HITACHI

COBOLオンライン Live セミナー

生成AI活用で変えるCOBOLの未来 － 可能性と展望

2025年12月18日

株式会社 日立製作所
クラウドサービスプラットフォームビジネスユニット
マネージド＆プラットフォームサービス事業部
ミドルウェアサービス本部
藤垣 健太郎
情報処理学会 情報規格調査会 SC 22/COBOL WG 主査

HITACHI

Contents

1. COBOLはなぜ今も重要なのか

HITACHI

社会基盤を支えるCOBOLの重要性

COBOLは社会基盤を支える基幹系システムで多く採用されています。
膨大なCOBOLソースコード資産が稼働中です。

銀行オンラインシステム

座席予約システム（鉄道・航空）

国や地方自治体の税金計算システム

ガス・電力料金計算システム

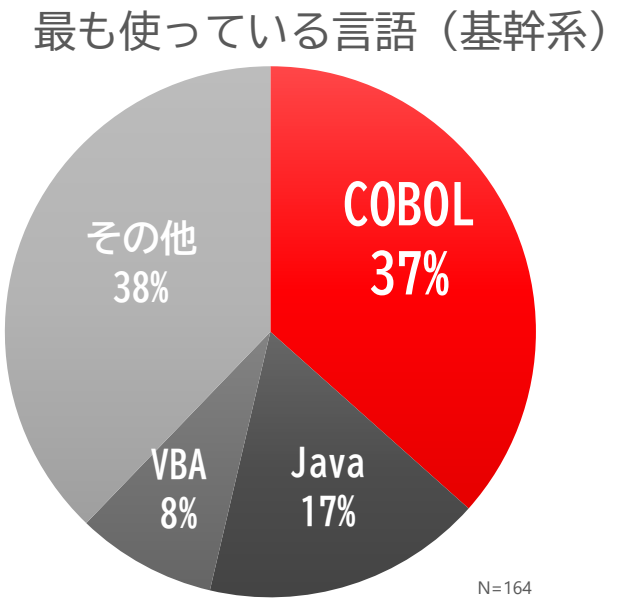
カード会社のクレジットカードシステム

コンビニエンスストアの受発注管理システム

etc.

HITACHI

基幹系で使われ続けるCOBOL



※：本スライドは、一般的なCOBOLの市場動向に関する情報の提供を目的としたものです。特定の商品・サービスを推奨しているものではありません。

出典：日経クロステック2024年11月18日掲載　最も使っている言語は「Python」が2連覇、急上昇したのは「COBOL」 (https://xtech.nikkei.com/atcl/nxt/column/18/03006/111400004/)より当社作成(一部を抜粋)

