Das CT-Debug & Trace_Modul 4.0

Add-On für die SAP®-ABAP™-Workbench

Traditionelles Debugging und CT-Tracemodus
Ein Performance Vergleich

Die erweiterte Qualitätskontrolle
innerhalb der ABAP Programmentwicklung

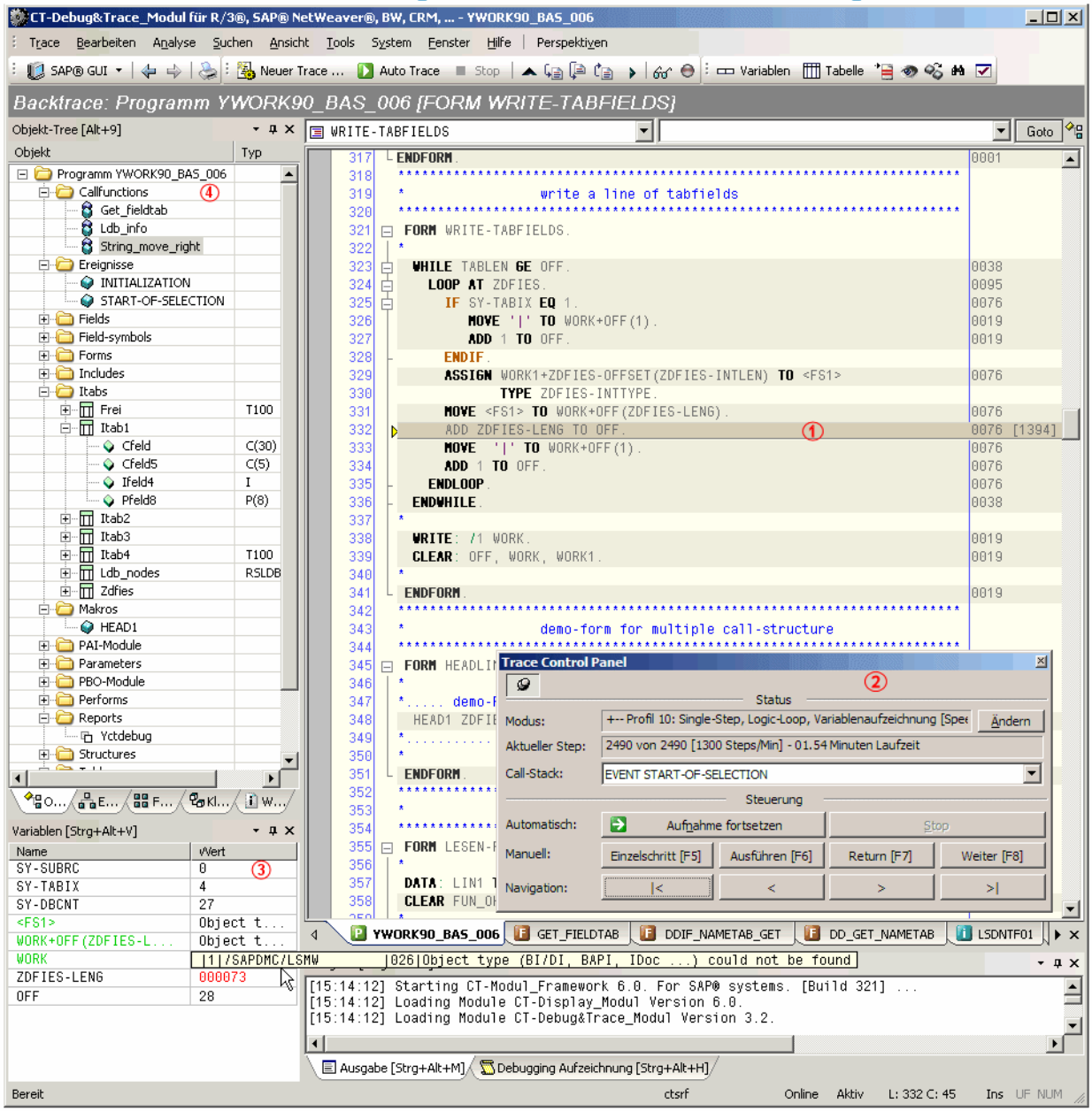
SAP® Certified
Integration with SAP NetWeaver®

Inhaltsverzeichnis

I. Welche Vorteile bietet Ihnen das CT-AddOn_Modul ?

Durch die Integration von traditionellem Debugging **(klassischer ABAP™-Debugger auf dem Applikationsserver)** in Verbindung mit der Leistungsfähigkeit heutiger PC-Prozessoren **(CT-AddOn_Modul auf dem Client-PC)**, können umfangreiche Unterstützungstechniken realisiert werden.

Automatische Steuerung und Trace-Aufzeichnung



SCREEN Informationen

① Jede prozessierte Codingszeile wird im Source Code markiert.

② Das Steuerungspanel zeigt Ihnen LIVE den aktuellen Stand der gerade prozessierten Debug-Session.

③ Die prozessierten Variablen des aktuellen Befehls werden in diesem Bereich mit dem Feldinhalt gezeigt.

④ Der erweiterte Objekttree des prozessierten Programms dient der schnellen Doppelklicknavigation.

Ihr Frontend-PC unterstützt Sie während einer Debugging-Session LIVE durch folgende Aktionen:

- maschinelle **Aufzeichnung** des **Programmablaufes**,
- automatische **Registrierung** der jeweils prozessierten **Variableninhalte und Parameter,**
- schnelle maschinelle AddOn-Steuerung (mehr als **1300** Steps/Min.) der Debugging-Session über **Debug-Profile**,
- über Multistep-Profile wird auch der Ablauf **umfangreicher ABAP™ Programme** (z.B. Transaktionen) transparent,
- **Back-Trace** Funktionen (Stepping im prozessierten Coding vorwärts / rückwärts) während der aktuellen Debugging-Session,
- **interaktive Analysen** des Programmablaufes mit den jeweiligen Variableninhalten,
- Sicherung der **aktuell prozessierten** Coding Member (INCLUDES, Funktionsbausteine, Klassen) mit Debug-Kennzeichnung,
- Auswahl der geeigneten **Ablaufstrategie** während der Debuging-Session,
- umfangreiche **Auswertungsumgebung** (**20** Listen) innerhalb der aufgezeichneten Trace-Daten (Cockpit-Monitoring),
- **Offline-Trace** älterer aufgezeichneter Debugging-Sessions zur Programmanalyse z.B. **vor Modifikationen,**
- **Zwischenstopp** der automatischen Debugging-Session und **manuelles Einzelstepping** mit Trace-Aufzeichnung,
- **Wechsel der Debug-Steuerung** innerhalb der aktuellen Debug-Session vom **CT-AddOn** zum ABAP™-Debugger und zurück,
- umfangreiche **Zusatzinformationen während der Debugarbeiten** (Sourcecode-Komprimierung, Syntaxhighlighting, Multisearch, Trace-Control Panel, Repository Objektbrowser, Zoom innerhalb Source-Karteikarten, ...

