multilinkverhaltens.  Neue Funktionen:  getLockedConnections()  lockConnection(IConnectionHandle)  unlockConnection(IConnectionHandle)  Alle von einem Steuergerät abhängigen Funktionen bekommen nun als ersten Parameter ein IConnectionHandle übergeben, mit dem ein Steuergerät identifiziert wird. Das Handle erhält man aus den Methoden zur Verbindung mit einem Steuergerät (connectToEcu, connectToEcuWithLogicalLink). Damit ändern sich Signaturen von bestehenden Funktionen. Dies betrifft auch die Makroschnittstelle. Zur Migration von bestehenden Makros gibt es ein spezielles Konvertierungsmakro.  Weitere Details sind im Kapitel Multilinkunterstützung beschrieben. |
| 3.0.0 | 3.0.0 | 09.02.2012 | T-Systems International GmbH | Weitere neue Funktionen:  connectToEcuWithLogicalLink |
| 3.0.0 | 3.0.0 | 09.02.2012 | T-Systems International GmbH | Im Rahmen einer Fehlerbehebung wurde das Verhalten des aktuellen Locales auf die Zahlendarstellung geändert. Dies betrifft z.B. bei Ganzzahlen die Tausendertrennzeichen und bei Gleitkommazahlen das Trennzeichen für die Nachkommastellen. Die Änderung wird sich auf den Inhalt der sprachabhängigen Attribute wie z.B. IDiagResultValue. getTranslatedValue() aus. |
| 3.0.0 | 3.0.0 | 20.02.2012 | T-Systems International GmbH | Zusammenführung von Schnittstellenbeschreibung und Makrotutorial |
| 3.0.1 | 3.0.0 | 26.03.2012 | T-Systems International GmbH | Einarbeitung Reviewergebnisse von Thorsten Schröder |
| 3.0.2 | 3.1.0 | 17.04.2012 | T-Systems International GmbH | Entfernung der Konfigurationsmöglichkeit für das Flashen über die Schnittstellenmethode configureSettings (Überprüfung der „expected idents“ für UDS bzw. Software- une Teilenummern für KWP). Diese Konfiguration wird nun direkt über einen weiteren Parameter bei den entsprechenden Flashmethoden durchgeführt.  Die Methode resetEventMemory hat 3 Parameter bekommen, mit denen man das Verhalten steuern kann.  ACHTUNG: Bislang wurden die Werte für die neuen Parameter aus den Admin-Einstellungen genutzt. Nun werden als Defaults fest definierte Werte verwendet, wenn für die Parameter der Wert null übergeben wird. Die Defaults sind in der Beschreibung der einzelnen Funktionen dokumentiert.  Neue Konfigurationsmöglichkeiten für Admin-Einstellungen sind hinzugekommen. Diese sind bei der Methode configureSettings beschrieben.  Ersetzung der Methode setCommunicationTraceLevel durch setCommunicationTrace und Nutzung der Level aus den Admin-Einstellungen, die über configureSettings gesetzt werden. |
| 3.0.3 | 3.1.0 | 19.04.2012 | T-Systems International GmbH | Verfeinerung Dokumentation zu Admin-Einstellungen |
| 3.0.4 | 3.2.0 | 26.04.2012 | T-Systems International GmbH | Schnittstellenanpassungen für DoIP  List<IDoIPVCI> getDoIPVCIs()  List<IDoIPVCI> searchDoIPVCIs(String ipAddress)  setDoIPVehicleProject(IDoIPVCI doIPVCI, String projectName) |
| 3.0.5 | 3.2.0 | 21.05.2012 | T-Systems International GmbH | Neues Kapitel Besonderheiten bei Start als Web-Service Server |
| 3.0.6 | 3.2.0 | 08.03.2013 | T-Systems International GmbH | Neue Schnittstellenfunktionen:   * getClamp15State() * getClamp30State() * checkVciState() |
| 3.0.7 | 3.4.0 | 19.04.2013 | T-Systems International GmbH | Rückgabestruktur [IDiagResultRawService](#r191) hinzugefügt und als Rückabe der Funktion [sendRawServiceFunctional](#r372) verwendet. |
| 4.0.0 | 4.1.0 | 12.09.2013 | T-Systems International GmbH | Schnittstellenerweiterung:   * connectToEcuAndOpenConnection(int address) |
| 4.0.1 | 4.1.0 | 12.09.2013 | T-Systems International GmbH | Abschnitt 2.1.: Erweiterung des Zustandsdiagramms mit num neue Funktion connectToEcuAndOpenConnection |
| 5.0.0 | 5.0 | 05.11.2013 | T-Systems International GmbH | Erweiterung des Kapitels 6.2 |
| 5.0.0 | 5.0 | 19.11.2013 | T-Systems International GmbH | Hinzufügen der Unterkapitel für das Kapitel 6  Erweitern des Kapitels 6.2 |
| 6.0.0 | 6.0 | 10.12.2013 | T-Systems International GmbH | Neue Unterkapitel 5.7: Protokollierung |
| 7.0.0 | 7.0 | 14.03.2013 | T-Systems International GmbH | Schnittstellenerweiterung:   * getCommParam(IConnectionHandle, String name) * getCommParamDefault(String logicalLink,String name) * getCommParamConfig(String logicalLink,String name) * getCommParams(IConnectionHandle handle) * getCommParamsConfig(String logicalLink) * getCommParamsDefault( String logicalLink) * resetCommParameters(String logicalLink) * setCommParameters(String logicalLink, Map<String,String>) |
| 7.0.1 | 7.0 | 05.05.2013 | T-Systems International GmbH | Erweiterung Unterkapitel 5.7: Protokollierung |
| 7.0.2 | 7.0 | 31.07.2014 | T-Systems International GmbH | Erweiterung der Methode configureSetting(String, String) um den neuen Schlüssel DiagnosisData.CustomerSpecificOperationMode |
| 7.0.3 | 7.0 | 26.08.2014 | T-Systems International GmbH | Korrektur der Beschreibung zu configureSetting(String, String) |
| 8.0.0 | 8.0 | 09.10.2014 | T-Systems International GmbH | Schnittstellenerweiterung:   * flashProgrammingSequence(String controlFilePath, Boolean checkExpectedIdents) |
| 8.0.1 | 8.0 | 14.01.2015 | T-Systems International GmbH | Anpassung Unterkapitel 5.7: Protokollierung  Erweiterung Unterkapitel 4.2.3: Verwendung von Java-Libs |
| 9.0.0 | 9.0 | 23.03.2015 | T-Systems International GmbH | Schnittstellenerweiterung für Multilink-Funktionen:   * createMultilinkHandleByAddress * createMultilinkHandleByLogicalLink * createMultilinkMeasurementDescriptor * createMultilinkCodingDescriptor * createMultilinkAdaptationDescriptor * createMultilinkAdaptationValueDescriptor * readMultilinkIdentification * readMultilinkEventMemory * resetMultilinkEventMemory * readMultilinkMeasurement * readMultilinkAdaptation * readMultilinkCoding * writeMultilinkCoding * writeMultilinkAdaptation |
| 9.0.1 | 9.0 | 25.03.2015 | T-Systems International GmbH | Schnittstellenerweiterung für CommParam-Funktionen:   * getCommParamGroup * getCommParamGroupDefault * getCommParamGroupConfig * resetAllCommParameters   Schnittstellenerweiterung für Hex-Service-Funktionen:   * sendRawServiceWithoutConnectionCheck |
| 9.0.2 | 9.0 | 07.04.2015 | T-Systems International GmbH | Schnittstellenanpassung für Gesamtsystem auslesen:   * ‚ProtocolType‘ für BZD-/VDS-Protokolle entfernt * getAvailableProtocols * createProtocolVDS   Schnittstellenerweiterung für Gesamtsystem auslesen:   * getAvailableAdaptationPresets * getAvailableMeasurementPresets |
| 9.0.3 | 9.0 | 08.04.2015 | T-Systems International GmbH | Schnittstellenerweiterung:   * setVehicleProjectAutomatic * setDoIPVehicleProjectAutomatic |
| 10.0.0 | 10.0 | 14.07.2015 | T-Systems International GmbH | Schnittstellenerweiterung:   * readEEPROMWithFixedAddressLength * writeEEPROMWithFixedAddressLength |
| 10.0.1 | 10.1 | 10.09.2015 | T-Systems International GmbH | Ergebnisstruktur ‚IDiagResultAdaptation‘ um RDID erweitert |
| 11.0.1 | 11.1 | 18.05.2015 | T-Systems International GmbH | Schnittstellenerweiterung siehe Abschnitt 6.3:   * getAutomationApiVersion |
| 12.0.1 | 12.0 | 21.06.16 | T-Systems International GmbH | Schnittstellenerweiterung/-anpassung siehe Abschnitt 6.4:   * createProtocolVDSBZD * getAvailableAdditionalPresets * sendProtocolsToCarport * sendProtocolsToFileServer * sendProtocolsToVDS * readEventMemories * resetEventMemories * updateGatewayComponentList |
| 12.2.1 | 12.2 | 12.09.16 | T-Systems International GmbH | Schnittstellenerweiterung siehe Abschnitt 6.5:   * [actuatorDiagnostics2Init](#r422) * [basicSetting2Init](#r477) * IDiagDescriptor2WithParametersAndMeasurements * [createDiagActuator2](#r428) (nur Makro) * [createBasicSetting2](#r473) (nur Makro) |
| 13.1.0 | 13.1 | 17.10.16 | T-Systems International GmbH | Diagnosefunktion „Rollenmodus“ wurde aus Schnittstelle entfernt (siehe Abschnitt 6.6) |
| 14.1.0 | 14.1 | 11.11.16 | T-Systems International GmbH | Schnittstellenerweiterung für Entwickler-Ereignisspeicher siehe Abschnitt 6.7:   * readAvailableDevEventMemorySelection * readDevEventMemory * resetDevEventMemory |
| 15.2.0 | 15.2 | 19.01.2017 | T-Systems International GmbH | Schnittstellenerweiterung zum Senden von Protokollen an Carport siehe Abschnitt 6.8:   * readAvailableCarportBrands * sendProtocolsToFileServer |
| 15.2.1 | 15.2 | 20.01.2017 | T-Systems International GmbH | Schnittstellenerweiterung zum Ermitteln und Exportieren von DTC Snapshot Records siehe Abschnitt 6.8.  Neue Methoden:   * exportEventMemorySnapshots * getNumberOfEventMemorySnapshots * readEventMemorySnapshot * readEventMemorySnapshots   Geänderte Methoden:   * readEventMemoryWithMask |
| 15.3.0 | 15.3 | 21.02.2017 | T-Systems International GmbH | Schnittstellenerweiterung zur Ausführung von Makros. |
| 15.4.0 | 15.4 | 30.03.2017 | T-Systems International GmbH | Neue Methode requestSecAccessASM (Security Access nach SA3G), nur innerhalb von Makros verwendbar. |
| 15.4.1 | 15.4 | 11.05.2017 | T-Systems International GmbH | Beschreibung zum Starten/Ansprechen des Web-Services angepasst. |
| 16.0.0 | 16.0 | 09.06.2017 | T-Systems International GmbH | Schnittstellenerweiterung zum Ermitteln und schreiben der Subsystem-Verbaulisten |
| 16.0.1 | 16.0 | 23.10.2017 | T-Systems International GmbH | Aufruf „requestSecAccessASM“ aus WEB-Schnittstellen-API entfernt. Über Makro weiterhin verfügbar. |
| 17.0.0 | 17.0 | 31.08.2017 | T-Systems International GmbH | Schnittstellenerweiterung zur Verwendung von CAN-FD-Projekten/Fahrzeugen  Neue Methoden:   * setCanFDVehicleProject * setCanFDVehicleProjectAutomatic |
| 18.0.0 | 18.0 | 12.12.2017 | T-Systems International GmbH | Schnittstellenerweiterung für detailliertere Rückgabeergebnisse beim „Datensatz Download“ (CCB\_DS\_1704, siehe Abschnitt 6.12)  Geänderte Methode:   * dataSetDownload |
| 18.0.1 | 18.0 | 18.12.2017 | T-Systems International GmbH | Schnittstellenerweiterung zum Senden von BZD-Protokollen an AVx (CCB\_DS\_1978, siehe Abschnitt 6.12). Neue Methode:   * sendProtocolsToAVx |
| 18.0.2 | 18.0 | 19.12.2017 | T-Systems International GmbH | Schnittstellenerweiterung zum Überprüfen von Flash-Vorbedingungen (CCB\_DS\_2047, siehe Abschnitt 6.12) Neue Methode:   * [checkFlashPreConditions](#r526) |
| 19.0.0 | 19.0 | 23.05.2018 | T-Systems International GmbH | Schnittstellenerweiterung für SFD-Funktionen (CCB\_DS\_2164, siehe Abschnitt 6.13)  Neue Methoden:   * securityAccessSFDOnline * securityAccessSFDOffline * securityAccessSFDEcuLock * securityAccessSFDAllEcuLock * securityAccessSFDReset * securityAccessSFDCheckStatus |
| 19.0.1 | 19.0 | 15.06.2018 | T-Systems International GmbH | Die folgenden Makro-Methoden stehen nun auch an der WEB-API zur Verfügung:   * createDiagActuator * createDiagActuator2 * createBasicSetting * createBasicSetting2 * createDiagMeasurementDescriptor |
| 19.0.2 | 19.1 | 26.10.2018 | T-Systems International GmbH | Neue Methode getEntryType() in IDiagResultComponentListEntry, um zu erkennen ob ein Element der Verbauliste ein Softwarecluster oder Busmasterelement ist (CCB\_DS\_2149). Aufrufe von readSubsystemComponentList() liefern den Typ [SUBSYSTEM\_COMPONENT\_ENTRY](#r129). |
| 19.0.3 | 19.1 | 13.11.2018 | T-Systems International GmbH | Erweiterung der API-Dokumentation um SFD-Parameterwerte für WEB-API und Macro. |
| 19.0.4 | 19.2 | 22.01.2019 | T-Systems International GmbH | Schnittstellenerweiterung für Gesamtsystem auslesen (Neues Kapitel 6.14):   * getAvailableIdentificationPresets * createProtocolVDSBZD * readEventMemories |
| 20.0.0 | 20.0 | 18.02.2019 | T-Systems International GmbH | Schnittstellenerweiterung für die Kurzläufer-Routinen (Neues Kapitel 6.15):   * getAvailableIdentificationPresets * createProtocolVDSBZD |
| 20.0.1 | 20.1 | 05.03.2019 | T-Systems International GmbH | Schnittstellenänderung zum Wechsel schwischen Gateway Verbauliste und Maxverbau (neues Kapitel 6.16, geändertes Kapitel 5.2.2) |
| 20.2.0 | 20.2 | 20.03.2019 | T-Systems International GmbH | Schnittstelenerweiterung um eine Flashreinfolgedatei beim Flashen (neues Kapitel 6.17)   * flashProgrammingCoordinated |
| 21.0.0 | 21.0 | 12.07.2019 | T-Systems International GmbH | Schnittstellenerweiterung für Entwickler-Ereignisspeicher Snapshots (neues Kapitel 6.18.1)   * readDevEventMemory * getNumberOfDevEventMemorySnapshots * readDevEventMemorySnapshots * exportDevEventMemorySnapshots   Schnittstellenerweiterung für Software Composition / Software Components (neues Kapitel 6.18.2)  Identifikation:   * getShortNamesOfSoftwareCompositions * readIdentificationSoftwareCompositions * Erweiterung readIdentification * Erweiterung readMultilinkIdentification   Codierung:   * createSoftwareCompositionComponentListEntry * readSoftwareCompositionComponentList * updateSoftwareCompositionComponentList * writeSoftwareCompositionComponentList * writeSoftwareCompositionComponentListFromPresentState |
| 21.1.0 | 21.1 | 20.08.2019 | T-Systems International GmbH | Schnittstellenerweiterung bzgl. CCB 2312 (Schalten DoIP Activation-Line)   * Neues Kapitel 6.19 |
| 22.0 | 22.0 | 17.10.2019 | T-Systems International GmbH | Schnittstellenerweiterung bzgl. CCB 2269 (SFD-Ende-zu-Ende-Absicherung)   * Neues Kapitel 6.20 |
| 22.0.0 | 22.0 | 12.11.2019 | T-Systems International GmbH | Schnittstellenerweiterung CCB 2248 (ZUG-Update Stufe2)   * Neues Kapitel 6.21 |
| 23.0.0 | 23.0 | 31.01.2019 | T-Systems International GmbH | Schnittstellenerweiterung CCB 2226 (Multilink Funktionen zum Lesen und Schreiben der Komponentenliste)   * Neues Kapitel 6.22 |
| 23.1.0 | 23.1 | 19.05.2020 | T-Systems International GmbH | Schnittstellenerweiterung CCB 2249 (Datensatz Upload)   * Neues Kapitel 6.23 |
| 23.1.1 | 23.1 | 10.07.2020 | T-Systems International GmbH | JIRA ODISE-1272: Nutzung absoluter Pfade für Konfigurationsdatei (Änderung in 5.2.1.1) |
| 24.0.0 | 24.0 | 27.11.2020 | T-Systems International GmbH | Schnittstellenerweiterung CCB 2467 (SFD Anmeldung über Soft-Zertifikat / Freischaltung aller SFD-Steuergeräte)   * Neues Kapitel 6.24 |
| 24.0.0 | 24.0 | 12.01.2021 | T-Systems International GmbH | CCB\_DS\_2511 - Automatische SFD-E2E Freischaltung bei sendRawService   * Geändert Kapitel 6.24 |
| 24.0.0 | 24.0 | 25.01.2021 | T-Systems International GmbH | CCB\_DS\_2458 - Protokolle der Funktion "BZD"   * Geändert Kapitel 6.24 |
| 24.0.0 | 24.0 | 03.02.2021 | T-Systems International GmbH | CCB\_DS\_2517 - Zugriffsberechtigung für Fahrzeuge mit Markenkennung „Ford“   * Geändert Kapitel 6.24 |
| 24.1.0 | 24.1 | 24.02.2021 | T-Systems International GmbH | CCB\_DS\_2371 - Multiple Identifikationsdaten   * Neues Kapitel 6.25 |
| 24.1.1 | 24.1 | 11.03.20211 | T-Systems International GmbH | Vertraulichkeitsstufe „vertraulich“  in der Fußzeile eingefügt |
| 24.3.0 | 24.3 | 04.08.2021 | T-Systems International GmbH | CCB\_2335 - neue Eigendiagnosefunktion „Fahrzeugfunktionen“   * Neues Kapitel 6.26 |
| 24.4.0 | 24.4 | 30.08.2021 | T-Systems International GmbH | CCB\_2528 – Konfiguration des partiellen Flashens   * Neues Kapitel 6.27 |
| 24.5.0 | 24.5 | 18.10.2021 | T-Systems International GmbH | WSDL-Anpassung für Eigendiagnosefunktion „Fahrzeugfunktionen“   * Neues Kapitel 6.28 |
| 25.0.0 | 25.0 | 23.11.2021 | T-Systems International GmbH | CCB 2525 Entfernung Schnittstellenversion aus URL der WebSchnittstelle   * Neues Kapitel 6.29 |
| 25.1.0 | 25.1 | 14.12.2021 | T-Systems International GmbH | CCB 2598 Verwendung Standard-Vehicle-Info   * Neues Kapitel 6.30   Schnittstellenerweiterung:   * getVehicleInfos * setVehicleInfo |
| 25.1.1 | 25.1 | 25.01.2022 | T-Systems International GmbH | CCB 2589 SFD-Freischaltung über TBT-Token   * Erweiterts Kapitel 6.30   Schnittstellenerweiterung:   * securityAccessSFDSetTBTToken (neu) * securityAccessSFDOnline (erweitert) * securityAccessSFDReset (erweitert) * securityAccessSFDAllOnline (erweitert) * configureSetting (erweitert) |
| 25.2.0 | 25.2 | 14.02.2022 | T-Systems International GmbH | CCB 2548 Deaktivierbarkeit der Flashdatensicherheit   * neues Kapitel 6.31 |
| 25.2.1 | 25.2 | 07.04.2022 | T-Systems International GmbH | CCB 2598 Verwendung Standard-Vehicle-Info   * Anpassung Kapitel 6.30 |
| 25.2.2 | 25.2 | 24.06.2022 | T-Systems International GmbH | CCB 2423 Webschnittstelle auf Localhost begrenzen  Anpassung Kapitel 5.2.1 |
| 25.2.3 | 25.2 | 15.08.2022 | T-Systems International GmbH | #22519 Korrektur der Import-Anweisung für Python-Module  Anpassung Kapitel 4.2.2 |
| 26.0.0 | 26.0 | 04.10.2022 | T-Systems International GmbH | #22613 Entfernen von Carport als Sendeziel von BZD Protokollen  Neues Kapitel 6.32 |