HITACHI

COBOLが使われ続ける理由

データ構造が静的で理解しやすい

実行時に参照関係を作っていくようなデータ構造を使わない

バッチ処理が得意

大量のデータをレコード単位に効率的に入出力する

COBOLからSQL構文でDBアクセス

埋め込みSQLが広く使われていていて、SQLの知識で
DBを使った処理の記述や理解が可能

国際規格によって言語仕様が安定

互換性を守りつつ、現在も国際規格の改訂が続いている

金額計算に向いている

固定小数点で十進数の桁数を指定して変数宣言することで、
演算のまるめを制御する

必要メモリー量が見積もりやすい

静的なデータ構造を使うので、実行時に必要となるメモリー量を
設計段階で計算できる

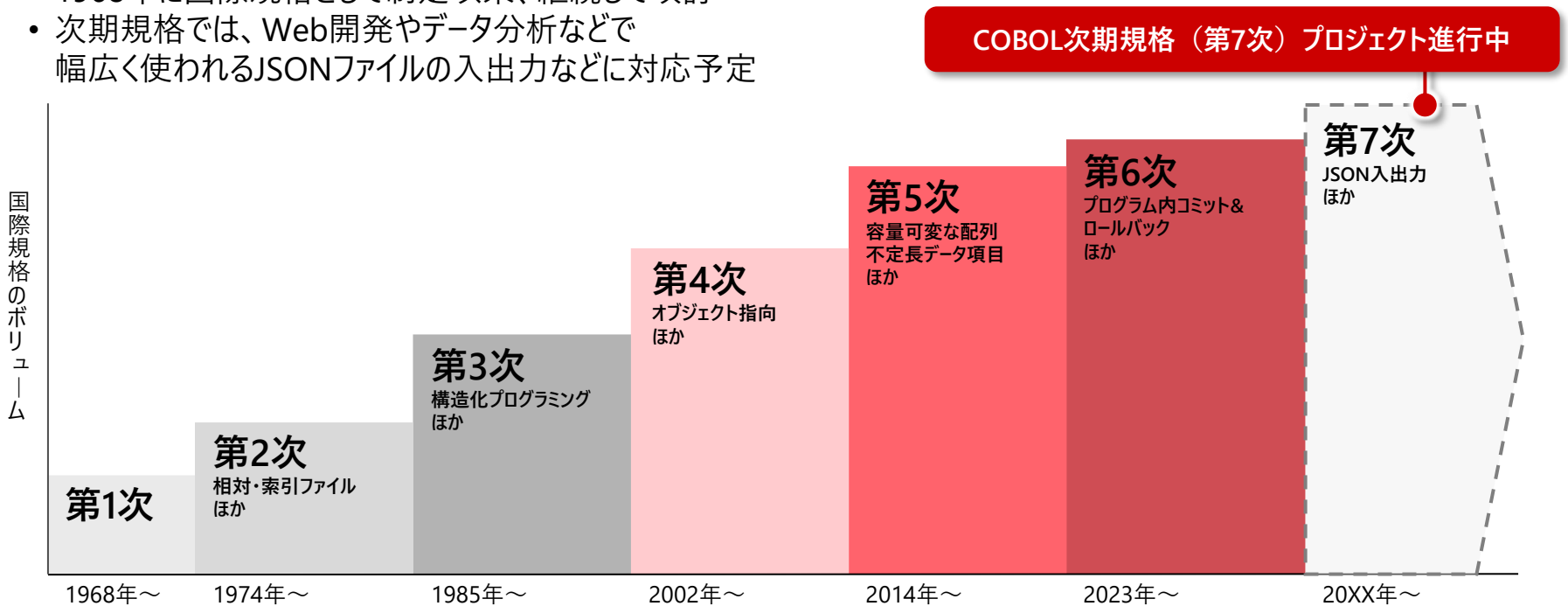
既存のプログラム資産を活用できる

実績のある信頼性の高いプログラム資産を使って
新たな価値を生み出せる

HITACHI

進化し続けるCOBOLの国際規格

- 1968年に国際規格として制定以来、継続して改訂
- 次期規格では、Web開発やデータ分析などで
  幅広く使われるJSONファイルの入出力などに対応予定



2. 現場課題を救うAIエージェントという選択肢

HITACHI

COBOLを取り巻く現場課題

COBOL資産のブラックボックス化

- 長年にわたるメンテナンスでシステムが肥大化・複雑化
- 整理・ドキュメント化が必要

COBOL技術者不足

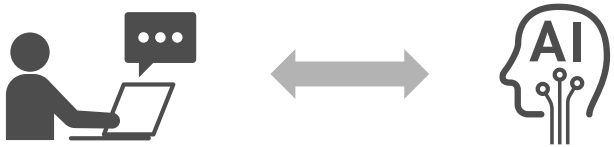
- COBOL技術者の多くが高齢化により開発現場から引退
- COBOL言語の新規学習者は少ない

HITACHI

アプリケーション開発・保守における生成AI活用

さまざまなユースケースで生成AI活用による効率向上が期待されています。
さらに、AIエージェントが利用可能なサービスも登場しています。

プロセス	現状分析・要件定義	設計	開発	テスト	運用・保守
ユースケース	母体調査 影響調査 要件定義書作成	設計書作成	コーディング リファクタリング コードレビュー	テストケース生成 テストコード生成 デバッグ	障害調査