II. Im Vergleich: traditionelles Debugging und Trace-Modus

Lassen Sie uns die unterschiedlichen Methoden und Arbeitsabläufe im traditionellen Debugging (klassischer ABAP™-Debugger) und im Trace-Modus (maschinell unterstützte Debugging-Session, CT-AddOn) aufzeigen und bewerten. Um eine differenzierte Bewertung zu erreichen, wurden **4 Schwerpunkte aus typischen Debuggingarbeiten im R/3®** in den Mittelpunkt gestellt:

1. Die Ermittlung der Informationen im Debugging

1.1. Das traditionelle Debugging im klassischen ABAP™-Debugger

Im traditionellen Debugging (ABAP™-Debugger) erfordert das stepweise Prozessing (F5, F6, ...) die **sofortige Auswertung** der aktuellen Situation durch den Benutzer. Der jeweils nachfolgende Debuggingschritt **"überschreibt"** allgemein die Informationen der vorhergehenden Situation. Eine **jederzeit verfügbare Analyse bereits prozessierter Codingteile** (Back-Trace) ist aktuell **nicht** vorhanden**.**

1.2. Der Trace-Modus im CT-AddOn_Modul

Im Trace-Modus werden die einzelnen Debuggingschritte von dem CT-AddOn durch Debug-Profile automatisch prozessiert. Dabei wird eine **relativ hohe Aufzeichnungsgeschwindigkeit** der Verarbeitungsschritte und der Variableninhalte im Online erreicht (z.B. mehr als **1300** Steps/Min. durch Profil-10), manuelle Aktionen des Benutzers sind **nicht** erforderlich. Darüber hinaus erfolgt zeitgleich der Download des jeweiligen prozessierten Codingmembers.

Das **automatische Aufzeichnungsverfahren** können Sie mit einem Mausklick **jederzeit anhalten** und in den **Back-Trace Modus** umschalten.

Der **schrittweise** (je Step) aufgezeichnete Informationsablauf von bereits prozessierten Codingteilen kann auch **während der aktuellen Debugging-Session im Back-Trace Modus** vorwärts / rückwärts **einschließlich** der jeweils prozessierten **Variableninhalte** beliebig oft wiederholt werden.

Durch Debug-Profile wird das Volumen der Traceaufzeichnung gesteuert, dadurch wird auch ein **umfangreicher Programmablauf transparent** (z.B. Transaktionen).

2. Analyse und Strategie im Debugging

2.1. Das traditionelle Debugging im klassischen ABAP™-Debugger

Im traditionellen Debugging zeigt der ABAP™-Debugger in den manuellen Einzelsteps dem Entwickler **LIVE** den Prozessablauf des jeweiligen ABAP™ Programms. Wenn z.B. während der Debugging-Session die Verarbeitungslogik einer bereits prozessierten Codingstelle dem Benutzer nicht mehr präsent ist, bleibt nur die **Wiederholung der Session** (einschließlich der sich daraus ergebenden Vorarbeiten, aktueller Stand von Tabellen).

Auch das Entstehen bestimmter Variableninhalte kann nicht immer sofort nachvollzogen werden. Häufig ergeben sich auch wichtige Fragen erst **nachträglich** in Verbindung mit **anderen prozessierten Codingteilen.** Der Inhalt der **gerade prozessierten Variablen** ist im traditionellen Debugging nur kurzzeitig visuell verfügbar und kann in einem der nachfolgenden Steps **bereits überschrieben** sein.

Die während der Debugging-Session erforderliche Analyse/Strategie über den weiteren Ablauf der Session (ausgehend von z.B. der Veränderung von Variableninhalten) ist aktuell nicht erkennbar (z.B. **wann** wurde **wie** von **welchem Befehl** der **Inhalt der Variablen** x, y oder z geändert).

2.2. Der Trace-Modus im CT-AddOn_Modul

Im Trace-Modus kann die **automatische Aufzeichnung** des Programmablaufes und der jeweils prozessierten Variableninhalte sofort erfolgen, dies gilt sowohl im **manuellen** Modus als auch innerhalb einer **maschinell** unterstützten Debugging-Session. Sie können Ihre **Debugging-Strategie** hiernach ausrichten.

- Noch **während der aktuellen Debugging-Session** können Sie in den **Back-Trace Modus** schalten. Hier können Sie **stepweise** (vorwärts / rückwärts) den Programmablauf der bereits prozessierten **Codingstellen** mit den dazugehörenden **Variableninhalten** nachvollziehen.
- Auch eine **bereits abgeschlossene**, aufgezeichnete Debugging-Session können Sie (oder ein anderer Mitarbeiter) zu einem **späteren Zeitpunkt im Offline-Debugging** auswerten.
- Im Rahmen der Trace-Analysen stehen z.B. **Standardauswertungen** über den zeitlichen Verlauf von Variableninhalten zur Verfügung. Sie können also, je nach dem aktuellen Stand Ihrer Debugging-Session, anhand dieser Informationen die weitere Ablaufstrategie bestimmen.

Weil bereits zu **Beginn** Ihrer Analysearbeiten die jeweils für die Dauer der gesamten Debugging-Session **relevanten Variableninhalte** ersichtlich sind, können Sie **gezielt die Fehlerursachen untersuchen** und Ihre weitere Debuggig-Strategie hiervon abhängig machen.

Wenn eine aktuelle Fehlersituation **kurzfristig** geklärt werden muss, können Sie möglicherweise auf eine **frühere Trace-Aufzeichnung** eines Testfalles zurückgreifen oder eine **schnelle automatische Aufzeichnung** veranlassen.

3. Auswertungen der Debug-Informationen

3.1. Das traditionelle Debugging im klassischen ABAP™-Debugger

Im traditionellen Debugging stehen im ABAP™-Debugger Auswertungen aus einem innerhalb einer Debugging-Session aufgezeichneten Trace-Datenpool im Sinne von Listen, Übersichten, Zusammenfassung aktuell **nicht** zur Verfügung.

Die im klassischen ABAP™-Debugger angebotenen Informationen sind zwar umfangreich, beziehen sich aber auf den jeweils gerade prozessierten Stand des Programmablaufes. Sie gelten für die Dauer der aktuellen Debugging-Session, eine explizite Sicherung von Trace-Daten ist derzeit nicht vorhanden.

3.2. Der Trace-Modus im CT-AddOn_Modul

Im Trace-Modus wird durch das CT-AddOn die Analyse einer Debugging-Session durch eine Reihe von Auswertungen unterstützt.
Das bei der automatischen Steuerung verwendete **Debug-Profil** bestimmt den **Umfang** und die **Details** der aufgezeichneten Daten. Sie können **während der aktuellen Debugging-Session** jederzeit Zwischenauswertungen anfordern.

Die aus dem Trace-Pool resultierenden Auswertungen (mehr als **20 Ergebnislisten**) können Sie einzeln oder gleichzeitig in Kombination mit anderen Auswertungen zur Klärung von Fehlern oder unplausiblen Ergebnissituationen einsetzen.