[1 Einleitung 15](#_Toc115767106)

[2 Konzeption der Automatisierungsschnittstelle 15](#_Toc115767107)

[2.1 Zustandsdiagramm 15](#_Toc115767108)

[2.2 Richtung / Informationsaustausch 16](#_Toc115767109)

[2.3 Multilinkunterstützung 17](#_Toc115767110)

[2.4 Versionierung 17](#_Toc115767111)

[3 Die Makrofunktion im Offboard Diagnostic Information System Engineering 18](#_Toc115767112)

[3.1 Aufgabe der Makrofunktion im Offboard Diagnostic Information System 18](#_Toc115767113)

[3.2 Aufzeichnen von Makros 18](#_Toc115767114)

[3.3 Ausführen von Makros 19](#_Toc115767115)

[3.4 Debuggen von Makros 21](#_Toc115767116)

[4 Erstellen eigener Makros für das Offboard Diagnostic Information System 23](#_Toc115767117)

[4.1 „Getting Started“ 23](#_Toc115767118)

[4.2 Die Makrosprache Python 23](#_Toc115767119)

[4.3 Verwenden von Offboard Diagnostic Information System Diagnosefunktionen 29](#_Toc115767120)

[4.4 Migrationsunterstützung Makros auf Schnittstellenversion 3.0 37](#_Toc115767121)

[5 Automatisierungsschnittstelle über Web-Service 37](#_Toc115767122)

[5.1 Schnittstellenverfahren 37](#_Toc115767123)

[5.2 Konfiguration 38](#_Toc115767124)

[5.3 Logging 41](#_Toc115767125)

[5.4 Transaktionssicherheit 41](#_Toc115767126)

[5.5 Fehlerbehandlung 42](#_Toc115767127)

[5.6 Security 42](#_Toc115767128)

[5.7 Protokollierung 42](#_Toc115767129)

[5.8 Besonderheiten 43](#_Toc115767130)

[6 API Änderungen Offboard Diagnostic Information System 43](#_Toc115767131)

[6.1 Version 4.0 43](#_Toc115767132)

[6.2 Version 5.0 43](#_Toc115767133)

[6.3 Version 11.1 44](#_Toc115767134)

[6.4 Version 12.0 44](#_Toc115767135)

[6.5 Version 12.2 45](#_Toc115767136)

[6.6 Version 13.1 45](#_Toc115767137)

[6.7 Version 14.1 45](#_Toc115767138)

[6.8 Version 15.2 46](#_Toc115767139)

[6.9 Version 15.3 47](#_Toc115767140)

[6.1 Version 16.0 47](#_Toc115767141)

[6.11 Version 17.0 47](#_Toc115767142)

[6.12 Version 18.0 48](#_Toc115767143)

[6.13 Version 19.0 49](#_Toc115767144)

[6.14 Version 19.2 50](#_Toc115767145)

[6.15 Version 20.0 51](#_Toc115767146)

[6.16 Version 20.1 51](#_Toc115767147)

[6.17 Version 20.2 51](#_Toc115767148)

[6.18 Version 21.0 52](#_Toc115767149)

[6.19 Version 21.1 53](#_Toc115767150)

[6.20 Version 22 54](#_Toc115767151)

[6.21 Version 22.0 56](#_Toc115767152)

[6.22 Version 23.0 56](#_Toc115767153)

[6.23 Version 23.1 57](#_Toc115767154)

[6.24 Version 24.0 57](#_Toc115767155)

[6.25 Version 24.1 59](#_Toc115767156)

[6.26 Version 24.3 60](#_Toc115767157)

[6.27 Version 24.4 61](#_Toc115767158)

[6.28 Version 24.5 61](#_Toc115767159)

[6.29 Version 25.0 62](#_Toc115767160)

[6.30 Version 25.1 62](#_Toc115767161)

[6.31 Version 25.2 63](#_Toc115767162)

[6.32 Version 26.0 63](#_Toc115767163)

# Einleitung

Die interne Funktionalität vom Offboard Diagnostic Information System Engineering wird über eine Automatisierungsschnittstelle nach außen zur Verfügung gestellt. Auf dieser Schnittstelle setzten die Makro- und Web-Service Schnittstelle auf. Makros, die in der Sprache Python geschrieben werden, greifen direkt auf die Automatisierungsschnittstelle zu. Damit entspricht die API der Automatisierungsschnittstelle eins zu eins der Makroschnittstelle.

Da sich Web-Service Schnittstelle und Automatisierungsschnittstelle gleich verhalten sollen, wird die Web-Service Schnittstelle aus der Automatisierungsschnittstelle heraus ohne weitere Indirektionsschicht erstellt. Damit wirken sich Änderungen der Automatisierungsschnittstelle immer auf die beiden anderen Schnittstellen aus.

In der Web-Service Schnittstelle werden nicht alle Methoden der Automatisierungsschnittstelle angeboten. Dabei handelt es sich z.B. um Hilfsfunktionen oder um vereinfachte Funktionen für die Makroschnittstelle. Diese Methoden werden in der Dokumentation besonders gekennzeichnet.

Es gibt jedoch auch eine Methode, die nur über die Web-Service Schnittstelle angeboten wird. Dabei handelt es sich um die Methode exit. Mit ihr wird das im Web-Service Modus gestartete Offboard Diagnostic Information System Engineering wieder beendet.

Die weitere Dokumentation ist in drei Teile gegliedert. Zuerst werden allgemeine Konzepte der Automatisierungsschnittstelle beschreiben, die für die Nutzung über Makros und Web-Service identisch sind. Danach wird auf die Besonderheiten von Makro und Web-Service eingegangen. Am Ende erfolgt die API Dokumentation der Automatisierungsschnittstelle.

# Konzeption der Automatisierungsschnittstelle

## Zustandsdiagramm

Einige Funktionen der Schnittstelle sind zustandsbehaftet und können nicht immer ausgeführt werden. Das Zustandsmodell der Schnittstelle ist in Abbildung 1 dargestellt. Es enthält alle Funktionsaufrufe, die zu einem Zustandsübergang führen. Exemplarisch sind auch Diagnosefunktionen aufgeführt, die im jeweiligen Zustand erlaubt sind, jedoch nicht zu einem Zustandswechsel führen.

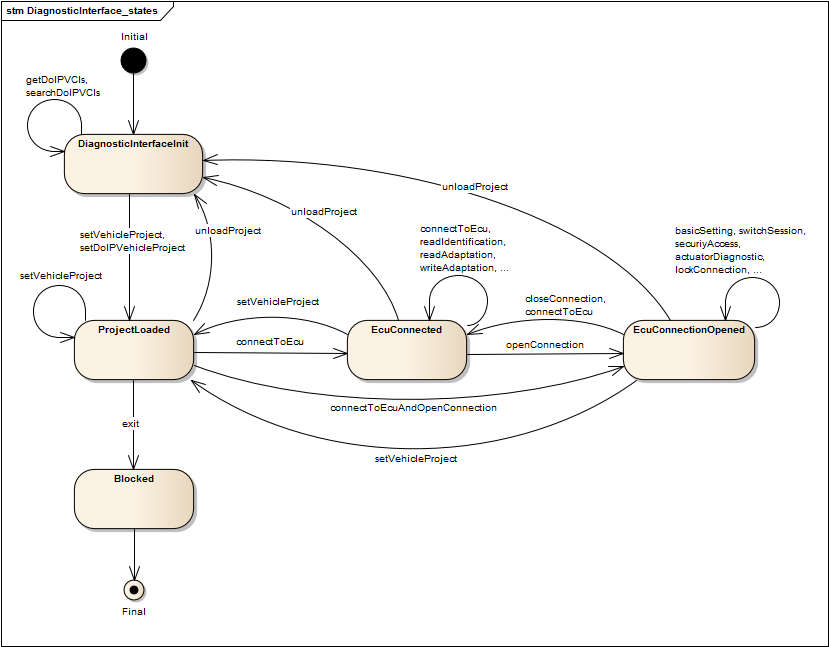


Abbildung 1: Zustandsdiagramm der Schnittstelle

## Richtung / Informationsaustausch

Bidirektional. Synchrone Anfragenbearbeitung.

### Abbildungsmöglichkeit von internen auf externe Bezeichner

Im Verzeichnis *AutomParamMapping* unterhalb des Offboard Diagnostic Information System Installationsverzeichnisses liegt die Datei *ODIS-Mapping.xml*. In ihr sind Abbildungsvorschriften von Offboard Diagnostic Information System systeminternen Bezeichnern auf externe definiert. Liegt für einen Bezeichner keine Abbildung vor, so wird der systeminterne Bezeichner verwendet.

Ausgewertet wird die Datei nur im Falle der Identifikation (readIdentification). Die Abbildung erfolgt auf dem Attribut IDiagResultValue.getName(). Im Fall von UDS, für den keine Abbildungen definiert sind, wird der ODX-Shortname geliefert

## Multilinkunterstützung

Ab Schnittstellenversion 3.0 (Offboard Diagnostic Information System Version 4.0) wird Multilink unterstützt. Die Parametrierung erfolgt über die Methode configureSettings mittels folgender Schlüssel:

|  |  |
| --- | --- |
| **Schlüssel configureSettings** | **Beschreibung** |
| Multilink.MaxNumberOfLogicalLinks | Maximale Anzahl von Logical-Links für die Nutzung von Multilinks |
| Multilink.ConsiderRestrictionsOfVRTVPT | Berücksichtigung der VRT / VPTs |

Die Parameter werden analog den Admin Einstellungen aus der GUI verwendet.

Alle Schnittstellenmethoden, die sich auf ein Steuergerät beziehen, haben als ersten Parameter ein Handle auf eine Steuergeräteverbindung (IConnectionHandle). Dieses Handle erhält man aus der Rückgabestruktur der Methoden, die eine Verbindung zu einem Steuergerät aufbauen (connectToEcu, connectToEcuAndOpenConnection und connectToEcuWithLogicalLink).

Fehlerfälle, wie z.B. Zugriff auf ein Steuergerät ohne freien Link, werden über die DiagException an den Aufrufer weiter gegeben.

Beispiel:

ODS8035E

Infrastruktur.Fahrzeug (EcuCom):

Die Kommunikation zum Steuergerät mit der Reizadresse 0x19 war fehlerhaft.

Der Verbindungsaufbau wird durch ein anderes Steuergerät blockiert.

## Versionierung

Da die Web-Service Schnittstelle aus der Automatisierungsschnittstelle erzeugt wird, haben beide Schnittstellen dieselbe Versionsnummer. Es kann jedoch vorkommen, dass nur eine Schnittstelle geändert wird, mit der Folge dass auch die jeweils andere Schnittstelle eine neue Versionsnummer bekommt, obwohl an ihr keine Veränderungen durchgeführt wurden.

Die Versionsnummer kann über die Methode getVersions().getAutomationAPIVersion() ermittelt werden.

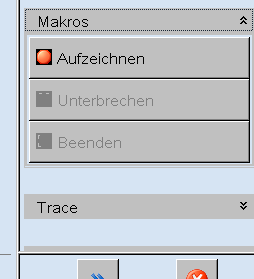
# Die Makrofunktion im Offboard Diagnostic Information System Engineering

## Aufgabe der Makrofunktion im Offboard Diagnostic Information System

Mit Hilfe der Makrofunktionen lassen sich Bedienungsabläufe vom Offboard Diagnostic Information System Engineering automatisieren. Der Benutzer wird hierdurch in die Lage versetzt, immer wiederkehrende Bedienabläufe aufzuzeichnen und später automatisch ablaufen zu lassen.

## Aufzeichnen von Makros

Um Bedienschritte des Benutzers aufzeichnen zu können, befindet sich im „Expandbar“ an der rechten Seite des Anwendungsfensters ein Gruppenfeld „Makros“:

Abbildung 3.1: Steuerung der Makroaufzeichnung im Expandbar

Mit Hilfe der drei Schaltflächen kann der Anwender die Makroaufzeichnung starten, unterbrechen und beenden.

* *Aufzeichnen*
  + Nach Betätigen der Schaltfläche „Aufzeichnen“ wird dem Benutzer über einen Dateiauswahldialog zunächst die Festlegung des Ablageortes für das neue Makro angeboten. Voreingestellt sind ein von der Anwendung gebildeter eindeutiger Dateiname und der im Anwendungsbereich „Administration“ eingestellte Ablageort für Makros. Beides kann vom Anwender in diesem Dialog auf seine aktuellen Bedürfnisse angepasst werden. Nach erfolgter Dateiauswahl werden alle folgenden Benutzeraktionen aufgezeichnet. Es werden nur tatsächlich ausgeführte Diagnosefunktionen ins Makro übernommen. Wird eine Diagnosefunktion vor ihrer Übertragung an das Steuergerät abgebrochen, erscheint sie nicht im Makro.
* *Unterbrechen*
  + Möchte der Benutzer bei laufender Aufzeichnung eine Diagnosefunktion zwar ausführen, diese aber nicht ins Makro übernommen haben, kann er die Makroaufzeichnung mit dieser Schaltfläche deaktivieren. Soll die Aufzeichnung dann wieder fortgeführt werden, erreicht man dies durch das erneute Betätigen der Schaltfläche „Aufzeichnen“.
* *Beenden*
  + Nach Betätigen dieser Schaltfläche wird die Aufzeichnung endgültig abgeschlossen und als Makrodatei am gewählten Ort abgelegt. Ein nachfolgendes Betätigen der Schaltfläche „Aufzeichnen“ beginnt eine neue Aufzeichnung und erzeugt ein weiteres Makro.

Zur Beachtung:

Das zum Zeitpunkt der Aufzeichnung geladene Fahrzeugprojekt wird nur in einem Kommentar in der Makrodatei gespeichert, damit das Makro auch für andere Fahrzeugprojekte verwendet werden kann. Auch Wechsel des Fahrzeugprojektes werden nicht aufgezeichnet..

## Ausführen von Makros

Die Ausführung eines Makros wird in der Funktionsauswahlliste über den Eintrag „Makros“ gestartet. Daraufhin erscheint ein Dialog, der alle Makros zur Auswahl anbietet, die im eingestellten Ablageverzeichnis abgelegt sind [[1]](#footnote-1).