HITACHI

AIエージェントとは

特定の目標を達成するのに必要なタスクを自律的に作成し、計画的に各タスクを実行するAIシステム

人間がゴールとなる目標だけを設定すれば、AIエージェントが以下のような作業を自動で行う

- 必要なタスクの洗い出し
- 情報の収集と分析
- 各タスクの順次実行
- 環境との相互作用
- 必要なプログラムの生成と実行
- 実行結果の評価

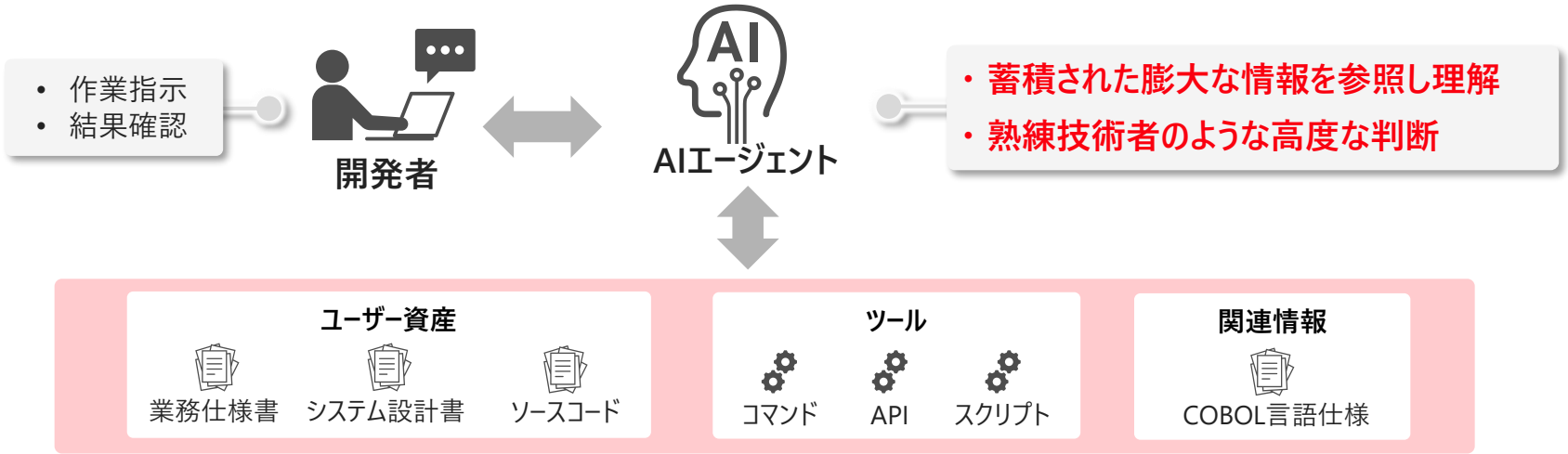
出典：AIエージェント（AI Agent）とは？：AI・機械学習の用語辞典 - ＠IT　https://atmarkit.itmedia.co.jp/ait/articles/2411/06/news021.html　より作成

HITACHI

COBOL開発・保守へのAIエージェント活用の可能性（将来像）

AIエージェントがCOBOL言語やシステム仕様に精通した熟練技術者のように作業遂行

- ✓ ブラックボックス化した資産の調査を効率化
- ✓ COBOL言語や対象システムの経験が浅い開発者でも効率的に作業が可能



3. COBOL改修にAIエージェントを試す
― 試行と考察 ―

※本試行ではVisual Studio Code上でGitHub Copilot（モデル：GPT-5 mini）をAgentモードで使用しています。

HITACHI

COBOLアプリケーションと改修要件

COBOLアプリケーション

週ごとの各店舗の売上データを入力として、月ごとの各店舗の売上金額を集計して出力

＜入力ファイル＞
週ごとの各店舗の売上データ

```
売上年月日, 店舗コード, 売上金額, 店舗コード, 売上金額,・・・
 20210101,     M00001,   886963,     B00001,   433840,・・・
 20210108,     M00001,   318476,     B00001,    63476,・・・
   :
```

＜出力ファイル＞
月ごとの各店舗の売上データ

```
売上年月, 最大売上店舗, 最大売上金額,・・・
 202101,      新潟支店,     3,987,545,・・・
 202102,      高知支店,     3,123,325,・・・
   :
```

改修要件

月ごとに全店舗の売上を集計し、「全店舗売上合計」として出力ファイルに追加

＜期待する出力ファイル＞
月ごとの全店舗売上合計が追加されたファイル

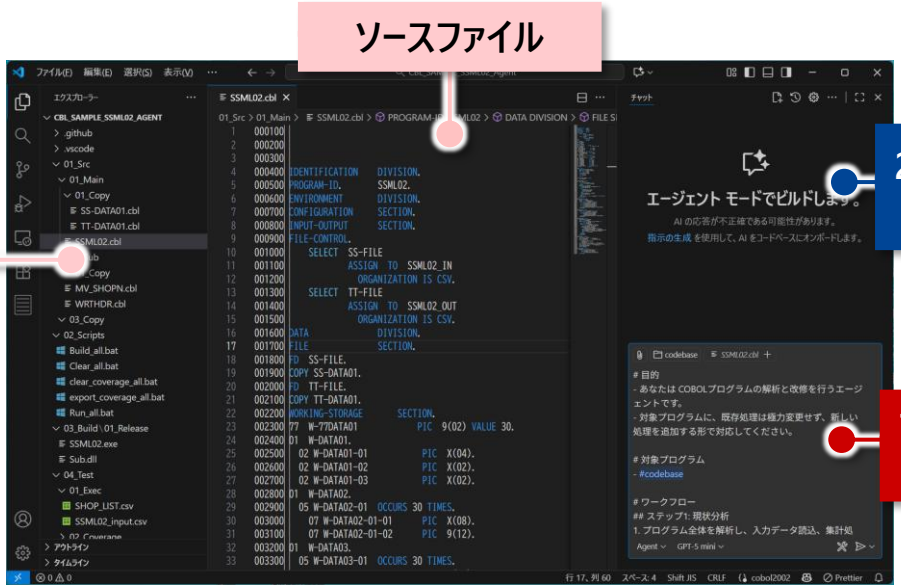
```
売上年月, 全店舗売上合計, 最大売上店舗, 最大売上金額,・・・
 202101,      71,248,972,      新潟支店,     3,987,545,・・・
 202102,      56,887,658,      高知支店,     3,123,325,・・・
   :
```