Durch eine **Synchronisierung** der jeweils aufgerufenen Auswertungen werden z.B. durch Doppelklick auf einen bestimmten Tracestep, die relevanten **restlichen Auswertungen** ebenfalls **auf diesen Tracestep positioniert**. Diese Übersicht (**Cockpit Monitor**) zeigt Ihnen z.B. schnell innerhalb einer Trace-Session aus **unterschiedlichen Perspektiven** die Entwicklung eines Variableninhalts.

Im Vergleich zu den traditionellen Methoden des Debuggings ist die **Trennung** von **automatischer Aufzeichnung** der Debug-Informationen und **integrierter Auswertung der Trace-Daten** eine sehr effiziente Bearbeitungspraxis innerhalb der Programmentwicklung.

4. Dokumentation der Debug-Ergebnisse

4.1. Das traditionelle Debugging im klassischen ABAP™-Debugger

Im traditionellen Debugging steht eine Sicherung der während einer traditionellen Debugging-Session produzierten Daten als Funktion des ABAP™-Debuggers aktuell nicht zur Verfügung. Die **Dokumentation einer aktuellen** oder **bereits zurückliegenden traditionellen Debugging-Session** wird derzeit nicht angeboten.

4.2. Der Trace-Modus im CT-AddOn_Modul

Im Trace-Modus können durch das CT-AddOn sowohl im **manuellen** oder **maschinell unterstützten** Debugging jederzeit **Sicherungen** (auch Zwischensicherungen) des bereits aufgezeichneten Trace-Pool durchgeführt werden. Diese Daten (z.B. Programmablauf, Variableninhalte, Informationen über den Steuerungsablauf, Synchronisierungspunkte, Historydaten usw.) sind allgemein **zeitpunktbezogen** und können unter diesem Aspekt auch ausgewertet werden.

Die Dokumentation enthält zusätzlich auch die **jeweils prozessierten Codingmember**. Im Unterschied zu der **statischen Auflösung** eines Programms (z.B. **INCLUDES**), sind hier z.B. auch die **aktuell gültigen prozessierten Funktionsbausteine und Methoden** mit dem entsprechenden Coding (z.B. TOP-Member der Funktionsgruppe und Funktionsbausteincoding) enthalten.

Die **zeitgenaue Dokumentation** des jeweils prozessierten Codings ermöglicht es auch, **weit zurückliegende** im Debugging prozessierte **Programmstände** einer aktuellen Prüfung zu unterziehen **(Revisionsaspekt)**.

Dabei ist **zusätzlich zu berücksichtigen**, dass auch die während der Originalaufzeichnung verwendeten aktuellen Zugriffe auf **Steuerungstabellen, Stamm- und Bewegungsdaten** implizit in der Dokumentation enthalten sind. Eine alternative Rekonstruktion der Tabellenstände auf den **Zeitpunkt des betreffenden Programmeinsatzes** dürfte vergleichsweise aufwendig sein.

Ein weiterer Aspekt der Trace-Dokumentation ist die Arbeitsteilung von **Datenerfassung** und **Datenauswertung** sowie die eindeutige Zuordnung von **Verantwortungsbereichen** (Vorgabe, Realisierung).

Die Trennung von Datenermittlung, Datenauswertung und Dokumentation innerhalb einer Debugging-Session führt somit zu einer aus mehreren Gründen sinnvollen Arbeitsteilung z.B. zwischen dem **internen Entwicklungsteam** und einer **externen Realisierungsgruppe** (Werkverträge, **Offshore Programmierung** usw.).

III. Einzelinformationen zum Performance-Vergleich

Die Ermittlung von Informationen im Debugging (II, 1)...

1.1. Traditionelles Debugging (klassischer ABAP™-Debugger)

In der bisherigen **konventionellen Debuggingarbeit** im R/3® werden je Programm die zu prozessierenden Programmschritte über die Funktionstasten F5, F6, F7, F8 **manuell** vorgenommen. Der Inhalt von **einzeln manuell** zu bestimmenden **Variablen** wird gesondert z.B. über Doppelklick **LIVE** angezeigt, gleichzeitig ist der Inhalt von **maximal** 2 mal 4 Variablen je Step verfügbar.

Zusätzlich stehen über eine relativ große Menüauswahl eine Vielzahl weiterer Informationsmöglichkeiten über den **aktuellen Stand des prozessierten Programms** zur Verfügung.

Eine **kontinuierliche Aufzeichnung** der im Online **prozessierten Codingzeilen**, der ausgewählten **Variablen** mit Inhalt und der umfangreichen **Zusatzinformationen** erfolgt aktuell **nicht**.

Ein Fazit:
Die **schrittweise** (je Step) ermittelten Informationen im traditionellen Debuggen erfordern die **sofortige Auswertung** durch den Benutzer. Der jeweils nachfolgende Debuggingschritt **"überschreibt"** allgemein die Informationen der vorhergehenden Situation. Eine **jederzeit verfügbare Analyse bereits prozessierter Codingteile** (Back-Trace) ist aktuell **nicht** vorhanden**.**

Zur Vollständigkeit ist noch darauf hinzuweisen, dass die im ABAP™-Debugger während der Debugging-Session verfügbaren **Detailinformationen** (im Vergleich zu den Debuggern anderer Programmiersprachen), **relativ umfangreich** vorhanden sind.

1.2 Automatischer Trace (CT-AddOn_Modul)

Die automatische Informationsermittlung für das Debugging erfolgt über eine Dialogsteuerung durch Debug-Profile. Anstelle der jeweiligen **manuellen** Steuerung über Funktionstasten (F5, F6, F7, F8 ...), dem Setzen und Aufheben von temporären BREAK-POINTS und der Eingabe der relevanten Variablennamen zur Ermittlung des Variableninhaltes, werden diese Aufgaben **maschinell** mit hoher Geschwindigkeit von dem CT-AddOn_Modul übernommen.

Eine 'Logic-Loop'-Funktion sorgt zusätzlich dafür, dass je nach verwendetem Debug-Profil der redundante Datenanteil innerhalb der Schleifenbearbeitung reduziert wird.

So werden z.B. innerhalb einer automatischen Debugging-Session von etwa **10 Minuten** mehr als **10.000 Verarbeitungsschritte** aufgezeichnet und (je nach Coding) zusätzlich mehr als **20.000** Variablen mit ihrem Inhalt registriert (autonom mit dem Debugprofil-10, ohne Steuerung oder Einflußnahme durch den User). Darüber hinaus erfolgt zeitgleich der Download des jeweiligen prozessierten Codingmembers.

Je nach Programm oder Transaktion erfolgt automatisch während einer Debugging-Session **zusätzlich** die Übernahme von z.B. mehr als **100** Membern (ca. **30.000** Codingzeilen) aus der Programmbibliothek des Applikationsservers zur **Analyse, Steuerungs- und Dokumentationszwecken.**

Ein Fazit:

Die **schrittweise** (je Step) aufgezeichneten Informationen von bereits prozessierten Codingteilen können auch **während der aktuellen Debugging-Session im Back-Trace Modus** vorwärts / rückwärts **einschließlich** der jeweils prozessierten **Variableninhalte** beliebig oft wiederholt werden.