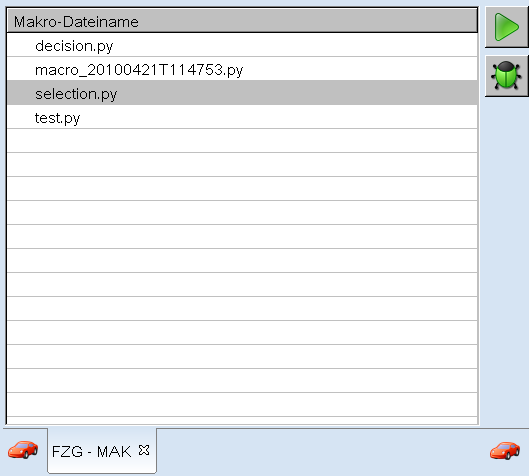


Abbildung 3.2: Dialog zum Starten eines Makros

Nach der Auswahl einer Makrodatei durch Selektieren mit der Maus oder Tastatur kann das gewählte Makro über eine von zwei Schaltflächen gestartet werden. Die obere Schaltfläche mit dem Pfeilsymbol startet das Makro normal, die untere Schaltfläche mit dem Käfersymbol im Debugmodus (Vgl. Abbildung 3.2).

Im normalen Ablaufmodus wechselt die Dialogdarstellung zu einer Konsolendarstellung, in der man die Textausgaben des Makros verfolgen kann. Darüber hinaus bietet dieser Dialog weitere Bedienmöglichkeiten unterhalb der Konsolausgabe.

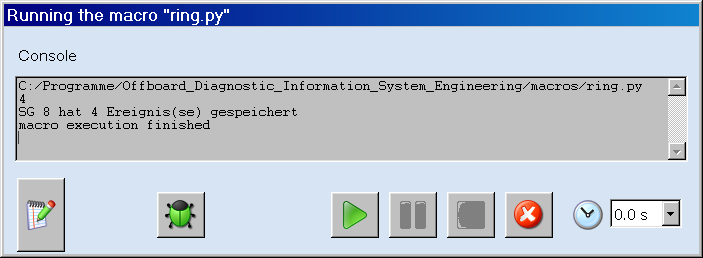


Abbildung 3.3: Ablauffenster eines Makros während der Ausführung.

|  |  |
| --- | --- |
|  | Wechsel in den bzw. Neustarten im Debug-Modus |
|  | Makro erneut starten (erst nach Ablauf des Makros aktiv) |
|  | Makroablauf unterbrechen |
|  | Makroablauf beenden |
|  | Dialog schließen. Nur aktiv, wenn das Makro beendet ist. |

Über die Auswahlliste in der rechten unteren Ecke des Dialogs kann die Wartezeit zwischen zwei Ausführungsschritten des Makros eingestellt werden. „0“ bedeutet dabei eine unverzögerte Ausführung. Alle Schaltflächen wirken nur auf die Makrobefehle. Das bedeutet, dass eine in einem Makroschritt ausgelöste Diagnosefunktion nicht unterbrochen wird. Erst wenn die Diagnosefunktion und damit der laufende Makroschritt ausgeführt sind, wird eine Unterbrechung oder Beendigung des Makros wirksam.

## Debuggen von Makros

Die Schaltfläche mit dem Käfersymbol in Abbildung 3.2 und Abbildung 3.3 führt in den Debug-Modus, in dem der Anwender den Ablauf des Makros visuell verfolgen kann. In diesem Modus öffnet sich ein erweiterter Ausführungsdialog:

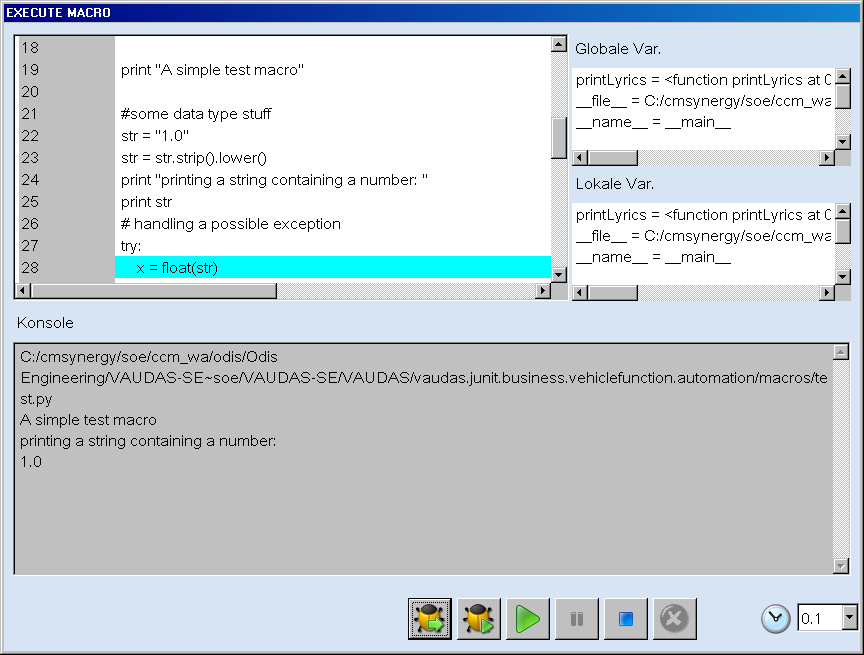


Abbildung 3.4: Ausführungsdialog im Debug-Modus

Nach einem Wechsel in den Debug-Modus ist der Debugger zunächst in der Betriebsart „Einzelschritt“, das heißt, das Makro wird zunächst nicht (weiter) ausgeführt. Mit der Schaltfläche  kann die nächste Makroanweisung zur Ausführung gebracht werden. Der Ablauf lässt sich oberhalb der Konsolausgabe links verfolgen. Die aktive Zeile des Makroskripts, also diejenige, die allls nächste auszuführen ist, bzw. diejenige, die sich gerade noch in Ausführung befindet, ist dabei farblich hervorgehoben. Im rechten oberen Teil des Dialogfensters werden die lokalen und globalen Variablen mit ihren momentanen Werten angezeigt. Die lokalen Variablen unterscheiden sich von den globalen dann, wenn das Makro ein Unterprogramm ausführt.

Die Schaltfläche  wechselt in die Betriebsart „Animation“. Hier wird das Makro mit den unten rechts eingestellten Zeitintervallen ausgeführt (minimal 0,1 s Schrittweite) und der Ablauf in der Darstellung des Makrocodes angezeigt. Bei Bedarf kann der Benutzer wieder in den Einzelschritt-Modus wechseln, oder den Ablauf mit der Schaltfläche  unterbrechen. Die Schaltfläche  wechselt in den normalen Ablaufmodus. Die übrigen Schaltflächen verhalten sich, wie beim Normalmodus beschrieben.

# Erstellen eigener Makros für das Offboard Diagnostic Information System

## „Getting Started“

Dieses Tutorial soll den interessierten Anwender schrittweise in die Lage versetzen, Makros für das Offboard Diagnostic Information System zu erstellen. Am besten arbeitet man damit, indem man auf seinem PC das Offboard Diagnostic Information System startet und die Beispiele sofort ausprobiert. Dazu verwendet man am besten einen Texteditor, der das Erstellen von Programmcode unterstützt. Eine gute Wahl dabei ist zum Beispiel der frei verfügbare PSPAD (<http://www.pspad.com/>), der die Syntax von Python hervorheben kann. Andere Editoren (z. B. Ultraedit) können das aber genauso.

Egal, welchen Editor man verwendet, Python hat eine Besonderheit, die beim Editieren sehr wichtig ist. Wie wir später sehen werden, wird der Programmcode mit Tabulatoren strukturiert. Diese haben bei Python also einen mehr als nur optischen Effekt. Obwohl der Python-Interpreter sich bemüht, auch mit sogenannten „Soft-Tabs“, also einer Ersetzung des Tabulatorzeichens durch eine entsprechende Anzahl Leerzeichen, zurechtzukommen, gelingt ihm das nicht immer. So kann es vorkommen, dass Sie den Programmcode anders wahrnehmen als der Interpreter. Das zeigt sich dann beim Programmlauf möglicherweise durch einen

IndentationError: expected an indented block ,

obwohl die Formatierung für Sie völlig korrekt aussieht.

Hier hilft nur, beim Editor die visuelle Darstellung von Leerzeichen und Kontrollzeichen zu aktivieren. Dann werden die Unterschiede zwischen dem, was man (nicht) sieht, und dem was im Programmcode tatsächlich steht, erst erkennbar.

## Die Makrosprache Python

Python ist eine stark verbreitete, dynamische, interpretierte Sprache, die oft als Skriptsprache genutzt wird. Python unterstützt sowohl objektorientierte als auch funktionale Programmierung. Die im Offboard Diagnostic Information System verwendete Implementierung ist das in Java programmierte „Jython“. Jython kann neben den zahlreichen frei verfügbaren Python-Bibliotheken auch Java-Klassen direkt einbinden. Die Sprachelemente von Python sollen hier nur kurz angerissen werden; eine Liste von Web-Links und Literaturempfehlungen für Sprachbeschreibungen und Tutorials findet sich im Anhang.

### Die wichtigsten Sprachelemente

Python verwendet zur Strukturierung von Programmblöcken im Unterschied zu anderen Programmiersprachen keine Schlüsselwörter oder geschweifte Klammern, sondern die Einrückung über Tabs. Dem ist beim Editieren besondere Aufmerksamkeit zu schenken.

#### Programmverzweigungen

Eine Programmverzweigung wird auch in Python durch eine if-Anweisung eingeleitet:

var1 = 65 # Wertebereich 0 bis 99

print „Ausgabe von var1 modulo 50“

**if (var1 > 50):**

# Block um einen TAB einruecken

print var1 - 50

**else:**

print var1

# Ende des else-Blocks durch Ende der Einrueckung

print „Good Bye“

Ausgabe:

Ausgabe von var1 modulo 50

15

Good Bye

An diesem Beispiel wird auch die Bildung von Programmblöcken deutlich. Hinter der Bedingung bzw. dem Schlüsselwort „else“ folgt ein Doppelpunkt, der bedingt ausgeführte Anweisungsblock ist um einen Tabulator eingerückt. Jede weitere Verschachtelung bedingt dabei einen weiteren Tabulator.

#### for-Schleifen

Eine klassische for-Schleife mit numerischem Schleifenzähler formuliert man in Python so:

for num in range(0,3):

print num

print "Good bye!"

Ausgabe:

0

1

2

Good Bye!

Darüber hinaus bietet Python bei for-Schleifen weitere Möglichkeiten. Es kann über jedes Element einer Sequenz iteriert werden:

Eine Sequenz kann dabei eine Zeichenkette sein …

for letter in **'Python':**

print letter

Ausgabe:

… oder eine Liste:

P

y

t

h

o

n

fruits = ['banana', 'apple', 'mango']

for fruit in fruits:

print fruit

print "Good bye!"

Ausgabe:

banana

apple

mango

Good Bye!

#### While-Schleifen

While-Schleifen werden wie in anderen Programmiersprachen formuliert:

var1 = 165 # Wertebereich: var1 >= 0

print „Ausgabe von var1 modulo 50“

**while (var1 > 50):**

var1 = var1 - 50

print var1

print „Good Bye“

Ausgabe:

Ausgabe von var1 modulo 50

15

Good Bye

#### Schleifenkontrolle

Wie andere Programmiersprachen auch, bietet Python die Möglichkeit mit dem Schlüsselwort „continue“ den aktuellen Schleifendurchlauf abzubrechen und mit der nächsten Iteration fortzufahren, oder mit „break“ die Schleife zu verlassen. Zusätzlich kann man hinter der Schleife eine „else“-Anweisung einfügen. Der hiermit eingeleitete Anweisungsblock wird nur ausgeführt, wenn die Schleife vollständig durchlaufen wurde.

for num in range(10,20):

for i in range(2,num):

if num % i == 0:

**break**

**else:**

print num, ' ist eine Primzahl'

Ausgabe:

11 ist eine Primzahl

13 ist eine Primzahl

17 ist eine Primzahl

19 ist eine Primzahl

#### Unterprogramme

Mit dem Schlüsselwort „def“ definiert man in Python eine Funktion:

def abs(val):

if (val > 0):

return val

else:

return (0-val)

#main

print abs(-5)

print „Good Bye!”

#### Strukturierte Datentypen

Ein grundlegender strukturierter Datentyp in Python ist die

Liste:

fruits = ['banana', 'apple', 'mango']

Der Zugriff auf ein Listenelement erfolgt mit:

X = fruits[1] -> liefert die Zeichenkette „apple“

Y = fruits[1:2] -> liefert eine Teilliste [‚apple’, ‚mango’]

Den Wert eines Listenelements kann man wie folgt ändern:

fruits[2] = ‚Kiwi’

Mit del fruits[1] wird das Listenelement mit dem Index 1 (also das zweite !) aus der Liste entfernt. **Achtung:** Die nachfolgenden Elemente rücken auf ! fruits[1] ist jetzt also „Kiwi“, ein Zugriff auf fruits[2] führt zu einem Fehler !

*Tuples* sind eine Sonderform von Listen, bei der der Wert eines Elements nicht verändert werden kann. Sie werden definiert wie Listen, nur mit runden statt mit eckigen Klammern. Der Zugriff auf ein einzelnes Element erfolgt wie bei einer Liste.

Listen werden auch dazu verwendet, um Arrays abzubilden. Da die Listenelemente einen unterschiedlichen Datentyp besitzen dürfen, kann man auch Strukturen mit einer Liste abbilden. Dazu nimmt man aber besser ein

Dictionary:

Ein Dictionary ist eine Liste, bestehend aus Schlüssel/Werte-Paaren:

kid = {'Name': ‚Sara', 'Alter': 7, ‚Klasse': ‚eins'}

Der Zugriff auf ein Element erfolgt mit:

x = kid[‚Name’] -> liefert als Ergebnis die Zeichenkette „Sara“

mit kid[‚Name’] = „Kylie“ kann man den Wert des Dictionary-Elements „Name“ ändern; mit del kid[‚Klasse’] kann man das Element mit dem Schlüssel „Klasse“ entfernen.

### Verwenden von Python-Bibliotheken

Mit dem import-Kommando können andere Python-Module, wie z. B. Bibliotheken in das Skript eingebunden werden.

import hello

Importiert das gesamte Modul hello.py

from hello import hello, hi, priwjet

Importiert die Funktionen hello, hi und priwjet aus hello.py

* + Aber nicht z. B. „hallo, hola oder witam“

Damit zur Laufzeit benutzerdefinierte Module für Python sichtbar sind, muss das Verzeichnis dieser benutzerdefinierten Module zu den „Python-Suchpfaden“ hinzugefügt werden. Dieser Aufruf muss in das ODIS-Makro-Skript vor dem Import-Aufruf integriert werden:

sys.path.append(<Modulverzeichnis>)

Fügt ein benutzerdefiniertes Verzeichnis für Module hinzu

### Verwenden von Java-Bibliotheken

In Jython, der Java-Implementierung von Python, lassen sich Java-Klassen und Interfaces genauso einfach importieren, wie Python-Module.

from java.util import Vector

Importiert z. B. die Klasse „Vector“ aus dem package java.util

Bei der Verwendung von eigenen Java-Klassen muss sichergestellt werden, dass der Ablageort der zugehörige JAR-Bibliothek durch eine entprechende Pfadangabe dem System bekanntgemacht wird. Erst im Anschluss kann ein erfolgreicher Klassenimport stattfinden.

sys.path.append("<absoluter Pfad>/<JAR-Dateiname>.jar")

from java.util import Vector

v = Vector()

v.add(`aaa`)

v.add(`bbb`)

for val in v:

print val

liefert als Ausgabe:

aaa

bbb

## Verwenden von Offboard Diagnostic Information System Diagnosefunktionen

Ein Makro im Offboard Diagnostic Information System kann nicht auf jede beliebige Java-Klasse des Programms Offboard Diagnostic Information System zugreifen sondern nur auf die Klassen der Automatisierungsschnittstelle.

### Offboard Diagnostic Information System Automatisierungsschnittstelle

Offboard Diagnostic Information System lässt sich über eine Webservice-Schnittstelle fernsteuern. Diese benutzt wie die Makros die interne Automatisierungsschnittstelle. Die Klassen dieser Schnittstelle und ihre Methoden werden in Kapitel 0 beschrieben.

### Code-Bausteine für Diagnosefunktionen

Um die Automatisierungsschnittstelle vom Offboard Diagnostic Information System einzubinden muss die Makrodatei zu Anfang die Zeile

from de.volkswagen.odis.vaudas.vehiclefunction.automation import IDiagnosticInterface

enthalten.

Durch diese Anweisungen werden die zur Ausführung von Diagnosefunktionen benötigten Klassen zugänglich.

#### Laden eines Fahrzeugprojekts und Zugriff auf ein Steuergerät

Voraussetzung für einen Wechsel des Fahrzeugprojekts ist ein Zugriff auf die Java-Klasse DiagnosticInterface. Diesen beschafft man sich mit der Anweisung

diagnosticInterface = IDiagnosticInterface.Factory.getInstance()

Das Laden des Fahrzeugprojekts erfolgt dann mit der Anweisung

diagnosticInterface.setVehicleProject('***Fahrzeugprojektname***')

Das angegebene Fahrzeugprojekt muss im Standardablagepfad für Fahrzeugprojekte vom Offboard Diagnostic Information System abgelegt sein.

Beispiel:

Mit

diagnosticInterface.setVehicleProject('VWFLEET')

öffnet man das Fahrzeugprojekt „VWFleet“

Mit der Anweisung

diagnosticInterface.connectToEcu(***ecu***)

wird die Diagnoseverbindung zum Steuergerät mit der Reizadresse ***ecu*** eröffnet.

Danach können auf dem gewählten Steuergerät Diagnosefunktionen ausgeführt werden.

diagnosticInterface.connectToEcu(1)

öffnet beispielsweise das Motorsteuergerät für die Diagnose.