HITACHI

AIエージェント活用の操作の流れ

AIエージェントが参照するファイル

- ソースコード（登録集原文を含む）
- ビルド用スクリプト
- テスト用スクリプト
- テスト用入力ファイル



2. AIエージェントの応答や作業状況が表示されます。

1. AIエージェントへの指示を入力します。

AIエージェントの処理結果は実行ごとに変動します。
今回は指示を改善しながら複数回試行しました。成功したパターンを紹介します。

HITACHI

AIエージェントへの指示内容 ： タスクを分割して指示

指示内容（全文）

目的

- あなたは COBOLプログラム点検および改修を行うエージェントです。
- 対象プログラムに、既存処理は極力変更せず、新しい処理を追加する形で対応してください。

対象プログラム

- #codebase

ワークフロー

ステップ1: 現状分析

1. プログラム全体を解析し、入力データ読込、業務処理、出力処理の流れを整理してください。
2. COPYやサブプログラムを含めて、今回の修正に関連する箇所をすべて洗い出してください。
3. 結果を@Markdown形式の表で提示してください（列名：ファイル名/段落名/役割/備考）。

完了条件

- プログラム構造をMarkdown形式の表で提示

ステップ2: 修正方針の提示（コードはまだ生成しない）

1. 要件は以下の通り
- 既存の集計処理に加え、月ごとにすべての店舗売上の合計を計算する。
- 出力ファイルに「全店舗売上合計」行を追加してください。
- 改修後出力ファイルの行名：「売上本月 全店舗売上合計 最大売上店舗 最大売上金額・・・」
2. 修正箇所ごとに、以下を提示してください。
- 修正理由
- 影響範囲
- 修正方針（どの節・段落に何を追加するのか）
3. Markdown形式の表で提示してください（列名：ファイル名、段落名、修正理由、修正方針、影響範囲）。
4. この段階ではコードを生成せず、承認を待ってください。

完了条件

- すべての修正箇所を@Markdown形式の表で提示

ステップ3: 修正済みコードの出力（承認後）

1. COBOLソースファイルに修正を入れてください。
2. 新しい集計処理を独立した段落として追加し、必要に応じてPERFORM文で制御フローに組み込んでください。
3. コメントは一行ごとに、修正理由と処理内容を簡潔に記載してください。
4. 生成コードは、1行あたり、固定形式を含めて72文字以内です。行を跨ぐ文字列はコードを出力しないでください。
5. COBOLソースファイルの一行の文字コードはShift-JISです。生成コードはShift-JISで出力し保存してください。
6. 修正したファイルと修正箇所の一覧をMarkdown形式の表で提示してください（列名：ファイル名/修正箇所/修正内容）。

完了条件

- すべてのソースコードの修正が完了、ビルド、テストは実行しない。

ステップ4: ビルド確認

1. ビルド実行し、ビルドエラーがないことを確認してください。
2. ビルドエラーがある場合は修正してください。

完了条件

- ビルドエラーなくビルド完了

ステップ5: サンプルテスト

1. 新しい集計処理に対するテスト用のサンプル入力と、修正されるお出力ファイル内容を提示してください。
2. 修正されるお出力ファイル内容になっているか確認してください。
3. 修正されるお出力ファイル内容にならない場合は、ソース修正案を提示してください。

完了条件

- テスト実行時にエラーがない
- 出力ファイルが想定通りであるか評価する
- 出力ファイル想定通りでない場合は、修正案を提示する

実行してよいコマンド

- ビルド: 02_Script\Build_all.bat
- テスト: 02_Script\Run_all.bat

前提条件

- #codebase 全体を対象に、関連するすべての COBOLソースファイルを修正してください。
- 修正による影響範囲を広く調べるため、既存の変数名やセクション名などは変更しないでください。
- ステップごとにお出力して進んでよいか確認してください。