Die **relativ hohe Aufzeichnungsgeschwindigkeit** der Verarbeitungsschritte und der Variableninhalte im Online wird u.a. durch **Debug-Profile** erreicht, manuelle Aktionen des Benutzers sind **nicht** erforderlich.

Das **automatische Aufzeichnungsverfahren** können Sie jedoch mit einem Mausklick **jederzeit anhalten** und in den **Back-Trace Modus** umschalten (Stepping vorwärts / rückwärts). Alternativ können zuvor gesetzte manuelle **Soft-Breakpoints** die automatische Debugging-Session zu einem Zwischenstop anhalten.

Sie können anschließend **wie gewohnt manuell weiterdebuggen**. Dabei werden zusätzlich die im online prozessierten **Codingzeilen** und die **Variableninhalte aufgezeichnet**.

Die Analysen und Strategien während der Debugging-Session (II, 2) ...

2.1. Traditionelles Debugging (klassischer ABAP™-Debugger)

Im Rahmen einer **manuellen** Einzelstep Debugging-Session zeigt der ABAP™-Debugger dem Entwickler im aktuellen realen Zeitverlauf "... wo es im Programm langgeht...".

In der täglichen Praxis kommt es gar nicht so selten vor, dass der Benutzer den **Programmablauf** und das Entstehen bestimmter **Variableninhalte** für sich **nicht sofort** nachvollziehen kann. Auch ergeben sich wichtige Fragen erst **nachträglich** in Verbindung mit **anderen prozessierten Codingteilen.**

Ein Backstep im Logging und damit eine Wiederholung von kritischen Stellen ist im **traditionellen Debugging** aktuell **nicht** verfügbar.

Der Inhalt der **gerade prozessierten Variablen** ist nur kurzzeitig visuell verfügbar und kann in einem der nachfolgenden Steps **bereits überschrieben** sein.

Zusätzlich sind die Entscheidungen über die weitere **Ablaufstrategie** vom Benutzer **sofort nach jedem Debugging-Schritt** zu treffen, eine umfangreiche Datenbasis über den bisherigen Debuggingablauf steht als Entscheidungshilfe aktuell nicht zu Verfügung.

Ein Fazit:

Wenn also die Verarbeitungslogik einer bereits prozessierten Codingstelle dem Benutzer nicht mehr präsent ist, bleibt nur die **Wiederholung der Session** bis zu dem gewünschten Verarbeitungszeitpunkt (einschließlich der sich daraus ergebenden Vorarbeiten).

Die einer Debug-Strategie vorausgehende Analyse von z.B. Variableninhalten über einen bestimmten Prozessverlauf (z.B. **wann** wurde **wie** von **welchem Befehl** der Inhalt der Variablen x, y oder z geändert) ist aktuell **nicht** verfügbar.

2.2 Automatischer Trace (CT-AddOn_Modul)

Die **automatische Aufzeichnung** des Programmablaufes und der jeweils prozessierten Variableninhalte kann sowohl im **manuellen** Modus als auch innerhalb einer **maschinell** unterstützten Debugging-Session erfolgen.

Diese Trace- und Steuerungsdaten können zu **abweichenden Zeitpunkten,** auf **unterschiedliche Weise**, für **verschiedene Zwecke** ausgewertet werden:

- Noch **während der aktuellen Debugging-Session** können Sie in den **Back-Trace Modus** schalten. Hier können Sie **stepweise** (vorwärts / rückwärts) den Programmablauf der bereits prozessierten **Codingstellen** mit den dazugehörenden **Variableninhalten** nachvollziehen.

- Auch eine **bereits abgeschlossene**, aufgezeichnete Debugging-Session können Sie (oder ein anderer Mitarbeiter) zu einem **späteren Zeitpunkt im Offline-Debugging** auswerten.

- Im Rahmen der Trace-Analysen stehen z.B. Standardauswertungen über den zeitlichen Verlauf von Variableninhalten zur Verfügung. Sie können also, je nach dem aktuellen Stand Ihrer Debugging-Session, anhand dieser Informationen die weitere Ablaufstrategie bestimmen.

Ein Fazit:

Der Zeitaufwand für die Klärung von unplausiblen Programmergebnissen kann erheblich durch eine entsprechende **Differenzierung der Arbeitsabläufe** reduziert werden:

- zuerst wird **maschinell** mit dem passenden Debug-Profil die Fehlersituation oder der jeweilige Programmabschnitt aufgezeichnet,

- danach beginnt die Analyse der aufgezeichneten Daten.

Weil bereits zu **Beginn** Ihrer Analysearbeiten die jeweils relevanten Variableninhalte innerhalb der Trace-Aufzeichnung ersichtlich sind, können Sie gezielt die Fehlerursachen untersuchen.

Die Ermittlung der realen Programmlogik und der Variableninhalte innerhalb einer maschinell unterstützten Debugging-Session kann in einer wesentlich "lockeren" Umgebung erfolgen. Falls Sie im Debuggingablauf eine kritische Stelle erst **nachträglich** entdecken, können Sie sich über diese (erst jetzt als wesentlich angesehenen) aufgezeichneten Trace-Daten gezielt informieren.

Wenn eine aktuelle Fehlersituation schnell zu klären ist, können Sie möglicherweise auf eine frühere Trace-Aufzeichnung eines Testfalles zurückgreifen oder eine schnelle automatische Aufzeichnung veranlassen.

Die Auswertung der Debug-Informationen (II, 3)

3.1. Traditionelles Debugging (klassischer ABAP™-Debugger)

Auswertungen aus einem innerhalb einer Debugging-Session aufgezeichneten Datenpool im Sinne von klassischen Übersichten, Zusammenfassung, Listen oder Tabellen stehen im traditionellen Debugging aktuell **nicht** zur Verfügung.

Die im ABAP™-Debugger angebotenen detailreichen Informationen beziehen sich auf den jeweils gerade prozessierten Stand des Programmablaufes. Sie gelten nur für die Dauer der aktuellen Debugging-Session.

Ein Fazit:
Die vom ABAP™-Debugger für den jeweils aktuellen Programmstand angebotenen **umfangreichen Informationsmöglichkeiten** sind nach dem Ende der Debugging-Session **nicht** mehr verfügbar. Klassische Auswertungen über die Debugging-Daten werden aktuell **nicht** unterstützt.

3.2 Automatischer Trace (CT-AddOn_Modul)

Die Analyse einer Debugging-Session wird durch eine Reihe von Auswertungen unterstützt. Das bei der automatischen Steuerung verwendete **Debug-Profil** bestimmt den **Umfang** und die **Details** der aufgezeichneten Daten. Sie können **während der aktuellen Debugging-Session** jederzeit Zwischenauswertungen anfordern.