Ab Release 4.0 vom Offboard Diagnostic Information System enthält das von dieser Methode zurückgegebene Objekt vom Typ *IDiagResultConnectEcu* eine Referenz, die in allen Methoden, die ausschließlich auf dieses eine Steuergerät zugreifen, angegeben werden muss. Das Referenzobjekt erhält man mit der Methode *getConnectionHandle()*. Für Offboard Diagnostic Information System Release 4.0 muss man also schreiben:

motSg = diagnosticInterface.connectToEcu(1)

motSgHandle = motSg.getConnectionHandle()

#### Ermitteln der Steuergeräteidentifikation

Die Steuergeräteidentifikation wird durch den Aufruf von

Ident = diagnosticInterface.readIdentification()

oder bei Offboard Diagnostic Information System, Release 4.0

Ident = diagnosticInterface.readIdentification(motSgHandle)

ausgelesen. Das Ergebnisobjekt Ident enthält die Identifikationsmerkmale.

sgName = Ident.getSystemName() liefert den Steuergerätenamen als Zeichenkette

standardId = Ident.getStandardData() liefert die Standardidentifikation als Objekt

extendedId = Ident.getExtendeddData() liefert die erweiterte Identifikation als Objekt

Diese Objekte sind vom Typ List<[DiagResultValueImpl](#r108)>, das bezeichnet eine Liste von Ergebniswerten in Form von Java-Objekten.

Diese Liste lässt sich mit Hilfe der Funktion getValueList (Vgl. Kapitel 4.3.3.1) in normale Python-Strukturen überführen:

pyStandardId = getValueList(standardId)

pyStandardId ist eine Liste von Dictionaries mit den Ergebniswerten.

Auf einen bestimmten Ergebniswert kann man wie folgt zugreifen:

firstResult = pyStandardId[0]

resultName = firstResult[‚name’]

resultValue = firstResult[‘value’]

…

#### Auslesen des Ereignisspeichers

Das Auslesen des Ereignisspeichers wird mit der Anweisung

eventMemories = diagnosticInterface.readEventMemory()

oder ab Release 4.0:

eventMemories = diagnosticInterface.readEventMemory(motSgHandle)

(*motSgHandle* , siehe Kapitel 4.3.2.2)

veranlasst.

Aus dem Java-Objekt eventMemories lassen sich die Details der Ereignisspeichereinträge ermitteln. Zunächst interessiert in den meisten Fällen wohl die Anzahl der gelesenen Ereignisse. Mit

size = eventMemories.getEventMemoryEntries().size()

kann man diese ermitteln.

An dieser Stelle können Sie bereits selbst Ihre erste Erweiterung vom Offboard Diagnostic Information System programmieren:

Ein Makro, das den Ereignisspeicher eines Steuergerätes überwacht, beim Auftreten eines Ereignisses ein akustisches Signal gibt, und sich dann beendet.

#

# Dieses Makro ueberwacht den Ereignisspeicher des Bremsensteuergeraets

# durch eine zyklische Abfrage und sendet ein akustisches Signal für

# jeden gefundenen Ereigniseintrag. Nach einer Stunde beendet sich das # Makro.

# zusätzlich ab Offboard Diagnostic Information System Release 4.0 erforderliche Elemente in dieser Farbe

#

from de.volkswagen.odis.vaudas.vehiclefunction.automation import IDiagnosticInterface

import time

#

# ein print “\a” (ASCII-Bell) kann im Offboard Diagnostic Information System nicht verwendet werden,

# daher:

from java.awt import Toolkit

ecu = 3 # Reizadresse des Bremsensteuergeraets

# Diagnoseinterface oeffnen, Fahrzeugprojekt waehlen und mit dem

# Steuergeraet verbinden:

diagnosticInterface = IDiagnosticInterface.Factory.getInstance()

#diagnosticInterface.setVehicleProject('VWFLEET')

myEcu = diagnosticInterface.connectToEcu(ecu)

myEcuHandle = myEcu.getConnectionHandle()

running = 0

while 1:

# Ereignisspeicher lesen

eventMemories = diagnosticInterface.readEventMemory(myEcuHandle)

# Anzahl der Ereignisse bestimmen

size = eventMemories.getEventMemoryEntries().size()

print size

# Wenn mindestens ein Ereignis gelesen wurde

if (size > 0):

# Ergebnisausgabe und <size> mal doppelt piepen

print "SG %d hat %d Ereignis(se) gespeichert" % (ecu, size)

for i in range(0,size):

# Doppel-Piep

Toolkit.getDefaultToolkit().beep()

time.sleep(0.2) # 0,2 s Pause

Toolkit.getDefaultToolkit().beep()

time.sleep(0.5) # 0,5 s Pause zwischen den Signalen

# nach 60 min. aus der Schleife heraus springen und Makro beenden

if running >= 60: break

running = running + 1

# Wartezeit von 60 Sekunden

time.sleep(60)

# diagnosticInterface.closeConnection(myEcuHandle)

#exit()

Noch schöner wäre es natürlich, wenn man nicht nur die Anzahl der Ereignisse anzeigen würde, sondern auch deren Inhalt. Dazu muss man aus dem zurückgegebenen Java-Objekt *eventMemories* die Detailinformationen herauslesen. Dieses Objekt vom Typ *DiagResultEventMemoryImpl* enthält eine Liste von Objekten des Typs [*IDiagResultEventMemoryEntry*](#r373), die jeweils die folgenden Informationen enthalten:

* *IDiagResultDtcState*: der interpretierte Inhalt des Statusbytes (als Liste)
* [*IDiagResultEnvironmentData*](#r342): Umgebungsbedingungen
* [*IDiagResultEventInfo*](#r348): detallierte Informationen zu dem Ereignis

Ein *IDiagResultDtcState*-Objekt beschreibt dabei ein Bit des Statusbytes. Seine Attribute können nur gelesen werden. [*IDiagResultEnvironmentData*](#r342) enthält die Umgebungsbedingungen zum betreffenden Ereignis in zwei Listen. Mit der Methode *getStandardData()* werden die Standardumgebungsbedingungen abgefragt, mit *getMeasurements()* zusätzlich verfügbare Messwerte. Letztere werden in Objekten vom Typ *IDiagResultValue* zurückgegeben. Die Standardumgebungsbedingungen werden in einer Liste von Objekten des Typs [*IDiagResultStandardEnvironmentCondition*](#r509) zurückgegeben. Dieses Objekt enthält zusätzlich zum Objekt *IDiagResultValue* ein weiteres Attribut, die Bedeutung bzw. den Typ der jeweiligen Umgebungsbedingung. [*IDiagResultEventInfo*](#r348) erweitert ebenfalls den Typ und enthält Informationen wie Ereigniscode und Ereignistext.

[IDiagResultEventMemory](#r364)

[IDiagResultEventMemoryEntry](#r373)

[IDiagResultEventMemoryEntry](#r373)

[IDiagResultEventMemoryEntry](#r373)

…

IDiagResultDtcState

IDiagResultDtcState

[IDiagResultEnvironmentData](#r342)

…

IDiagResultValue

IDiagResultValue

…

[IDiagResultStandardEnvironmentCondition](#r509)

[IDiagResultStandardEnvironmentCondition](#r509)

…

IDiagResultEventInfo

IDiagResultEventInfo

…

Abbildung 4.1: Objektstruktur der Rückgabe von Ereignisspeicherinhalten

Das obenstehende Makro lässt sich wie folgt um eine Ausgabe der gelesenen Ereignisspeicherinhalte erweitern:

Toolkit.getDefaultToolkit().beep()

time.sleep(0.5) # 0,5 s Pause zwischen den Signalen

printEventMemoryEntries(eventMemories.getEventMemoryEntries())

# nach 60 min. aus der Schleife heraus springen und Makro beenden

if running == 60: break

Die Funktion *printEventMemories* bearbeitet dabei die einzelnen Ereignisspeichereinträge:

#

# Drucke alle Ereignisspeicherinhalte in der uebergebenen Liste

# - Status

# - Event info

# - Environment data

# Parameter: list<IDiagResultEventMemory> entries

# Rueckgabe: Anzahl der bearbeiteten Fehlereintraege

#

def printEventMemoryEntries(entries):

i = 0

size = entries.size()

for entry in entries:

print "Ereignisspeichereintrag: ", i

# Ausgabe des Fehlerstatus

print '===> status record:'

printEventMemoryEntryStatus(entry.getDtcStates())

# Ausgabe der Ereignis-Infos (Ereigniscode, -text, …)

print '===> Ereignisspeicher Info'

infoList = entry.getEventInfos()

printResultValueList(infoList)

# Umgebungsdaten

print "===> Umgebungsdaten:"

envDat = entry.getEnvironmentData()

printResultValueList(envDat.getStandardData())

printResultValueList(envDat.getMeasurements())

print '------------------------------------------'

i = i + 1

return i

Die Ausgabe der drei Informationsarten wird dabei an die zwei Funktionen *printEventMemoryEntryStatus* und *printResultValueList* delegiert:

# Gibt den Inhalt des Statusbytes eines Fehlerspeichereintrags aus

# Parameter: stateList List<[IDiagResultDtcState](#r335)>

def printEventMemoryEntryStatus(stateList):

print "Status: "

for bit in stateList:

print bit.getBitPosition(), bit.getBitPositionText(), bit.getBitValueText

print ‘---------------------------------‘

#

# Gibt eine Liste von Diagnoseergebnissen aus

# Parameter: <List <IDiagResultValue>> valueList

def printResultValueList(valueList):

if not valueList.isEmpty():

iterate = valueList.iterator()

while iterate.hasNext():

printResultValue(iterate.next())

else:

print 'Keine Ergebniswerte vorhanden'

*printResultValueList* ruft seinerseits die Funktion *printResultValue* auf, die einen einzelnen Diagnose-Ergebniswert ausgibt:

#

# Gibt ein Diagnoseergebnis auf der Konsole aus

# Parameter result <IDiagResultValue> result

#

def printResultValue(result):

print 'name: ', result.getName(), ' value: ', result.getValue()

un = result.getTranslatedName()

uv = result.getTranslatedValue()

print 'translated: name', un.encode('utf-8', 'replace'), ' value: ', uv.encode('utf-8', 'replace')

unit = result.getUnit()

print 'unit: ', unit.encode('utf-8', 'replace')

children = result.getChildren()

if not children.isEmpty():

iterate = children.iterator()

while iterate.hasNext():

printResultValue(iterate.next())

#### Lesen von Messwerten

Messwerte werden mit Hilfe der Methoden *readMeasurement* (ein Messwert) oder *readMeasurements* (eine Liste von Messwerten) gelesen.

Beide liefern ein Objekt vom Typ IDiagResultValue als Ergebnis zurück, das bei Bedarf hierarchisch aufgebaut ist. Als Eingabeparameter benötigen die Methoden eine Messwertgruppe und einen Messwertbezeichner. Bei einem UDS-Steuergerät sind das die entsprechenden ODX-Kurznamen, bei KWP-Steuergeräten Blocknummer und Position:

result = readMeasurement(„myMeasGroup“, „myMeasName“) # UDS

bzw.

result = readMeasurement(“1”, “3”) # KWP

Die Methode readMeasurements benötigt jeweils eine Liste als Parameter:

groupList = [‚SN\_myGroup1’, ‚SN\_myGroup2’, ‚SN\_myGroup1’, ‚SN\_myGroup3’]

measList = [‘SN\_meas1’, ‘SN\_meas5’, ‘SN\_meas4’, ‘SN\_meas11’]

result = readMeasurements(groupList, measList)

Zu beachten ist hier, dass Gruppen- und Messwerteliste in der Position der Listeneinträge miteinander korrespondieren müssen. Das bedeutet, dass Gruppennamen in der Liste mehrfach enthalten sein können.

### Nützliche Hilfsbibliotheken

#### Umwandlung von Diagnoseergebniswerten

Die Diagnosefunktionen geben häufig Java-Objekte des Typs *IDiagResultValue* oder einen der davon abgeleiteten Typen [*IDiagResultStandardEnvironmentCondition*](#r509) und [*IDiagResultEventInfo*](#r348) zurück, der einen Ergebniswert der Diagnosefunktion enthält. Da ein Diagnoseergebnis auch hierarchisch aufgebaut sein kann, enthält es möglicherweise Kindelemente, die selbst wieder Diagnoseergebnisse sind. Dadurch ergibt sich eine Baumstruktur beliebiger Tiefe. Die Funktion getResultValue wandelt die Offboard Diagnostic Information System Engineering-typischen Java-Strukturen in ein normales *Python-Dictionary* um, das allerdings selbst wieder hierarchisch aufgebaut sein kann.

# transformers

#

# Die Funktionen dieser Bibliothek transformieren die Java Objekte,

# die von der Automatisierungsschnittstelle in Python

# Datenstrukturen

#

# **getResultValue**

#

# transformiert einen diagnostic result value in ein

# Python-Dictionary

#

# **parameter:** <IDiagResultValue> javaResult = ein diagnostic result

# **returns:** ein Python dictionary, dass den Inhalt des

# diagnostic result value enthaelt

#

# **Schluessel des dictionary:**

# **name** = name des diagnostic result value

# **value** = sein Wert

# **translatedName** = sein translated name

# **translated value** = sein translated value

# **unit** = seine Einheit

# **nChildren** = Anzahl seiner Kindelemente

# **0...n** = seine Kinder (selbst wieder dictionaries,

# rekursiv)

#

def getResultValue(javaResult):

r = {'name' : result.getName(), 'value' : result.getValue()}

r['translatedName'] = result.getTranslatedName()

r['translatedValue'] = result.getTranslatedValue()

r['unit'] = result.getUnit()

# rekursives Abarbeiten der Objekthierarchie

i = 0

nChildren = result.getChildren()

if not nChildren.isEmpty():

iterate = nChildren.iterator()

while iterate.hasNext():

r[i] = getResultValue(iterate.next())

r['nChildren'] = i

return r

# **getResultValueList**

#

# transformiert eine Liset von diag result values in eine Liste von

# Python Dictionaries

#

# **parameter:** <List <IDiagResultValue>> valueList = Liste von

# diagnostic results

# **returns:** Python-Liste mit Python Dictionaries mit Inhalt der

# diagnostic result values

#

def getResultValueList(valueList):

l = []

if not valueList.isEmpty():

iterate = valueList.iterator()

while iterate.hasNext():

l.append(getResultValue(iterate.next()))

return l

# **getMap**

#

# transformiert eine Java Map in ein Python dictionary

# parameter: <Map> m = Java Map Objekt

# returns: Python Dictionary mit Inhalt der map

#

def getMap(m):

d = {}

keys = m.keySet()

if not keys.isEmpty():

iterate = keys.iterator()

while iterate.hasNext():

key = iterate.next()

val = m.get(key)

d[key] = val

return d

## Migrationsunterstützung Makros auf Schnittstellenversion 3.0

Bei dem Wechsel auf Schnittstellenversion 3.0.0 (Offboard Diagnostic Information System Version 4.0) gab es aufgrund der neuen Multilinkfunktionalität eine Änderung der Signatur bestehender Methoden. Die Änderung bezieht sich auf ein zusätzliches erstes Argument IConnectionHandle. Für bestehende Makros gibt es die Möglichkeit diese automatisch in die neue Signatur zu konvertieren. Dafür gibt es in dem Ordner macros das Makro macro\_migration.py. Es kann wie alle anderen Makros aus der Makroansicht gestartet werden. Nach dem Start konvertiert es alle Makros aus dem Makroordner in das neue Format. Zur Sicherheit wird in einem Unterordner bak eine Kopie des ursprünglichen Makros angelegt. Diese Kopie wird nur für diejenigen Makros angelegt, die auch durch das Migrationsmakro verändert wurden.

# Automatisierungsschnittstelle über Web-Service

## Schnittstellenverfahren

Eine Web-Service Schnittstelle wird allgemein über eine WSDL definiert. Diese wird für das Offboard Diagnostic Information System Engineering generisch aus der Implementierung der Automatisierungsschnittstelle erzeugt. Aus diesem Grund wird die WSDL nicht gesondert dokumentiert. Die API Dokumentation erfolgt ausschließlich für die prozedurale Automatisierungsschnittstelle. Für Nutzer der Makroschnittstelle sind die Signaturen identisch. Web-Service Schnittstellennutzer werden sich aus der WSDL einen Zugriffsstub in der gewünschten Programmiersprache erstellen lassen. Dieser Stub ist abhängig von der eingesetzten Sprache und dem Generierungstool. Die Signaturen des Stubs sind nicht identisch mit denen der Automatisierungsschnittstelle. Dies betrifft insbesondere die Namen der nicht primitiven Ein- und Ausgabestrukturen und von Exceptions. Die Methodennamen werden in den meisten Fällen übereinstimmen.

Für die Rückgabe / Übergabestrukturen und Exceptions gelten im Normalfall folgende Abbildungen:

|  |  |
| --- | --- |
| **Automatisierungsschnittstelle** | **Stub** |
| Typ: I<XXX> | Typ: Diag<XXX>Impl |
| Übergabeparameter mit Datentyp Map:   * writeTextCoding(…,Map codingValues,…) * writeSpecialCoding(…,Map parameters,…) * writeAdaptation(…,Map adaptationValues,…)   Analoges Vorgehen für Rückgabestrukturen mit dem Datentyp Map. | Datentyp MapAdapter, da die WSDL keine Maps unterstützt:   * writeTextCoding(…,MapAdapter codingValues,…) * writeSpecialCoding(…,MapAdapter parameters,…) * writeAdaptation(…,MapAdapter adaptationValues,…) |
| setTraceLevel(Level level) | setTraceLevel(String level)  Analog zur Methode der Automatisierungsschnittstelle sind folgende Werte für den Parameter level gültig:  {“ALL”, “DEBUG”, “INFO”, “WARN”, “ERROR”, “FATAL”, “OFF”}  Wird ein ungültiger Wert versucht zu setzen, so wird intern der Level auf FATAL gesetzt.  Die Unterscheidung in der Signatur war notwendig, da die Methode der Automatisierungsschnittstelle eine interne Java-Struktur nutzt. |
| DiagException | Falls die Zielsprache ein Exceptionkonzept enthält, wird der Name DiagException wahrscheinlich in der generierten Struktur verwendet. |

Der zur Verfügung gestellte Web-Service ist kompatibel zu W3C Version 1.1.