Die aus dem Trace-Pool resultierenden Auswertungen (mehr als **20 Ergebnislisten**) können Sie einzeln oder gleichzeitig in Kombination mit anderen Auswertungen zur Klärung von Fehlern oder unplausiblen Ergebnissituationen einsetzen.

Durch eine **Synchronisierung** der jeweils aufgerufenen Auswertungen werden z.B. durch Doppelklick auf einen bestimmten Tracestep, die relevanten restlichen Auswertungen ebenfalls auf diesen Tracestep positioniert. Diese Übersicht (**Cockpit Monitor**) zeigt Ihnen z.B. schnell innerhalb einer Trace-Session aus unterschiedlichen Perspektiven die Entwicklung eines Variableninhalts.

Ein Fazit:
Aus dem Trace-Pool einer manuell oder maschinell gesteuerten Debugging-Session können Sie mehr als **20** Ergebnislisten für die **Programmanalyse**, **Klärung von Fehlern** oder **unplausiblen Programmergebnissen** einsetzen.

Im Vergleich zu den traditionellen Methoden des Debuggings ist die Trennung von **automatischer Aufzeichnung** der Debug-Informationen und **integrierter Auswertung der Trace-Daten** eine sehr effiziente Bearbeitungspraxis innerhalb der Programmentwicklung.

Hervorzuheben ist, dass bereits **vor dem Beginn der Analysearbeiten** die während der Debugging-Session im Online **aufgezeichneten Variableninhalte** zur Verfügung stehen und hieraus die jeweilige **Auswertungsstrategie** abgeleitet werden kann. In der Praxis kann von einer erheblichen Rationalisierung z.B. bei der Fehlersuche, Programmoptimierung oder Qualitätskontrolle ausgegangen werden.

Die Dokumentation der Debug-Ergebnisse (II, 4)

4.1. Traditionelles Debugging (klassischer ABAP™-Debugger)

Eine Sicherung der während einer traditionellen Debugging-Session produzierten Daten wird als Funktion des ABAP™-Debuggers aktuell nicht angeboten.

Ein Fazit:
Die Dokumentation einer aktuellen oder bereits zurückliegenden traditionellen Debugging-Session ist derzeit nicht verfügbar.

4.2 Automatischer Trace (CT-AddOn_Modul)

Im Rahmen einer **manuellen** oder **maschinell unterstützten** Debugging-Session können durch das CT-Debug & Trace_Modul jederzeit **Sicherungen** (auch Zwischensicherungen) des bereits aufgezeichneten Trace-Pool durchgeführt werden.

Diese Daten (z.B. Programmablauf, Variableninhalte, Informationen über den Steuerungsablauf, Synchronisierungspunkte, Historydaten usw.) sind allgemein **zeitpunktbezogen**. Auch eine Zwischensicherung wird grundsätzlich den **jeweiligen aktuellen Stand** zum Zeitpunkt der Sicherung wiedergeben.

Die Dokumentation enthält zusätzlich auch die **jeweils prozessierten Codingmember**. Im Unterschied zu der **statischen Auflösung** eines Programms (z.B. **INCLUDES**), sind hier z.B. auch die **prozessierten Funktionsbausteine und Methoden** mit dem entsprechenden Coding (z.B. TOP-Member der Funktionsgruppe und Funktionsbausteincoding) enthalten.

Die **zeitgenaue Dokumentation** des jeweils prozessierten Codings ermöglicht es auch, **weit zurückliegende** im Debugging prozessierte **Programmstände** einer aktuellen Prüfung zu unterziehen **(Revisionsaspekt)**.

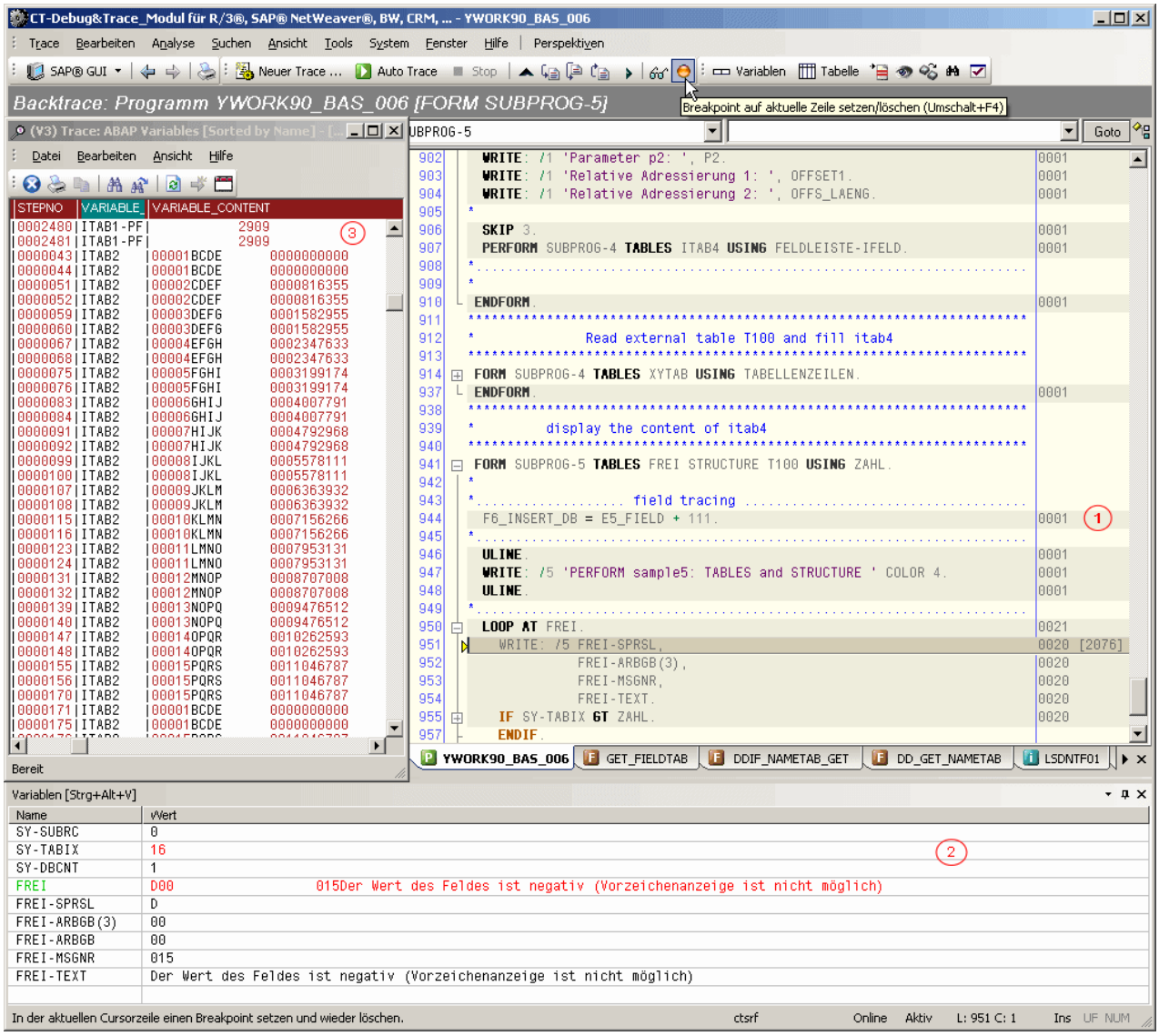
Ein Fazit:
Die Revisionsfähigkeit von Programmen, Abläufen und Testfällen wird durch die Dokumentation von Trace-Sessions wesentlich erhöht. Ein seit längerer Zeit zurückliegender Programmstand kann je nach verwendetem Debug-Profil über das **Offline-Debugging** in seiner ursprünglichen Logik nachvollzogen werden.