## Konfiguration

### Start Offboard Diagnostic Information System Engineering als Web-Service Server

In den Konfigurationsdateien config.ini und webservice.ini im Verzeichnis configuration können folgende Schalter verwendet werden, um die Web-Service Eigenschaften zu steuern.

*de.volkswagen.odis.vaudas.vehiclefunction.automation.webservice.enabled=true*

Dieser Schalter bewirkt, dass die Automatisierungsschnittstelle als Web-Service zur Verfügung gestellt und die Anwendung ohne GUI gestartet wird.

Beendet werden muss die Anwendung über die Methode *exit* des Web-Service.

Die Web-Schnittstelle stellt ihre Dienste ausschließlich für Programme auf dem lokalen Rechner bereit. Die Web-Schnittstelle ist von außerhalb nicht erreichbar (hat keine extern vermittelbare IP-Adresse).

Der Port über den der Web-Service gestartet wird, kann ebenfalls konfiguriert werden.

*de.volkswagen.odis.vaudas.vehiclefunction.automation.webservice.port=8081*

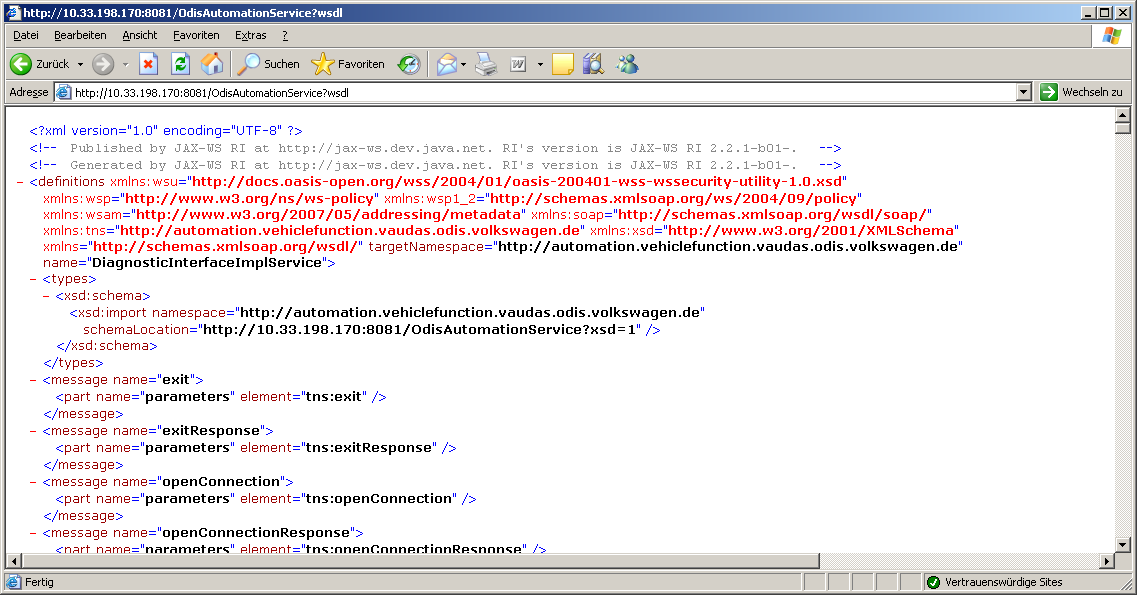
Der Web-Service wird dann über folgende URL gestartet:

http://<IP>:<PORT>/OdisAutomationService

<IP> „localhost“ oder „127.0.0.1“

<PORT> Der konfigurierte Port (Default: 8081)

Über die URL „http:://localhost:8081/OdisAutomationService?wsdl“ kann z.B. der Inhalt der WSDL-Datei in einem Web-Browser angezeigt werden.



Unter Nutzung der beschriebenen Schalter gibt es 2 Möglichkeiten, um Offboard Diagnostic Information System Engineering als Web-Service Server zu starten.

#### Nutzung Pogrammargument –configuration beim Start von OffboardDiagEngineering.exe

Mit dem optionalen Argument "-configuration" wird der relative Pfad ausgehend vom Offboard Diagnostic Information System Installationsverzeichnis zu einer zusätzlichen Konfigurationsdatei übergeben, z.B. "-configuration configuration\webservice.ini". Anstatt des relativen Pfades kann ebenso eine absolute Pfadangabe für die Konfigurationsdatei verwendet werden.

Damit können die Schalter für die Web-Service Server Nutzung in eine weitere Datei, wie z.B. webservice.ini verlagert werden und müssen nicht in der config.ini enthalten sein.

**Starten im Web-Service Server Modus:**

*OffboardDiagLauncher.exe -configuration configuration\webservice.ini*

Für einen Start ohne Web-Service Server Modus muss OffboardDiagLauncher.exe ohne Argument aufgerufen werden.

#### Parametrierung in der config.ini

Alternativ zu der Nutzung einer separaten Konfigurationsdatei für die Web-Service Server Schalter, können diese auch in der config.ini verwendet werden. Dann muss OffboardDiagLauncher.exe ohne Argument aufgerufen werden. Bei dieser Lösung ist allerdings immer eine Änderung der config.ini nötig, um zwischen Web-Service Modus und „interaktivem Modus“ zu wechseln.

### Festgelegte Admin-Einstellungen

Generell wirken sich die Admin-Einstellungen aus der Admin-Ansicht vom Offboard Diagnostic Information System Engineering auf das Verhalten der Web-Service-Schnittstelle aus. So werden z.B. die übersetzbaren Texte in der eingestellten Sprache zurückgegeben. Über die Automatiserungsschnittstelle lassen sich einige Einstellungen über die Methode configureSetting konfigurieren.

Für die in der folgenden Tabelle aufgelisteten Einstellungen sind Werte definiert, die unabhängig von den Einstellungen in der Admin-Ansicht verwendet werden.

|  |  |
| --- | --- |
| **Schalter Admin** | **Wert** |
| Funktionskonfiguration -> Steuergeräteliste -> Auswertung Gateway Verbauliste | TRUE |
| Funktionskonfiguration -> Steuergeräteliste->Dynamische Anpassung der Auswahl von Diagnosefunktionen | FALSE |
| Funktionskonfiguration -> Steuergeräteliste -> Darstellung von Subsystemen | FALSE |
| Funktionskonfiguration -> Zykluszeiten -> Steuergeräteliste | FALSE |
| Funktionskonfiguration -> Zykluszeiten -> Sammeldienst Ereignisspeicher | FALSE |
| Funktionskonfiguration -> Zykluszeiten -> Ereignisspeicher | FALSE |
| Funktionskonfiguration -> Zykluszeiten -> Messwerte | FALSE |
| Funktionskonfiguration -> Zykluszeiten -> Rollenmodus | FALSE |
| Funktionskonfiguration -> Codierung -> Überprüfung mit Prüfziffer aktivieren | FALSE |

## Logging

Eine Beispielkonfiguration *soapLog4J.properties* für das Logging wird mitgeliefert. Sie muss in das Verzeichnis Configuration kopiert werden.

Konfiguriert sind 3 Logdateien mit dazugehörigem Level:

1. engine.log: Offboard Diagnostic Information System Engineering Meldungen im Level ERROR
2. ecf.log: ECF Meldungen im Level ERROR
3. soap.log: Web-Service SOAP Nachrichten Meldungen im Level ERROR

Die Dateien werden in das Unterverzeichnis log der Installation geschrieben.

Für DEBUG-Zwecke kann die Konfiguration zur Laufzeit geändert werden. Dies ist inbesondere für das Ein- und Ausschalten des Loggings der SOAP-Nachrichten gedacht. Dazu muss in der Konfigurationsdatei *soapLog4J.properties* für das Einschalten der Level von ERROR auf INFO gesetzt werden. Änderungen der Konfigurationsdatei werden zyklisch jede Sekunde überprüft und dann wirksam gesetzt.

## Transaktionssicherheit

Derzeit ist das gewährleisten der Transaktionssicherheit nicht erforderlich. Es wird sichergestellt, dass nur ein Verarbeitungsthread die Anfragen der Web-Service Schnittstelle synchron bearbeitet.

## Fehlerbehandlung

Alle Methoden enthalten DiagException in ihren Signaturen. Diese Exception wird bei allen Fehlern innerhalb der Ausführung von Schnittstellenmethoden geworfen. Zusätzlich werden Exceptions in Zusammenhang mit der technischen Nutzung der Schnittstelle erzeugt, wie z.B. bei Vebindungsprobleme zu dem Web-Service. Eine Ausnahme bildet die IllegalArgumentException in setCommunicationTrace. Sie kann nur bei Verwendung von Clients vorkommen, die nicht typisiert auf die Schnittstelle zugreifen.

## Security

### Zugangsberechtigungen

Derzeit wird keine Überprüfung der Berechtigung durchgeführt.

### Verschlüsselung der Daten

Trusted Concept Inhouse (ohne Verschlüsselun

## Protokollierung

Die Automatisierung und Programmierschnittstelle bietet jetzt die Möglichkeit, Protokolle anzulegen und die Protokollfunktionen zu steuern. Das Interface IDiagnosticInterface bietet folgenden Funktionen:

* initProtocol() ermöglicht ein Protokoll anzulegen
* stopProtocol() ermöglicht die Aufzeichnung von Diagnoseprotokollen zu stoppen
* startProtocol() ermöglicht die Aufzeichnung von Diagnoseprotokollen zu starten
* saveProtocol() ermöglicht angelegte Diagnoseprotokolle zu speichern
* discardProtocol() ermöglicht das aktuell im Speicher befindliche Protokoll zu verwerfen

Abbildung „Zustandsdiagramm Protokollsteuerung“ zeigt die beschriebenen Befehle in einem Zustandsdiagramm.



Standardmäßig ist bei der Automatisierung und Programmierschnittstelle die Protokollierung deaktiviert. Lediglich während der Makroaufzeichnung werden die Makrobefehle startProtocol() und stopProtocol() dem Makro automatisch hinzugefügt, wodurch der gesamte Makroablauf protokolliert wird. Des Weiteren findet während der Makroausführung eine automatische Initialisierung beim Makrostart und automatische Speicherung beim Makroende statt. Durch das manuelle Hinzufügen des Makrobefehls stopProtocol()kann die Protokollierung jederzeit pausiert und durch startProtocol() aktiviert werden. Um innerhalb eines Makroablaufes ein neues Protokoll anzulegen, steht der Befehl initProtocol() zur Verfügung. Dieser Befehl ist jedoch nur nach einem Speichern oder Verwerfen des Protokolls erlaubt. Im Zustandsdiagramm ‚Protokollsteuerung‘ sind die möglichen Zustandsübergänge dargestellt.

## Besonderheiten

Falls keine gültige Lizenz vorliegt, wird die Anwendung nicht gestartet. Es wird kein spezieller Hinweisdialog angezeigt. Die Begründung geht aus der Logdatei hervor.

# API Änderungen Offboard Diagnostic Information System

## Version 4.0

Folgende Methoden sind in der Version 4.0 (API-Version 3.0.0) neu hinzugekommen (Siehe auch Änderungshistorie):

* void unloadProject()
* List<IDoIPVCI> getDoIPVCIs()
* List<IDoIPVCI> searchDoIPVCIs(String ipAddress)
* setDoIPVehicleProject(IDoIPVCI doIPVCI, String projectName)
* IDiagFlashSessionDescriptor createFlashSessionDescriptor(int ecuAddress, String containerFilePath, String sessionName, Boolean checkSessionWithEcu)
* IDiagResultFlashProgramming flashProgrammingParallel(List<IDiagFlashSessionDescriptor> diagFlashSessionDescriptors)
* getLockedConnections()
* lockConnection(IConnectionHandle)
* unlockConnection(IConnectionHandle)
* connectToEcuWithLogicalLink(String logicalLinkname)
* connectToEcuAndOpenConnection(int address)

## Version 5.0

Folgende Änderungen wurden vorgenommen:

* Schnittstellenfunktion flashProgramming um die Rückgabestruktur IDiagResultFlashSession erweitert.
* Erweiterung der Rückgabestruktur IDiagResultFlashSession um die Abfrage der nicht erfüllten Programmiervorbedingungen (IDiagResultPreCondition).
* Erweiterung der Rückgabestruktur IDiagResultValue um die Abfrage des RecordDataIdentifiers.
* Folgende Parameter für die Admin-Einstellung wurden hinzugefügt: DiagEntrance.DayCounter, DiagEntrance.LastClearDateOfFunctionalEventmemory , DiagEntrance.Milage, DiagEntrance.OnboardTime , DiagEntrance.WarmupCycleCounter
* Änderungen des Datentyps der Parameter im Interface IMemoryCellsInterface, in den Methoden readEEPROM(IConnectionHandle, BigInteger, Integer) und writeEEPROM(IConnectionHandle, BigInteger, byte[]). Früher war readEEPROM(IConnectionHandle, Integer, Integer) und writeEEPROM(IConnectionHandle, Integer, byte[]).

## Version 11.1

Folgende Änderungen wurden vorgenommen:

* Zur Abfrage der API-Verison steht nun folgende Methode bereit:

String **getAutomationApiVersion**()

## Version 12.0

Folgende Änderungen wurden vorgenommen:

* Für das Versenden von Protokollen existiert nun für jede Versendeart (VDS, Carport und File-Server) eine eigene Schnittstellenfunktion. Diese lauten:

IDiagResultProtocolTransfer **sendProtocolsToCarport**(<Protokollliste>, <taskCode>)

IDiagResultProtocolTransfer **sendProtocolsToFileServer**(<Protokollliste>, <Verzeichnis>)

IDiagResultProtocolTransfer **sendProtocolsToVDS**(<Protokollliste>)

Die zurückgelieferten Ergebniselemente 'IDiagResultProtocolTransferEntry' der einzelnen Protokolle enthält nur noch den Protokollnamen, den Versendestatus und ggfs. einen Hinweistext der zuvor aufgerufenen Versendeart.

* Die Methoden zum Erstellen von BZD-/VDS-Protokollen wurden zu einer Methode zusammengefasst und um das Argument für die zusätzlichen, exportierten Presets erweitert:

String **createProtocolVDSBZD**(<VIN>, <Fzg-Nummer>, <readEventMemory>, <Benutzername>, <Messort>, <Messtyp>, <Kilometerstand>, <Einheit Kilometerstand>, <Anpassungsliste>, <Messwertliste>, <Zusätzliche Presets>)

* Für die Ermittlung der zusätzlichen, verfügbaren Presets aus dem Export-Verzeichnis wird nun die folgende Methode bereitgestellt:

List<String> **getAvailableAdditionalPresets**()

* Für das Löschen von Ereignisspeicherinhalten exisitert nun eine zusätzliche Methode, welche das De-/Aktivieren der einzelnen Löschverfahren sowie die Vorgabe einer bestimmten Steuergeräteauswahl zulässt.

void **resetEventMemories**(<OBD-Löschen>, <Funktional-Löschen>, <Physikalisch-Löschen>, <Wartezeit vor KD-Bit-Aktualisierung>, <SG-Liste>)

* An der Automatisierungsschnittstelle (jedoch nicht an WEB-API) existieren neue Methoden zum Aktualisieren der Verbauliste und zum Lesen von Ereignisspeichereinträgen für eine bestimmte Auswahl von Steuergeräten.

**updateGatewayComponentList**()

List<IDiagResultEventMemory> **readEventMemories**(<SG-Liste>, <Anpassungsliste>, <Messwertliste>, <Zusätzliche Presets>)

## Version 12.2

Folgende Änderungen wurden vorgenommen:

* Für das Initialisieren von Stellglieddiagnosen / Grundeinstellungen mit Messwerten unterschiedlicher Steuergeräte werden nun zusätzlich die folgenden Automatisierungsfunktionen bereitgestellt:

**actuatorDiagnostics2Init**(IConnectionHandle connectionHandle,

List<IDiagDescriptor2WithParametersAndMeasurements> actuators,

List<IDiagMeasurementDescriptor> measurements)

**basicSetting2Init**(IConnectionHandle connectionHandle,

List<IDiagDescriptor2WithParametersAndMeasurements> basicSettings,

List<IDiagMeasurementDescriptor> measurements)

Dabei werden die Messwerte mit dem neuen Object „IDiagDescriptor2WithParametersAnd Measurements“ beschrieben.

* Zusäzlich existieren an der Makro-Schnittstelle die folgenden zwei Funktionen zum Erzeugen von Stellglieddiagnosen / Grundeinstellungen mit Messwerten unterschiedlicher Steuergeräte:

IDiagDescriptor2WithParametersAndMeasurements **createDiagActuator2**(String actuatorName,

Map<String, String> parameters,

List<IDiagMeasurementDescriptor> measurements)

IDiagDescriptor2WithParametersAndMeasurements **createBasicSetting2**(

String basicSettingName,

Map<String, String> parameters,

List<IDiagMeasurementDescriptor> measurements)

## Version 13.1

Folgende Automatisierungsfunktionen wurden aus der Schnittstelle entfernt:

* + **startRollMode()**
  + **stopRollMode()**
  + **getRollModeState()**

## Version 14.1

Folgende Änderungen wurden vorgenommen:

* Für das Ermitteln der verfügbaren MemorySelections (Speicherbereiche der Entwickler-Ereignisspeicher) wird folgende Funktion bereitgestellt:

List<String> **readAvailableDevEventMemorySelection**(IConnectionHandle connectionHandle)

* Für das Lesen der Entwickler-Ereignisspeicher steht folgende Funktion zur Verfügung:

IDiagResultEventMemory **readDevEventMemory**(IConnectionHandle connectionHandle,

final List<String> memorySelections,

final Long dtcStatusMask)

* Für das Löschen der Entwickler-Ereignisspeicher steht folgende Funktion zur Verfügung:

**resetDevEventMemory**(IConnectionHandle connectionHandle,

List<String> memorySelections)

## Version 15.2

Folgende Änderungen wurden vorgenommen:

* Für das Versenden von Protokollen an Carport wurde die bestehende Funktion um den optionalen Parameter brand erweitert. Die überarbeitete Funktion lautet:

IDiagResultProtocolTransfer **sendProtocolsToCarport**(<Protokollliste>, <taskCode>, <brand>)

* Für das Ermitteln der beim Senden an Carport verfügbaren Marken wird folgende Funktion bereitgestellt:

List<String> **readAvailableCarportBrands**()

* Für das Ermitteln der DTC Snapshot Records stehen die beiden folgenden Funktionen zur Verfügung. Mit der ersten Methoden können einzelne und mit der zweiten Methode alle Snapshot Records für einen bestimmten Ereignis-Code ermittelt werden:

[IDiagResultDtcSnapshot](#r148) **readEventMemorySnapshot**([IConnectionHandle](#r33) connectionHandle,

Integer eventCode,

Integer snapshotRecordNumber)

[IDiagResultDtcSnapshotList](#r150) **readEventMemorySnapshots**([IConnectionHandle](#r33) connectionHandle,

Integer eventCode)

* Des Weiteren wurde die bestehende Funktion „readEventMemoryWithMask“ um das optionale Flag „readSnapshotRecords“ erweitert, wodurch zusätzlich zu der Ermittlung der Ereignisspeichereinträge automatisch auch die Ermittlung der DTC Snapshot Records aktiviert werden kann. Das Ergebniselement enthält in diesem Fall auch die Snapshot Records.

IDiagResultEventMemory readEventMemoryWithMask(IConnectionHandle connectionHandle,

long dtcStatusMask,

Boolean readSnapshotRecords)

* Zur Ermittlung der Anzahl von verfügbaren DTC Snapshot Records für einen bestimmten Ereignis-Code steht die Funktion „getNumberOfEventMemorySnapshots“ zur Verfügung.

Integer **getNumberOfEventMemorySnapshots**([IConnectionHandle](#r33) connectionHandle,

Integer eventCode)

* Für den Export alle DTC Snapshot Records eines Steuergerätes wird die Funktion „exportEventMemorySnapshots” bereitgestellt. Sollten zum Export-Zeitpunkt einige Snapshot Records fehlen, so werden diese ermittelt, um ein vollständiges Export-Protokoll zu gewährleisten. Als Zielpfad kann ein leerer Pfad, ein Verzeichnis oder kompletter Dateipfad angegeben werden. Fehlende Pfadelemente werden durch Default-Werte der Anwendung ersetzt. Die Funktion liefert den vollständigen, verwendeten Ablagepfad zurück.

String **exportEventMemorySnapshots**(IConnectionHandle connectionHandle,

String targetFilePath)

## Version 15.3

Folgende Änderung wurde vorgenommen:

* Für die Ausführung von Python-Makros wurde eine neue Funktion hinzugefügt. Die Funktion lautet:

boolean **executeMacro**(<MacroFile>)

Der Aufrufparameter MacroFile ist der vollständige Pfadname der Makro-Datei.

## Version 16.0

Folgende Änderungen wurden vorgenommen:

* Für das Lesen der Subsystem-Verbaulisten wird die Funktion „readSubsystemComponentList“ bereitgestellt. Mit Hilfe des Connection-Handles wird das auszulesende Steuergerät bestimmt. Das Rückgabeergebnis enthält die Liste der Verbaulisteneinträge, welche jeweils Informationen zu Steuergerätename, -adresse, Knoten-ID, Datenbus-Identifier, Soll- und Istverbau enthalten.

IDiagResultComponentList **readSubsystemComponentList**(IConnectionHandle

connectionHandle)

* Die neue Funktion „writeSubsystemComponentList“ codiert einen benutzerdefinierten Sollverbau in das Steuergerät. Dazu kann eine Liste von Verbaulisteneinträgen übergeben werden, welche den neuen Sollverbau enthalten. Für die Zuordnung der Einträge wird standardmäßig die Knoten-ID und Steuergeräteadresse verwendet.

IDiagResult **writeSubsystemComponentList**(IConnectionHandle connectionHandle,

List<IDiagComponentListEntry> componentList)

* Das automatische Codieren wird über die neue Funktion “writeSubsystemComponent-ListFromPresentState” bereitgestellt. Hierbei wird der zuvor gelesene Ist-Verbau automatisch als Sollvorgabe verwendet. Die Funktion ist erfolgreich durchlaufen, wenn keine „DiagException“ geworfen wird.

IDiagResult **writeSubsystemComponentListFromPresentState**(IConnectionHandle

connectionHandle)

* Die neue Funktion „updateSubsystemComponentList“ steht nur an der Makro-Schnittstelle zur Verfügung und aktualisiert den bereits zuvor ausgelesenen Ist-Verbau.

IDiagResultComponentList **updateSubsystemComponentList**(IConnectionHandle

connectionHandle)

## Version 17.0

Folgende Änderungen wurden vorgenommen:

* Der Web-Service „setCanFDVehicleProject“ öffnet das übergebene Fahrzeugprojekt für CAN-FD. Sollte das Projekt nicht existieren, ein Diagnose-Interface (VCI) ohne CAN-FD-Unterstützung vorliegen oder ggfs. kein CAN-FD-Fahrzeug im Fahrzeugprojekt enthalten sein, so wird eine entsprechende DiagException geworfen. Dieser Aufruf ist unabhängig von den vorliegenden Admin-Einstellungen.

void **setCanFDVehicleProject**(String projectName)

* Der Web-Service „setCanFDVehicleProjectAutomatic“ führt einen automatischen CAN-FD-Diagnoseeinstieg durch, wobei das zu verwendende Fahrzeugprojekt mit Hilfe der VIN bestimmt wird. Dabei kann der Benutzer analog zur GUI entweder den „ODIS-Engineering“- oder „ODIS-Service“-Mode auswählen. Dieser Mode schaltet zwischen den konfigurierten Verzeichnispfaden für DIDB, Fahrzeugprojekte und VRT-/VPTs um. Des Weiteren findet im „ODIS-Service“-Mode die Projekt-VIN-Zuordnung über die VRT-VPTs und im „ODIS-Engineering“-Mode über eine Mapping-Datei statt. Sollte kein oder mehrere Projekte für die aktuelle VIN vorliegen, so führt dies zu einer DiagException. Entsprechend wird auch bei fehlendem CAN-FD-Diagnoseinterface oder fehlendem CAN-FD-Fahrzeug in dem Projekt eine Exception geworfen. Nach erfolgreichem CAN-FD-Diagnoseeinstieg liefert der Service das verwendete Fahrzeugprojekt zurück.

String **setCanFDVehicleProjectAutomatic**(PROJECT\_MODE projectMode)

## Version 18.0

Folgende Änderungen wurden vorgenommen:

* Das Rückgabeobjekt der Automatisierungsfunktion „dataSetDownload“ wurde um einen Fehlertext und eine Fehler-ID erweitert, damit eine detalliertere Fehlerbeschreibung möglich ist. Der neue Rückgabetyp „IDiagResultDatasetDownload“ erweitert den bisherigen Typ „IDiagResult“, wodurch die Abwärtskompatibilität gewahrt bleibt.

**IDiagResultDatasetDownload dataSetDownload(IConnectionHandle connectionHandle,**

**String datasetPath)**

Der Rückgabetyp „IDiagResultDatasetDownload“ bietet zusätzliche Funktionen zum Abfragen der Fehler-ID, des Fehlertextes und ob ein Fehler auftrat.

Dabei sind folgende Fehler-IDs möglich:

ODS3001E: Datensatz-Datei wurde nicht gefunden

ODS3006E: Falsches Dateiformat der Datensatz-Datei

ODS3014E: Fehler beim Entpacken der Datei

ODS3015E: Datensatz-Datei enthält keine Steuerungsdatei

ODS3021E: Fehlender Flash-Container

ODS3022E: Fehler beim Laden eines Flash-Containers

ODS6922E: Negatives Ergebnis bei Konsistenzprüfung

ODS6923E: Aktuelle Steuergeräteadresse nicht in Steuerungsdatei enthalten

ODS6999E: Fehler bei Ausführung des MCD-Jobs

* Für das Versenden von BZD-Protokollen existiert nun für die neue Versendeart AVx eine eigene Schnittstellenfunktion. Diese lautet:

IDiagResultProtocolTransfer **sendProtocolsToAVx**(<Protokollliste>)

Die zurückgelieferten Ergebniselemente 'IDiagResultProtocolTransferEntry' der einzelnen Protokolle enthält den Protokollnamen, den Versendestatus und ggfs. einen Hinweistext der zuvor aufgerufenen Versendeart.

* Für das Ermitteln von nicht erfüllten Flash-Vorbedingungen steht nun die Funktion „checkFlashPreConditions“ zur Verfügung. Diese Funktion ist unabhängig vom verwendeten Flash-Container und ließt die nicht eingehaltenen Vorbedingungen über den MCD-Service „DiagnServi\_­RoutiContr­Check­ProgrPreco“ aus dem übergebenen Steuergerät. Die Rückagbeliste enthält für jede nicht erfüllte Vorbedingung ein „[IDiagResultPreCondition](#r241)“-Objekt, welches zusätzlich zu dem bisher verfügbaren übersetzten Text auch eine Text-ID enthält.

List<[IDiagResultPreCondition](#r241)> **checkFlashPreConditions**(<[IConnectionHandle](#r33)>)

Bei den Funkionen „flashProgramming“, „flashProgrammingParallel“ und „flashProgrammingSequence“ werden nun vor jeder Flash-Durchführung die Vorbedingungen überprüft. Sollten nicht erfüllte Vorbedingungen vorliegen, so wird der Flash-Vorgang für dieses Steuergerät nicht weiter fortgesetzt. Die Rückgabe enthält ein entsprechendes Ergebnisobjekt mit der Liste von nicht erfüllten Vorbedingungen. Die nach außen sichtbaren Schnittstellenfunktionen bleiben unverändert.

## Version 19.0

Folgende Änderungen wurden vorgenommen:

* Für die SFD-Freischaltung eines Steuergerätes über einen Online-Token steht die API-Funktion „securityAccessSFDOnline“ zur Verfügung. Diese Funktion ist über die Web-Schnittstelle nur vom lokalen PC aufrufbar, ansonsten wird eine entsprechende Exception geworfen. Die Ermittlung des Online-Tokens findet über einen SFD-Backend-System statt. Über die Aufrufparameter kann die Freischaltrolle sowie -dauer vorgegeben werden. Das Rückagbeelement „IDiagResultSFD“ enthält evtl. aufgetretene Fehler oder NRCs inklusive der Error-IDs.

IDiagResultSFD **securityAccessSFDOnline**(IConnectionHandle connectionHandle, OnlineRequestedRole role, OnlineAccessDuration duration, OnlineAccessMethod accessMethod)

Mögliche Fehlermeldungen innerhalb von „IDiagResultSFD”:

ODS1003E: Keine Verbindung zum Backend-System

ODS1007E: Negative Antwort vom Server

ODS2003E: Login-Methode wird nicht unterstützt

ODS6912E: Keine funktionale Steuergeräteantwort

ODS6913E: Fehlerhaftes SFD-Ergebnis

ODS6914E: Unbekannte SFD-Rolle

ODS6915E: Fehlerhafte SFD-Freischaltung

ODS6916E: Fehlerhafter SFD-Reset

ODS6917E: Kein Offline-Token gefunden

ODS6918E: Name des Offline-Tokens enthält keinen Zeitstempel

ODS6924E: Fehlerhafte SFD-Offline-Freischaltung (Wiedereinspielschutz)

* Für die SFD-Freischaltung eines Steuergerätes mit einem Offline-Token steht die API-Funktion „securityAccessSFDOffline“ zur Verfügung. Im Unterschied zum Online-Token muss hier ein entsprechender Token auf dem lokalen PC vorhanden sein. Aus diesem Grund findet hier auch keine Anfrage an das SFD-Backend-System statt. Ansonsten entsprechen die Aufruf- und Antwortparameter dem der Online-Freischaltung. Ebenso sind nur Aufrufe vom lokalen PC erlaubt.

IDiagResultSFD **securityAccessSFDOffline**(IConnectionHandle connectionHandle, OnlineRequestedRole role, OnlineAccessDuration duration)

* Der Web-Service „securityAccessSFDReset“ setzt das Steuergerät auf den SFD-Werkszustand zurück. Der Aufruf und das Ergebnis sind analog zur Online-Freischaltung, wodurch auch hier eine Verbindung zum SFD-Backend-System notwendig ist.

IDiagResultSFD **securityAccessSFDReset**(IConnectionHandle connectionHandle, OnlineRequestedRole role, OnlineAccessDuration duration, OnlineAccessMethod accessMethod)

* Für die Abfrage vom SFD-Status eines Steuergerätes steht die API-Funktion „securityAccessSFDCheckStatus“ bereit und ist im Vergleich zu den vorherigen API-Funktionen auch von entfernten PCs aufrufbar. Das Ergebnis „IDiagResultSFDStatus“ enthält die aktuelle Freischaltdauer, -rolle, -typ und -status.

IDiagResultSFDStatus **securityAccessSFDCheckStatus**(IConnectionHandle connectionHandle)

* Mittels der beiden folgenden API-Funktionen können SFD-Steuergeräte gesperrt werden. Über die Funktion „securityAccessSFDEcuLock“ kann ein einzelnes Steuergerät und über „securityAccessSFDAllEcuLock“ alle Steuergeräte gleichzeitig gesperrt werden. Die Rückgabeergebnis enthält für jedes antwortende Steuergerät ein separates Ergebnisobjekt.

IDiagResultSFD **securityAccessSFDEcuLock**(IConnectionHandle connectionHandle)

List<IDiagResultSFD> **securityAccessSFDAllEcuLock**()

* Die folgenden Makro-Methoden zum Erstellen von Grundeinstellungen und Stellglieddiagnosen stehen nun auch an der WEB-API zur Verfügung:

IDiagDescriptorWithParametersAndMeasurements **createDiagActuator**( String actuatorName, Map<String, String> parameters, List<String> measurementValues)

IDiagDescriptor2WithParametersAndMeasurements **createDiagActuator2**( String actuatorName, Map<String, String> parameters, List<IDiagMeasurementDescriptor> measurementValues)

IDiagDescriptorWithParametersAndMeasurements **createBasicSetting**( String basicSettingName, Map<String, String> parameters, List<String> measurements)

IDiagDescriptor2WithParametersAndMeasurements **createBasicSetting2**( String basicSettingName, Map<String, String> parameters, List<IDiagMeasurementDescriptor> measurements)

IDiagMeasurementDescriptor **createDiagMeasurementDescriptor**(long ecuAddress, String measurementGroup, String measurementParam)

## Version 19.2

Folgende Änderungen wurden vorgenommen:

* Die Methode zum Erstellen von BZD-/VDS-Protokollen wurde um ein zusätzliches optionales Argument für Identifikationspresets erweitert:

String **createProtocolVDSBZD**(<VIN>, <Fzg-Nummer>, <readEventMemory>, <Benutzername>, <Messort>, <Messtyp>, <Kilometerstand>, <Einheit Kilometerstand>, <Anpassungsliste>, <Messwertliste>, <Identifikationsliste>, <Zusätzliche Presets>)

* Für die Ermittlung der Presets zur erweiterten Identifikation wird nun die folgende Methode bereitgestellt:

List<String> **getAvailableIdentificationPresets**()

* Die Methode der Automatisierungsschnittstelle (nicht WEB-API) zum auslesen von Ereignisspeichereinträgen wurde um ein neues optionales Argument erweitert:

List<IDiagResultEventMemory> **readEventMemories**(<SG-Liste>, <Anpassungsliste>, <Messwertliste>, <Identifikationsliste>, <Zusätzliche Presets>)

## Version 20.0

Folgende Änderungen wurden vorgenommen:

* Zum Ausführen von Kurzläufer-Routinen eines Steuergerätes wurde eine Methode hinzugefügt. Diese erwartet das Verbindungsobjekt zum Steuergerät, den Namen der auszuführenden Routine und eine Map in der zu jedem Parameter der Routine ein Wert hinterlegt ist:

IDiagResultStatus **immediateRoutineExecution**(IConnectionHandle connectionHandle, String immediateRoutineName, Map<String, String> immediateRoutineParameters)

* Eine Methode zum Ermitteln der Namen aller Parameter einer Kurzläufer-Routinen wurde hinzugefügt:

List<String> **getShortNamesOfParametersForImmediateRoutine**(IConnectionHandle connectionHandle, String immediateRoutineName)

* Eine Methode zum ermitteln der verfügbaren Kurzläufer-Routinen eines Steuergerätes wurde hinzugefügt:

List<String> **getShortNamesOfImmediateRoutines** (IConnectionHandle connectionHandle)