Dabei ist **zusätzlich zu berücksichtigen**, dass auch die während der Originalaufzeichnung verwendeten aktuellen Zugriffe auf **Steuerungstabellen, Stamm- und Bewegungsdaten** implizit in der Dokumentation enthalten sind. Eine alternative Rekonstruktion der Tabellenstände auf den **Zeitpunkt des betreffenden Programmeinsatzes** dürfte vergleichsweise aufwendig sein.

Ein weiterer Aspekt der Trace-Dokumentation ist die Arbeitsteilung von **Datenerfassung** und **Datenauswertung** sowie die eindeutige Zuordnung von **Verantwortungsbereichen** (Vorgabe, Realisierung).

Die Trennung von Datenermittlung, Datenauswertung und Dokumentation innerhalb einer Debugging-Session führt somit zu einer aus mehreren Gründen sinnvollen Arbeitsteilung z.B. zwischen dem **internen Entwicklungsteam** und einer **externen Realisierungsgruppe** (Werkverträge, **Offshore Programmierung** usw.).

Ein Screenshot zeigt Ihnen ein Auswertungslayout des CT-AddOn_Moduls:



① Die prozessierten Zeilen im Source Code sind markiert und mit einer Step-Nummer gekennzeichnet.

② In diesem Bereich werden für den aktuellen Debug-Step die jeweiligen Variablen angezeigt.

③ In der Auswertung V3 sind die dokumentierten Variableninhalte einer Debugging-Session übersichtlich nach Variablenname und Step-Nummern geordnet. Sie sehen mit einem Blick z.B. die Änderungen in der Variablen 'ITAB2'.

IV. Die erweiterte Qualitätskontrolle innerhalb der Programmentwicklung...

Neben den bekannten Debuggingarbeiten im ABAP™-Sourcecode, können Sie mit dem CT-Modul zusätzlich **einige generelle Aufgaben innerhalb der Programmentwicklung optimieren.**

Das CT-Modul ermöglicht die **zeitliche Trennung** von **Aufzeichnung** und **Auswertung** der Debugginginformationen.

Dadurch wird sinngemäß ein Offline-Debugging realisiert und Arbeitsabläufe im Entwicklungsprozess können **flexibler** gestaltet werden:

- Die **aktuelle Aufzeichnung** eines Programmablaufes ist also getrennt von der
- **späteren Auswertung und Analyse** der umfangreichen Detailinformationen.

1. Steuerung und Kontrolle im Rahmen externer Programmentwicklung

Die Trennung von **Datenermittlung** und **Datenauswertung** führt zu einer aus mehreren Gründen sinnvollen Arbeitsteilung zwischen dem internen Entwicklungsteam und der externen Realisierungsgruppe (**Werkverträge, Offshore Programmierung** usw.).

Neben der eindeutigen Zuordnung von Verantwortungsbereichen kann bereits in einer frühen Phase der Projektarbeiten eine Kontrolle durch die **maschinell ermittelten Zwischenergebnisse** erfolgen. Im Rahmen der **Endabnahme** und **Funktionskontrolle** der extern erstellten Programme, sind die Aufzeichnungen der Testergebnisse ein zentraler Bestandteil.

Sie können anhand der aufgezeichneten Testergebnisse, aufgrund der **prozessierten Einzelsteps** und der **registrierten Variableninhalte**, im Detail von dem externen Dienstleister

- den Umfang der Testfälle,
- die Qualität der Tests,
- die ordnungsgemäße Durchführung des Testablaufes,
- die Einhaltung der vorgegebenen mathematischen Regeln,
- die Ergebnis der Testfälle

eindeutig nachweisen lassen. Zusätzlich steht Ihnen nach Archivierung dieser Trace-Informationen ein detailreiches Instrument für eventuell später erforderliche Programmänderungen zur Verfügung.

2. Optimierung der Testarbeiten und Programmkontrollen

Umfangreiche Testarbeiten oder die Kontrolle von **Programmänderungen** können **rationeller** und präziser durchgeführt werden, weil Sie sich auf die jeweils relevanten Programmteile **konzentrieren** können. **Die Routinearbeiten übernimmt das CT-Debug & Trace_Modul.**

Je nach Problemstellung können Sie **wechselweise** einen Teil eines umfangreichen Programmablaufes wie bisher in **Echtzeit 'invisible'** prozessieren (F8) und (nach einem Zwischenstop) den für Sie relevanten Teil im **CT-Debug & Trace_Modul maschinell aufzeichnen** lassen. Auch können danach problemlos die restlichen Programmabläufe wieder in Echtzeit **oder** im manuellen Einzeldebugging prozessiert werden.

3. Qualitätskontrolle und Transparenz innerhalb der eigenen Entwicklungsabteilungen

Durch die über Debugging-Profile erreichte **Normierung** von **Aufzeichnung** und **Auswertung** einer Debugging-Session, sind die Voraussetzungen für die **Vergleichbarkeit von Testabläufen** geschaffen.

Die zu erfüllenden **Prüfungsarbeiten** für die Endabnahme eines Programms oder Kontrolle nach Programmänderungen können z.B. in einem **Prüfungshandbuch** vorgegeben werden. Hier kann auch der Aufzeichnungsumfang (= Debugging-Profile) von Testabläufen je nach Qualitätsanforderung angegeben werden.

Über die vorgegebene Debugging-Profile (mehr als 10) können Sie die **Dauer der Debugging-Session** und den **Umfang der Aufzeichnung** bestimmen. Je nach Problemstellung erhalten Sie von einem prozessierten Programm z.B. einen **groben Überblick** über die eingesetzten Funktionsbausteine und deren Einzelergebnisse oder **Detailinformationen** über die einzelnen Programmschritte mit Variableninhalten.

4. Vorbereitung und Unterstützung bei Programmänderungen

Als wesentlicher Bestandteil für die **Planung und Durchführung von Programmänderungen** kann eine **archivierte, frühere Aufzeichnung** des Programmablaufes und der Variablen eingesetzt werden.