## Version 20.1

Folgende Änderungen wurden vorgenommen:

* Standardmäßig ist ab dieser Version das Auslesen der Gatewayverbauliste anstelle des Maxverbaus aktiv und wirkt sich unter anderem auf das erstellen des BZD protokolls aus (siehe auch Einstellung „Funktionskonfiguration -> Steuergeräteliste -> Auswertung Gateway Verbauliste“, im Kapitel 5.2.2)
* Das Auslesen der Gatewayverbauliste oder des Maxverbaus kann mit folgender Funktion konfiguriert werden:

void **configureSetting** (String paramKey, String paramValue) throws DiagException

Dazu wurde der neue Parameter „DiagEntrance.GatewayComponentList“ eingeführt. Als Wert kann FALSE (d.h. verwenden des Maxverbaus) oder TRUE (verwenden der Gatewayverbauliste) angegeben werden

## Version 20.2

Folgende Änderungen wurden vorgenommen:

* Zum Flashen unter Verwendung eine Flashreihenfolgedatei im json Format wurde folgende neue Funktion eingeführt:

IDiagResultFlashProgramming **flashProgrammingCoordinated** (List <IDiagFlashSessionDescriptor> flashSessionDescriptors, String coordinationFile)

* Die folgende Funktion zum Flashen unter Verwendung eine Flashreihenfolgedatei im json Format wurde nur für die Macro Schnittstelle hnzugefügt:

IDiagResultFlashProgramming **flashProgrammingCoordinated** (List <IDiagFlashSessionDescriptor> flashSessionDescriptors, List <String> coordinationFiles)

## Version 21.0

Folgende Änderungen wurden vorgenommen:

### Entwickler-Ereignisspeicher

* Für das Lesen der Entwickler-Ereignisspeicher optional mit Snapshot Records wurde folgende Funktion um zusätzliches Parameter readSnapshotRecords erweitert:

IDiagResultEventMemory **readDevEventMemory**(IConnectionHandle connectionHandle,

List<String> memorySelections,

Long dtcStatusMask,

Boolean readSnapshotRecords)

* Für das Ermitteln der Anzahl von verfügbaren Entwickler-Ereignisspeicher MemorySnapshots wird folgende Funktion bereitgestellt:

Integer **getNumberOfDevEventMemorySnapshots**(IConnectionHandle connectionHandle,

List<String> memorySelections,

Integer eventCode)

* Für das Ermitteln der verfügbaren MemorySnapshots für ein bestimmtes Entwickler-DTC wird folgende Funktion bereitgestellt:

IDiagResultDtcSnapshotList **readDevEventMemorySnapshots**(

IConnectionHandle connectionHandle,

List<String> memorySelections,

Integer eventCode)

List<String> **readAvailableDevEventMemorySelection**(IConnectionHandle connectionHandle)

* Für den Export aller Entwickler-DTC Snapshot Records eines Steuergerätes wird die Funktion „exportDevEventMemorySnapshots” bereitgestellt. Sollten zum Export-Zeitpunkt einige Snapshot Records fehlen, so werden diese ermittelt, um ein vollständiges Export-Protokoll zu gewährleisten. Als Zielpfad kann ein leerer Pfad, ein Verzeichnis oder kompletter Dateipfad angegeben werden. Fehlende Pfadelemente werden durch Default-Werte der Anwendung ersetzt. Die Funktion liefert den vollständigen, verwendeten Ablagepfad zurück. Zusätzlich zur XML Datei wird eine gleichnamige HTML Datei erstellt.

String **exportDevEventMemorySnapshots**(IConnectionHandle connectionHandle,

List<String> memorySelections,

String targetFilePath)

### Software Compositions / Software Components

#### Identifikation

* Die Methode readIdentification wurde um Software-Compositions und Software-Components erweitert. Die Software-Components werden als Kind-Elemente ihrer zugehörigen Software-Composition aufgeführt. Die Elemente auf oberster Ebene enthalten die Typinformation des Elementes (MASTER, SLAVE, SW\_CLUSTER, SW\_COMPOSITION, SW\_COMPONENT).
* Die Methode readMultilinkIdentification wurde um Software-Compositions und Software-Components erweitert.
* Methode getShortNamesOfSoftwareCompositions zum Lesen aller Namen der (codierten) Software-Compositions und Software-Components. Es wird eine flache Liste geliefert, in der die Software-Components nach ihren zugehörigen Software-Compositions stehen. Die Einträge enthalten nebem dem Shortname auch die Typinformation SW\_COMPOSITION oder SW\_COMPONENT.
* Die Methode readIdentificationSoftwareCompositions liefert nur den Teilbaum von readIdentification, der das Master Steuergerät und alle seine Software-Compositions (mit Software-Components) enthält.

#### Codierung

* Die Methode createSoftwareCompositionComponentListEntry ist eine Hilfsmethode, um Eingabeparameter für das Schreiben der Software-Composition Codierung zu erzeugen (steht nur an der Makro-Schnittstelle zur Verfügung).
* Mittels readSoftwareCompositionComponentList wird der Codierstatus und Zusatzinformationen zu allen Software-Compositions des übergebenen Steuergerätes gelesen. Die Rückgabestruktur enthält zusätzlich zu allen Software-Compositions deren Software-Components.
* updateSoftwareCompositionComponentList liesst nur den Codierstatus neu und liefert die gleiche Ergebnisstruktur wie readSoftwareCompositionComponentList (steht nur an der Makro-Schnittstelle zur Verfügung).
* Mittels writeSoftwareCompositionComponentList erfolgt die Umcodierung der Software-Compositions.
* writeSoftwareCompositionComponentListFromPresentState setzt alle Software-Compositions auf codiert.

Ausschluss:

* Die Methode readMultilinkCoding liefert (auch bei Schalterstellung readSubsystems) keine SoftwareCompositions / SoftwareComponents.

## Version 21.1

Folgende Änderungen wurden vorgenommen:

* Die folgende Methode zum Setzen / Rücksetzen der Activation-Line eines festgelegten, ereichbaren VCIs (welches DoIP unterstützt) wurde hinzugefügt:

void **setDoIPActivationLine**(boolean toSet) throws DiagException

## Version 22

Die SFD Ende zu Ende Absicherung wird in folgenden schreibenden UDS Methoden für die Web- und Makroschnittstelle durchgeführt:

* IDiagResult **writeAdaptation**(IConnectionHandle connectionHandle, String adaptationChannel, Map<String, String> adaptationValues)
* IDiagResult **writeByteCoding**(IConnectionHandle connectionHandle, String systemName, String codingValues)
* IDiagResult **writeTextCoding**(IConnectionHandle connectionHandle, String systemName, Map<String, String> codingValues)
* IDiagResult **writeComponentListFromPresentState**(IConnectionHandle connectionHandle)
* IDiagResult **writeGatewayComponentList**(IConnectionHandle connectionHandle, List<IDiagComponentListEntry> componentList)
* IDiagResult **writeSoftwareCompositionComponentListFromPresentState**(IConnectionHandle connectionHandle)
* IDiagResult **writeSoftwareCompositionComponentList**(IConnectionHandle connectionHandle, List<IDiagSoftwareCompositionComponentListEntry> componentList)
* IDiagResult **writeSubsystemComponentListFromPresentState**(IConnectionHandle connectionHandle
* IDiagResult **writeSubsystemComponentList**(IConnectionHandle connectionHandle, List<IDiagComponentListEntry> componentList)

Ebenso werden folgende, nur an der Makroschnittelle verfügbaren Methoden unterstützt:

* IDiagResult **writeByteCoding**(IConnectionHandle connectionHandle, String systemName, String codingValues, **boolean** ecuReset, **int** checksum)
* IDiagResult **writeTextCoding**(IConnectionHandle connectionHandle, String systemName, Map<String, String> codingValues, **boolean** ecuReset)

Die Signaturen der Methoden wurden bzgl. der SFD Ende zu Ende Absicherung nicht angepasst. Die Schaltung zwischen online und offline Verhalten wird mittels der Methode configureSetting durchgeführt. Der Default ist online ('SFD.EndToEndProtectionModeOffline', 'FALSE'). Die Einstellung gilt solange, bis sie geändert wird, längstens bis zum Laden eines Projektes, das zum Setzen des Defaults führt.

Im Anwendungsfall „Offline-Freischaltung“ wird nicht versucht über eine Backendverbindung den benötigten Token zu bekommen. Es wird davon ausgegangen das ein solcher für die DIDs und die zu schreibenden Werte im Vorfeld über die GUI abgerufen und gespeichert wurde.

Bei der Suche nach dem zum Schreibrequest passenden Token wird abhängig von der Eigendiagnosefunktionen in folgenden festen Verzeichnissen unterhalb des Installationsverzeichnisses gesucht:

|  |  |
| --- | --- |
| **Verzeichnis** | **Token Verwendung** |
| \SFD\_Token\E2E\_TOKEN\_ADAPTATION | Anpassungen |
| \SFD\_Token\E2E\_TOKEN\_BUSMASTER\_CODING | Busmastercodierung (RDID "04A3" bzw. "01AC") |
| \SFD\_Token\E2E\_TOKEN\_CODING | Text und Binärcodierung (RDID "0600" im Falle Master ansonsten "0600" Subsystemindex + 0x600) |
| SFD\_Token\E2E\_TOKEN\_SOFTWARE\_COMPOSITION\_CODING | Codierung Software-Compositions (RDID "0442") |
| \SFD\_Token\E2E\_TOKEN\_SUBSYSTEM\_CODING | Codierung von Subsystemen (RDID "061A") |

Tabelle : Zuordnung Token zu Verzeichnissen

Der Name der Token wird nicht ausgewertet, er muss aber auf ".e2etoken" enden. Es werden alle Token des entsprechenden Verzeichnisses durchgegangen, bis der erste passende gefunden wurde. Dabei wird keine Reihenfolge garantiert. Der gefundene Token wird dann verwendet und nach der Verwendung gelöscht.

Wurde kein „passender“ Token gefunden, so wird die aufrufende Methode mit der Exception ODS6941E beendet.

Besonderheiten bei der Bestimmung des passenden Tokens:

* Der Token muss genau alle schreibenden Daten enthalten. Wenn er noch zusätzliche Daten enthält, die nicht geschrieben werden, wird es als nicht „passend“ klassifiziert. Es ist entscheidend, was letztendlich in dem Schreibrequest an Daten enthalten ist. Genau dazu muss es einen passenden Token geben.
* Wenn bei Anpassungen mehrere Kanäle gleichzeitig für ein „Übernehmen“ geändert werden, wird im Makro für jeden Kanal ein Schreibbefehl erzeugt, da man über Web / Makro mit einem Befehl nur jeweils einen Kanal schreiben kann. Demzufolge darf es nicht ein Token für alle Kanäle geben, sondern zu jedem Kanal einen zugehörigen Token.

Im Anwendungsfall „Online-Freischaltung“ wird von einer bestehenden Backendverbindung ausgegangen. Liegt diese nicht vor, wird der jeweilige Aufruf der Automatisierungsschnittstelle mit der Exception ODS1003E beendet.

Die „Online-Freischaltung“ wird intern identisch wie der interaktive Ablauf behandelt (bis natürlich auf Besonderheiten der Automatisierung, wie die Unterdrückung von Benutzerinteraktionen und ähnlichem).

Als Anmeldemethode im Anwendungsfall „Online-Freischaltung“ wird fest „PKI-Ausweis“ benutzt. Die Anmeldemethode kann nicht geändert werden. Wenn vor dem Abspielen eines Makros die Anmeldemethode „Secure-ID“ verwendet wurde, wird diese nach dem Abspielen nicht automatisch wieder als Default gesetzt. Dies muss vom Anwender manuell erfolgen.

## Version 22.0

Folgende Änderungen wurden vorgenommen:

Die folgende Methode zum Flashen von zusammenhängenden Systemen über ein ZIP-Archiv wurde hinzugefügt:

[IDiagResultFlashProgramming](#r196) [**flashProgrammingCoordinatedByZipArchive**](#r597)(

String coordinationZipFilePath)

Als Übergabeparameter wird der Pfad des ZIP-Archives übergeben. Das ZIP-Archiv muss mindestens eine JSON-Steuerungsdatei enthalten.

## Version 23.0

Folgende neue Automatisierungsfunktionen werden bereitgestellt:

* Die Methode createSoftwareCompositionComponentCodingDescriptor ist eine Hilfsmethode, um Eingabeparameter für das Schreiben der Komponentenliste von Software-Composition im Kontext der Multilink-Ausführung für ein bestimmtes Steuergerät zu.

IDiagMultiSoftwareCompositionComponentCodingDescriptor **createSoftwareCompositionComponentCodingDescriptor**(IMultilinkEcuHandle controlUnit,

List<IDiagSoftwareCompositionComponentListEntry> compositionComponents)

* Die Methode readMultilinkSoftwareCompositionComponentList liest die Komponentenliste der Software-Compositions für mehrere Steuergeräte über Multilink. Die Liste der auszulesenden Steuergeräte wird über eine Liste von IMultilinkEcuHandle übergeben.

List<IDiagResultSoftwareCompositionComponentList> **readMultilinkSoftwareCompositionComponentList**(List<IMultilinkEcuHandle> ecuList)

* Die Methode writeMultilinkSoftwareCompositionComponentList schreibt die Komponentenliste der Software-Compositions gleichzeitig für mehrere Steuergeräte über Multilink. Die Liste der auszulesenden Steuergeräte inklusive der kodierten Kompoenten wird der Funktion als Liste übergeben.

List<IDiagResultMulti> **writeMultilinkSoftwareCompositionComponentList**(

List<IDiagMultiSoftwareCompositionComponentCodingDescriptor> codingDescriptors)

## Version 23.1

Folgende neue Schnittstellenerweiterungen werden bereitgestellt:

* Die Methode getAvailableDatasetsOnEcu ermittelt alle Datensätze, welche in einem Steuergerät verfügbar sind. Dazu muss das jeweilige IConnectionHandle vom Steuergerät der Methode als Argument übergeben werden. Die ermittelten Datensätze werden als Liste zurückgeliefert, wobei jedes Element die Informationen zu einem Datensatz enthält. Diese Elemente können direkt als Eingabeparameter für die Upload-Methode datasetUpload verwendet werden.

List<IDiagDatasetDescriptor> **getAvailableDatasetsOnEcu**(

IConnectionHandle connectionHandle)

* Die Methode datasetUpload führt den Upload vom Steuergerät zum Tester für jeden der übergebenen Datensatzbeschreibung durch. Über das Argument destPath kann das Zielverzeichnis vorgegeben werden. Falls der Pfad leer ist, wird das Standardablageverzeichnis aus der Administration verwendet.

List<IDiagResultDatasetUpload> **datasetUpload**(

IConnectionHandle connectionHandle,

List<IDiagDatasetDescriptor> dataBlocks, String destPath)

* Die Methode createDatasetDescriptor erzeugt eine Datensatzbeschreibung, die für den Upload verwendet werden kann. Für die korrekte Auswahl/Zuordnung muss mindestens einer der beiden Angaben von Adresse oder Name vorliegen. Über das Argument additionalFileName kann ein benutzerdefinierter Zusatzname für die Zieldatei definiert werden. Sollte dieser leer sein, so wird anstattdessen das aktuelle Datum verwendet.

IDiagDatasetDescriptor **createDatasetDescriptor**(String datasetAddress,

String datasetName, String additionalFileName)

## Version 24.0

Die Methode configureSetting unterstützt ab API-Version 24.0 einen weiteren Konfigurationsparameter für die Ende-zu-Ende-Absicherung über Security-Tokens (SFD2). Über den Parameterschlüssel SFDE2E.AuthMethod kann die Anmeldemethode für die Authentifizierung am SFD-Backend bei einer Ende-zu-Ende-Absicherung konfiguriert werden. Als gültige Werte stehen PKI oder SoftPSE zur Verfügung. Beim Wert SoftPSE wird zusätzlich die Vollständigkeit der zugehörigen Zertifikatskonfiguration überprüft. Sollte diese nicht vollständig sein oder ein ungültiger Parameterwert übergeben werden, so liefert der Aufruf ein DiagException zurück.

Für die Freischaltung über SFD-Tokens bietet nun OnlineRequestedRole die zusätzliche Rolle EPTI und OnlineAccessMethod die zusätzliche Anmeldemethode SoftPSE. Diese Parameter können bei den API-Funktionen securityAccessSFDOnline, securityAccessSFDReset, securityAccessSFDOffline und securityAccessSFDAllOnline verwendet werden.

Folgende neue Schnittstellenerweiterungen werden bereitgestellt:

* Die Methode getSFDLinkNames ermittelt alle Steuergeräte, welche die SFD-Freischaltung über Tokens unterstützen. Die ermittelten Steuergeräte werden als Liste von IDiagResultEcu zurückgeliefert, wobei diese Ergebnisstruktur den LogicalLink-, Short- und übersetzten Namen sowie die Diagnoseadresse enthält.

List<IDiagResultEcu> **getSFDLinkNames**()

* Die Methode securityAccessSFDAllOnline führt eine SFD-Freischaltung aller Steuergeräte im Fahrzeug durch, welche die SFD-Freischaltung über Tokens unterstützen. Dazu kann die Rolle, die Freischaltdauer sowie die verwendete Zugriffsmethode zur Authentifizierung am Backend (PKI oder SoftPSE) übergeben werden. Das Ergebnisliste enthält für jedes SFD-Steuergerät die Ergebnisstruktur IDiagResultSFD, die evtl. aufgetretene Fehler oder NRCs inklusive der Error-IDs beinhaltet (siehe hierzu Kapitel 6.13).

List<IDiagResultSFD> **securityAccessSFDAllOnline**(OnlineRequestedRole role,

OnlineAccessDuration duration,

OnlineAccessMethod accessMethod

Die Methoden

* byte[] **sendRawService**(IConnectionHandle connectionHandle,

byte[] requestPdu

* byte[] **sendRawService**(IConnectionHandle connectionHandle,

String requestPduString)

unterstützen ab API-Version 24.0 jetzt die automatische SFD-E2E-Freischaltung. Die SFD-E2E-Freischaltung über die Methode sendRawService erfolgt dabei ausschließlich bei 0x2E-Aufrufen.

Im Offline-Fall wird bei der Suche nach dem zum Schreib-Request passenden Token für die Methode sendRawService in den Ordnern für Anpassung und Codierung gesucht, d.h. in den Ordnern \SFD\_Token\E2E\_TOKEN\_ADAPTATION und \SFD\_Token\E2E\_TOKEN\_CODING. Damit ein vorhandener Offline-Token verwendet wird, muss er genau den gleichen Request und dieselbe Steuergeräteadresse enthalten, wie das Steuergerät, für den der Schreib-Request ausgeführt werden soll.

Bei Ausführung der Methode

* String **createProtocolVDSBZD**(<VIN>, <Fzg-Nummer>, <readEventMemory>, <Benutzername>,

<Messort>, <Messtyp>, <Kilometerstand>, <Einheit Kilometerstand>, <Anpassungsliste>, <Messwertliste>, <Identifikationsliste>, <Zusätzliche Presets>)

werden beim Auslesen der Ereigsnisspeicherinformationen jetzt auch die Snapshotdaten ermittelt und dem Gesamtfahrzeugprotokoll hinzugefügt.

Bei den Methoden

* IDiagResultSFD **securityAccessSFDOnline**(IConnectionHandle connectionHandle, OnlineRequestedRole role, OnlineAccessDuration duration, OnlineAccessMethod accessMethod)
* List<IDiagResultSFD> **securityAccessSFDAllOnline**(OnlineRequestedRole role, OnlineAccessDuration duration, OnlineAccessMethod accessMethod

wird bei einem Fahrzeug aus der Kooperation Ford, d.h. ein Fahrzeug dessen Fahrgestellnummer mit "WF0" beginnt, automatisch die Role OnlineAccessDuration.BASIC und die Freischaltdauer OnlineAccessDuration.SHORT verwendet, unabhängig von den übergebenen Parameterwerten. Desweiteren wird für die Erstellung des Backend-Requests das Markenkürzel „F“ verwendet, unabhängig von der aktuell ermittelten Marke.

## Version 24.1

Ab API-Version 24.1 stehen drei neue Funktionen für die Eigendiagnosefunktion „Multiple Identifikationsdaten“ zur Verfügung. Die gelesenen Identifikationsdaten werden innerhalb des Rückgabeelementes IDiagResultIdentification geliefert. Dieses Element enthält die Standard- sowie die ermittelten multiplen Identifikationsdaten. Die Standardidentifikationswerte sind jedoch nur enthalten, wenn diese bereits durch einen vorherigen Diagnoseaufruf gelesen wurden. Diese Daten können mit den Funktionen getStandardData() bzw. getMultipleData() am Ergebniselement abgefragt werden. Die multiplen Daten werden als eine Liste von IDiagResultMultipleIdentValue zurückgeliefert. Dieser Datentyp enthält neben den üblichen Abfragewerten des Elementes IDiagResultValue zusätzlich noch die Blocknummer. Die Blocknummer gibt die Nummer des Diagnoseaufrufes an, mit dem der Identifikationswert ermittelt wurde. Der erste Aufruf beginnt mit der Blocknummer 1 und kann maximal 8 Identifikationswerte umfassen. Der 9. Identifikationswert würde somit die Blocknummer 2 enthalten u.s.w..

Diese Funktionen werden nur von UDS-Steuergeräten unterstützt, welche den MCD-Service DiagnServi\_ReadDataByIdentECUIdentMulti in der ODX-Bedatung enthalten. In allen anderen Fällen wird eine DiagException geworfen. Ebenso führt ein nicht unterstützter, multipler Identifikationsname zu einer entsprechenden DiagException.

Folgende neue Schnittstellenerweiterungen werden bereitgestellt:

* Die Methode getShortNamesOfMultipleIdentificationDatas ermittelt alle unterstützten multiplen Identifikationsnamen für das aktuelle Steuergerät, welches über das Argument connectionHandle übergeben wird. Die Namen werden als String-Liste zurückgeliefert und können beim Lesen einer definierten Auswahl von multiplen Daten über die API-Funtkion readMultipleIdentificationData genutzt werden. Der Listeninhalt ist abhängig von den ODX-Daten des geladenen Fahrzeugs.

List<String> **getShortNamesOfMultipleIdentificationDatas**(

IConnectionHandle connectionHandle)

* Die Methode readMultipleIdentification liest alle multiplen Identifikationswerte für ein bestimmtes Steuergerät. Dieses Steuergerät wird über das Argument connectionHandle festgelegt. Die Liste umfasst alle multiplen Daten aus dem MCD-Service DiagnServi\_ReadDataByIdentECUIdentMulti der ODX-Bedatung. Die Ergebnisse werden, wie oben beschrieben, über das Rückgabeelement IDiagResultIdentification geliefert.

IDiagResultIdentification **readMultipleIdentification**(

IConnectionHandle connectionHandle)

* Über die Methode readMultipleIdentificationData kann eine definierte Anzahl von multiplen Identifikationsdaten für ein bestimmtes Steuergerät ermittelt werden. Dazu werden die beiden Argumente connectionHandle für das Steuergerät sowie multipleIdentDataIdentifiers für die Liste der auszulesenden Identifikationswerte benötigt. Für die Liste der auszulesenden Identifikationswerte sind alle Namen erlaubt, welche über die API-Funktion getShortNamesOfMultipleIdentificationDatas für das jeweilige Steuergerät geliefert werden. Die Ergebnisse werden, wie oben beschrieben, über das Rückgabeelement IDiagResultIdentification geliefert.

IDiagResultIdentification **readMultipleIdentificationData**(

IConnectionHandle connectionHandle,

List<String> multipleIdentDataIdentifiers)

## Version 24.3

Ab API-Version 24.3 wird die Eigendiagnosefunktion „Fahrzeugfunktionen“ unterstützt. Über die Routine   
Request\_systems\_involved\_in\_function (Abfrage ECU-IDs für eine Funktion) können alle Funktionen einzelner SWCL / SWC / SWCOs ermittelt werden.

Folgende neue Schnittstellenerweiterungen werden bereitgestellt:

* Die Methode getVehicleFunctions liest alle über die verbauten Fahrzeugfunktioen aus und stellt die Ergebnisse über den Rückgabewert List<IVehicleFunction> zur Verfügung.

List<IVehicleFunction> getVehicleFunctions() throws DiagException

* Elemente vom Datentyp IVehicleFunction bieten folgende Abfragen:
  + getFunctionNumber()  
    liefert die Fahrzeugfunktions-ID als hexadezimalen String
  + getFunctionName()   
    liefert den Fahrzeugfunktions-Namen als String
  + getFunctionVersion()   
    liefert die Fahrzeugfunktions-Version
  + getFunctionMaster()   
    liefert den Fahrzeugfunktions-Master als Element vom neuen Typ   
    IVehicleFunctionInvolvedSystem
  + getInvolvedSystems()   
    liefert die an der Fahrzeugfunktion beteiligten Systeme als Liste von Elementen des   
    neuen Typs IVehicleFunctionInvolvedSystem
  + getFunctionStatus()   
    liefert den Status der Fahrzeugfunktion als Element des neuen Typs  
    IVehicleFunctionStatusInfo
* Elemente vom Datentyp IVehicleFunctionInvolvedSystem stellen ein System (ECU oder SWCO) dar und bieten folgende Abfragen:
  + getId()   
    liefert die ID des Systems in hexadezimaler Form   
    (2 Byte für ECU, 8 Byte für SWCO)
  + getName()   
    liefert den Namen des Systems als String
  + getInvolvesSystemType()   
    liefert den Typ des beteiligten Systems als Element des neuen Aufzählungstyps   
    Systemtype. Dieser ist eingebettet in IVehicleFunctionInvolvedSystem und hat die Elemente „ECU“ und „SWCO“.
* Elemente vom Datentyp IVehicleFunctionStatusInfo bieten Methoden zum Abfragen einzelner festgelegter Status-Bits. Die StatusBits werden dabei geliefert als Elemente des neuen Typs IBitValue, der in IVehicleFunctionStatusInfo eingebettet ist.  
  Ein StatusBit bietet folgende Abfragen:
  + getBit()   
    lefert den Bit-Wert als „bool“ (true, wenn das Bit gesetzt ist, sonst false)
  + getTextValue()   
    liefert die Textdarstellung des Bit-Wertes
  + getTranslatedValue()   
    liefert den übersetzten Text des Bit-Wertes (i.W. zur Anzeige in der GUI)

## Version 24.4

Ab API-Version 24.4 steht die Konfiguration des partiellen Flashen über die bereits bestehende API-Funktionen configureSetting zur Verfügung. Dazu ist der Eingabeparameter paramKey mit dem Wert „Flashing.partialMode“ und der Parameter paramValue entweder mit „TRUE“ oder „FALSE“ zu setzen. Der Wert „TRUE“ aktiviert und „FALSE“ deaktiviert das partielle Flashen. Standardmäßig ist das partielle Flashen aktiviert. Die Konfigurieration bleibt für die Dauer der Server-Instanz erhalten, so dass nach einem Server-Neustart immer das partielle Flashen aktiv ist.

void **configureSetting**(String paramKey, String paramValue)

## Version 24.5

Aufgrund von Problemen bei einigen Web-Clients wurde ab API-Version 24.5 die WSDL-Datei geringfügig bei den Statusinformationen von „Fahrzeugfunktionen“ angepasst. Die Automatisierungs-API bleibt dagegen unverändert. Somit stehen beim Element „IVehicleFunctionStatusInfo“ weiterhin die Methoden „getStateBit()“, „getCalibrationBit()“, „getActivationBit()“, „getCommissioningBit()“, „getCommissioningStatusBit()“, „getStatusBit()“ und „getRegistrationBit()“ zur Abfrage er Statusinfomationen zur Verfügung. Die WSDL/Schema ist nun im Bereich der Statusinformationen folgendermaßen aufgebaut:

<xs:complexType name="vehicleFunctionStatusInfoImpl">

<xs:sequence>

<xs:element name="stateBit" type="tns:bitValueImpl"/>

<xs:element name="calibrationBit" type="tns:bitValueImpl"/>

<xs:element name="activationBit" type="tns:bitValueImpl"/>

<xs:element name="commissioningBit" type="tns:bitValueImpl"/>

<xs:element name="commissioningStatusBit" type="tns:bitValueImpl"/>

<xs:element name="statusBit" type="tns:bitValueImpl"/>

<xs:element name="registrationBit" type="tns:bitValueImpl"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="bitValueImpl">

<xs:sequence>

<xs:element name="itsBitValue" type="xs:boolean"/>

<xs:element name="itsTextValue" type="xs:string"/>

<xs:element name="itsTranslatedText" type="xs:string"/>

</xs:sequence>

</xs:complexType>

## Version 25.0

Aus der Webschnittstelle wurde die Versionsnummer aus der URL entfernt. Damit braucht bei einer abwärtskompatiblen Änderung kein neues Plateau-Plugin generiert und installiert werden. Weiterhin brauchen Tools, die die WebSchnittstelle verwenden, bei abwärtskompatiblen Änderungen die Schnittstellen-WSDL nicht neu einlesen und aufwendige Anpassungen vornehmen.

## Version 25.1

Wenn beim Diagnoseeinstieg zu dem zu verwendenen MCD-Projekt mehr als ein Vehicle-Info für den ausgewählten Kommunikationsweg vorliegt, so wird das Standard-Vehicle-Info verwendet, falls dies existiert. Liegt kein solches vor, wird das erste ermittelte Vehicle-Info verwendet.

Das Standard-Vehicle-Info für ein MCD-Projekt <Projektname> ist für die unterschiedlichen Kommunikationswege wie folgt definiert:

|  |  |
| --- | --- |
| **Kommunikationsweg** | **Standard-Vehicle-Info** |
| CAN | VINFO\_<Projektname>CAN |
| CAN-FD | VINFO\_<Projektname>CANFD |
| DoIP | VINFO\_<Projektname>DoIP |

Tabelle : Zuordnung Standard-Vehicle-Info zu Kommunikationsweg

Wenn ein definiertes anderes Vehicle-Info für den nächsten Einstieg genutzt werden soll, kann dies mit der Funktion setVehicleInfo festgelegt werden. Die Menge der verfügbaren Vehicle-Infos für ein MCD-Projekt kann mit getVehicleInfos abgefragt werden.

Folgende neue Schnittstellenerweiterungen werden bereitgestellt:

* Die Methode getVehicleInfos liefert alle Vehicle-Infos (Namen) zu einem Projekt. Sie ist nur im Zustand **DiagnosticInterfaceInit** aufrufbar.

List<String> **getVehicleInfos**(String projectName)

**ACHTUNG:** Die Methode darf nicht zwischen einem Aufruf von searchDoIPVCIs und setDoIPVehicleProject verwendet werden. Ansonsten wird die Liste der ermittelten VCIs aus searchDoIPVCIs ungültig und kann nicht für setDoIPVehicleProject verwendet werden.

* Die Methode setVehicleInfo setzt die zu nutzende Vehicle-Info für den nächsten Diagnoseeinstieg. Für die dann folgenden Einstiege wird der Wert verworfen und muss bei Bedarf erneut gesetzt werden. Das Verwerfen erfolgt auch bei einem Fehler im Einstieg. Unter Einstieg wird in diesem Zusammenhang der Aufruf einer der Methoden setVehicleProject, setVehicleProjectAutomatic, setCanFDVehicleProject, setCanFDVehicleProjectAutomatic, setDoIPVehicleProject oder setDoIPVehicleProjectAutomatic verstanden.

Mittels Leerstring für den Parameter vehicleName erfolgt ebenfalls das Rücksetzen / Verwerfen.

Die Methode kann in jedem Zustand außer **Blocked** aufgerufen werden.

**setVehicleInfo**(String vehicleName)

Eine zusätzliche Erweiterung der API-Version 25.1 umfasst eine neue Authentifizierungsmethode für den Aufruf der SFD-Backend-Services. Dabei wird vor dem eigentlichen SFD-Aufruf ein TBT-Token im ODIS gesetzt, welcher dann beim folgenden SFD-Aufruf von ODIS zur Authentifizierung verwendet wird. Die Besonderheit ist hierbei, dass nicht direkt der SFD-Backend auf dem GSB, sondern ein spezieller EXAM-SFD-Service aufgerufen wird, der dann den Request an den SFD-Backend weiterleitet. Der TBT-Token ist jeweils für einen SFD-Aufruf gültg und wird somit nach jeder Verwendung gelöscht.

* Die Methode securityAccessSFDSetTBTToken setzt einen neuen TBT-Token, der für den nächsten SFD-Aufruf mit TBT-Token-Authentifizierung verwendet wird. Voraussetzung für den Aufruf ist ein geladenes Fahrzeugprojekt. Nach einmaliger Verwendung oder nach dem Laden eines neuen Fahrzeugprojektes wird der Token gelöscht.

**securityAccessSFDSetTBTToken**(String tbtToken)

* Bei SFD2 (Absicherung) erfolgt das Setzen der Authentifizierungsmethode über die Methode configureSetting mit dem Schlüssel „SFDE2E.AuthMethod“. Zusätzlich zu den bisherigen Werten „PKI“ und „SoftPSE“ kann nun auch „TBT\_Token“ zum Aktivieren der TBT-Token-Authentifizierung übergeben werden.

**configureSetting**(String key, String value)

* Bei SFD1 (Freischaltung) erfolgt das Setzen der Authentifizierungsmethode direkt beim Aufruf der Zugriffberechtigungsmethoden securityAccessSFDOnline, securityAccessSFDReset und securityAccessSFDAllOnline. Dabei enthält das Argument accessMethod zusätzlich zu den bisherigen Authentifizierungsmethoden „PKI“ und „SoftPSE“ nun auch „TBT\_Token“ zum Verwenden der TBT-Token-Authentifizierung.

z.B.

**securityAccessSFDOnline**(IConnectionHandle connectionHandle,

OnlineRequestedRole role,

OnlineAccessDuration duration,

OnlineAccessMethod accessMethod)

## Version 25.2

Ab API Version 25.2 kann beim Flashen die Ausführung der FDS Prüfung konfiguriert werden. Dazu kann in der bereits bestehenden API-Funktion configureSetting der neue Parameter Flashing.IgnoreFDSCheck mit dem Wert „TRUE“ oder „FALSE“ übergeben werden. Mit „TRUE“ wird die FDS Prüfung ignoriert, mit „FALSE“ wird sie ausgeführt. Der Standardwert ist „FALSE“; d.h die FDS Prüfung wird ausgeführt. Die Konfiguration bleibt für die Dauer der Server-Instanz erhalten. Nach einem Server-Neustart wird wieder der Standardwert eingestellt.

void **configureSetting**(String paramKey, String paramValue)

## Version 26.0

Aus der Webschnittstelle wurde die Möglichkeit entfernt BZD-Protokolle an Carport zu senden. Die dazugehörigen Funktionen zum ein- und ausloggen bei System42 wurden ebenfalls entfernt.

API Dokumentation Automatisierungsschnittstelle

1. Ein Wechsel des Verzeichnisses ist nur über eine Änderung der entsprechenden Einstellung im Bereich „Administration“ möglich. [↑](#footnote-ref-1)