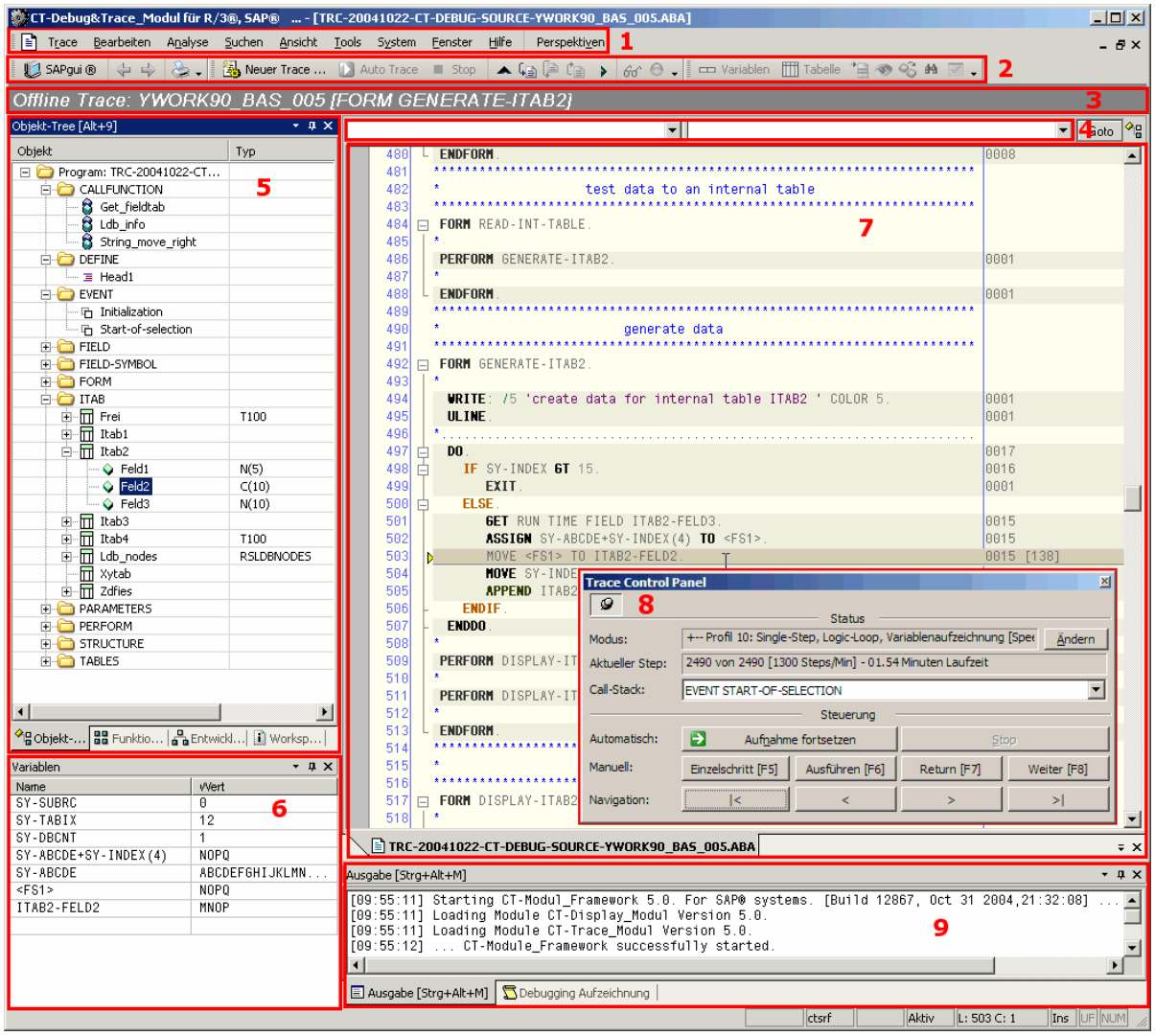
Sie können im Offline-Trace die **bisherige** Arbeitsweise des Programms 'Step by Step' **vor der Änderung** eingehend ermitteln. **Nach** Durchführung der Programmänderung kann in einer **mit gleichem Profil** aufgezeichneten Debugging-Session (und vergleichbaren Testdaten) relativ einfach der geänderte Programmablauf ausgewertet werden.

Zusätzlich besteht die Möglichkeit, die aufgezeichneten Informationen über die Standardauswertungen als ASCII-Tabelle zu konvertieren und mit eigenen ABAP™-Programmen maschinell auszuwerten.

V. Die GUI-Oberfläche 'Online Trace'

Der hier abgebildete Screenshot zeigt eine typische Konfiguration der Oberflächenelemente (GUI, Graphical user interface) des CT-Debug & Trace_Moduls im Modus 'Online Trace'. Sie können die verwendeten Oberflächenelemente (z.B. Menü, ICONs, Nachrichtenzeilen, Funktionswindows, Dialogboxen, PopUp-Windows, Tastaturbelegung usw.) individuell anpassen bzw. positionieren (Look & Feel) und unter einem gesonderten Namen als Perspektive sichern.

Menü: System ->Optionen ->Einstellungen -> Oberfläche anpassen



Die Kurzbezeichnungen der GUI-Elemente

Die nachfolgenden Kurzbezeichnungen der jeweiligen Oberflächenelemente werden allgemein innerhalb der Programmdokumentation verwendet.

1. Die **Menü-Leiste** befindet sich im oberen Teil der GUI-Oberfläche als 2. Element unterhalb der Überschriftszeile.

2. Die **ICON-Leiste** zeigt Menüfunktionen als Piktogramme, eine Unterstützung zur individuellen Anpassung der ICON-Leiste steht zur Verfügung (Menü ->Ansicht ->Symbolleisten).

3. Eine **Überschriftzeile** im Anschluss an die ICON-Leiste zeigt Ihnen den aktuellen Tracemodus, das prozessierte Programm und das betreffende Programmereignis.

4. Über die **Infoline** werden Sie z.B. nach einem Doppelklick auf eine Codingzeile informiert über die Deklaration der verwendeten Variablen und das aktuelle Unterprogramm.

5. Das Window **'Objekt-Tree'** ist in diesem Standardlayout an der linken Seite angeordnet (Funktionsgruppen, Entwicklungsklassen, Workspace sind als weitere Karteikarten verfügbar).

6. Das ebenfalls links positionierte **Variablen Window** sollte genügend Platz für die angezeigten Variablennamen bieten. Je nach prozessiertem Befehl ist eine unterschiedliche Zahl von Variablen beteiligt. Unmittelbar über dem Wort 'Variablen' kann mit dem Cursor und gedrückter Linkstaste dieses Window nach oben erweitert werden.

7. Das **Source Code Window** befindet sich in der Mitte der GUI-Oberfläche, die bereits prozessierten Programme/Includes sind als Karteikarten organisiert.

8. Das frei positionierbare **Trace Control Panel** dient der Steuerung der jeweiligen Debuggingsteps (F5, F6, F7, F8, Shift-F5, Shift-F8 usw.) und befindet sich aktuell im rechten unteren Teil der GUI-Oberfläche. Sie können dieses Trace Control Panel auch fixieren (Pin-Symbol).

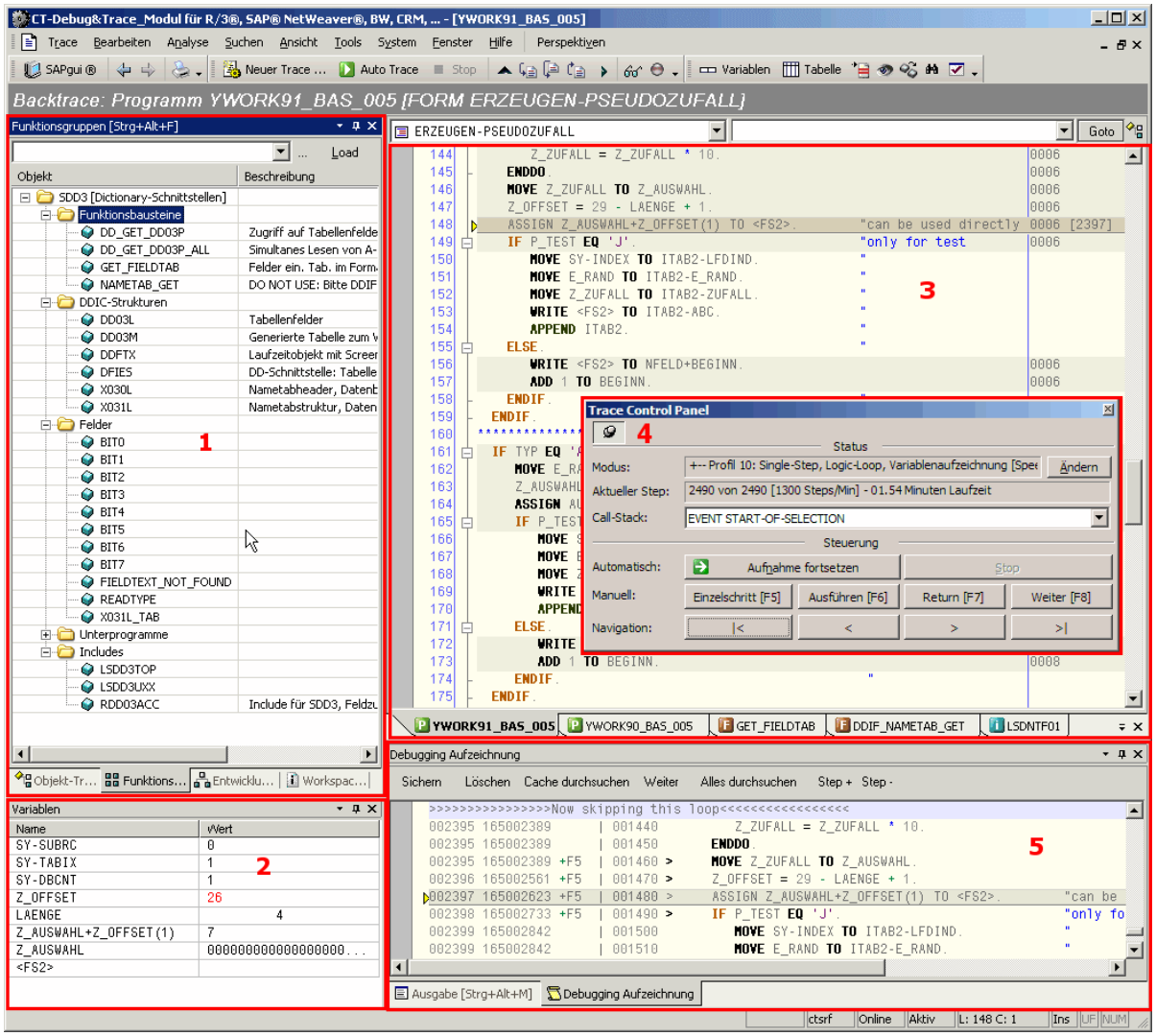
9. Unterhalb des Sourcecode Windows werden in 2 Karteikarten die **Ausgabemeldungen** und die **Debuggingaufzeichnung** (Grundliste) angeboten.

VI. Die GUI-Oberfläche 'Offline Trace'

Der hier abgebildete Screenshot zeigt eine typische Konfiguration der Oberflächenelemente (GUI, Graphical user interface) des CT-Debug & Trace_Moduls im Modus 'Offline Trace'. Sie können die verwendeten Oberflächenelemente (z.B. Menü, ICONs, Nachrichtenzeilen, Funktionswindows, Dialogboxen, PopUp-Windows, Tastaturbelegung usw.) individuell anpassen bzw. positionieren (Look & Feel) und unter einem gesonderten Namen als Perspektive sichern.

Menü: System ->Optionen ->Einstellungen -> Oberfläche anpassen

Einzelne Windows können mit der gedrückten linken Maustaste an einer anderen Stelle des Gesamtwindows 'angedockt' werden.



Die Kurzbezeichnungen der GUI-Elemente

Die nachfolgenden Kurzbezeichnungen der jeweiligen Oberflächenelemente werden allgemein innerhalb der Programmdokumentation verwendet.

1. Das Window **'Funktionsgruppen'** ist in diesem Standardlayout an der linken Seite angeordnet (Objekt-Tree, Entwicklungsklassen, Workspace sind als weitere Karteikarten verfügbar).

2. Das ebenfalls links positionierte **Variablen Window** sollte genügend Platz für die angezeigten Variablennamen bieten. Je nach prozessiertem Befehl ist eine unterschiedliche Zahl von Variablen beteiligt. Unmittelbar über dem Wort 'Variablen' kann mit dem Cursor und gedrückter Linkstaste dieses Window nach oben erweitert werden.

3. Das **Source Code Window** befindet sich in der Mitte der GUI-Oberfläche, die bereits prozessierten Programme/Includes sind als Karteikarten organisiert.

4. Das frei positionierbare **Trace Control Panel** dient der Steuerung der jeweiligen Debuggingsteps (F5, F6, F7, F8, Shift-F5, Shift-F8 usw.) und befindet sich aktuell im rechten unteren Teil der GUI-Oberfläche. Sie können dieses Trace Control Panel auch fixieren (Pin-Symbol).

5. Unterhalb des Sourcecode Windows werden in 2 Karteikarten die **Ausgabemeldungen** und die **Debuggingaufzeichnung** (Grundliste) angeboten.

VII. Fazit

Das effektive Testen und Debuggen von ABAP™ Programmen wird durch den Einsatz des CT-Debug&Trace_Modul signifikant erleichtert.

Sie erhalten eine Testversion des CT-Debug&Trace_Modul unter:
www.ct-software.de/download.htm

VIII. Kontakt

CT-Softwareberatungs GmbH
Ziegeleiweg 8
33415 Verl
Germany

Web: www.ct-software.de
Email: sales@ct-software.de
Tel: +49-(0)5246-9310-15
Fax: +49-(0)5246-9310-16