COBOL改修作業を５タスクに分け、初回にまとめて指示。
AIエージェントが自律的に計画し順次処理します。

ステップ１ ：現状分析

▼

ステップ２ ：修正方針策定

▼

ステップ３ ：ソース修正

▼

ステップ４ ：ビルド確認

▼

ステップ５ ：サンプルテスト

HITACHI

ステップ１ ：現状分析（AIエージェントへの指示）

現状分析 | 修正方針策定 | ソース修正 | ビルド確認 | サンプルテスト

ステップ1: 現状分析

作業要件

1. プログラム全体を解析し、入力データ読込、集計処理、出力処理の流れを整理してください。
2. COPY句やサブプログラムも含めて、今回の修正に関連する箇所をすべて洗い出してください。

現状分析結果の出力形式

3. 結果はMarkdown形式の表で提示してください（列名：ファイル名,段落名,役割,備考）。

完了条件

完了条件

- プログラム構造をMarkdown形式の表で提示

HITACHI

ステップ１ ：現状分析（AIエージェントの応答）

現状分析 | 修正方針策定 | ソース修正 | ビルド確認 | サンプルテスト

修正に関連する箇所の洗い出し：○

COPY句とサブプログラムも含むすべての段落ついて、
ファイル名、段落名、役割、備考が表形式で出力された。
右記の処理フローの説明も合わせて、プログラム構造が文章で表現されていた。

処理の流れの整理：○

入力データ読込、集計処理、出力処理の流れは、
処理の流れがわかりやすい階層構造の箇条書きの形式で出力された。

ステップ1：現状分析（ファイル・段落一覧）

以下は解析結果の表（Markdown）です。列：ファイル名, 段落名, 役割, 備考。

ファイル名	段落名	役割	備考
SSML02.cbl	MAIN SECTION	全体制御：ヘッダ書込み、入力ファイルオープン、メインループ、終了	`SS-FILE`（入力CSV）読み、`TT-FILE`（出力CSV）へ出力
SSML02.cbl	TT-FILE-WRITE-HEADER SECTION	出力ファイルのヘッダ書込みを外部サブルーチンで実行	`CALL 'WRTHDR'` を実行
SSML02.cbl	SS-FILE-INPUT SECTION	入力（SS-FILE）をREADしてグルーピング処理を行うメインループ	同一年月（SS-DATA01-01(1:6)）ごとに集計・出力
SSML02.cbl	DD-DATA-SET SECTION	新グループ開始時の初期化と配列へのデータコピー	`W-DATA02` を初期化し、SS-DATA01 の店舗データを格納

～　以下、省略　～

ステップ1補足：処理フロー（要約）

- 起動 → `TT-FILE-WRITE-HEADER` 呼出（`WRTHDR` がヘッダを書込む）
- `SS-FILE` を READ。レコードの先頭6桁（おそらく「売上年月」でグルーピング
  - 同グループ内：`DD-DATA-SUM` で各店舗の売上を累計（配列 `W-DATA02-01-02`）
  - グループ終了時（年月が変わったら）：
    - `DD-DATA-MXSRT` で降順ソート（最大・最小抽出）
    - `DD-DATA-WRT` で TT レコードを組立て `WRITE`（最大店舗名は `CALL 'MV_SHOPN'` で得る）
- 全て終了 → ファイルクローズ → STOP RUN

凡例）○：指示通り
△：指示通りではないが、許容できる
×：指示通りではなく、要件を満たさない
☆：指示していないが、AIエージェントからの提案

HITACHI

ステップ２ ：修正方針策定（AIエージェントへの指示）

現状分析 | 修正方針策定 | ソース修正 | ビルド確認 | サンプルテスト

作業要件

ステップ2: 修正方針の提示（コードはまだ生成しない）

1. 要件は以下の通り

- 既存の集計処理に加え、月ごとにすべての店舗売上の合計を計算する。
- 出力ファイルに「全店舗売上合計」列を追加して出力する。
- 改修後の出力ファイルの列名：売上年月,全店舗売上合計,最大売上店舗,最大売上金額,・・・

修正方針の出力形式

2. 修正箇所ごとに、以下を提示してください。

- 修正理由
- 影響範囲
- 修正方針（どの節・段落に何を追加するのか）

3. Markdown形式の表で提示してください（列名：ファイル名, 段落名, 修正理由, 修正方針, 影響範囲）。
4. この段階ではコードを生成せず、承認を待ってください。

完了条件

完了条件

- すべての修正箇所をMarkdown形式の表で提示

HITACHI

ステップ２ ：修正方針策定 （AIエージェントの応答）

現状分析 | 修正方針策定 | ソース修正 | ビルド確認 | サンプルテスト

修正方針の提示：○

修正箇所ごとに、修正理由、修正方針、影響範囲が表形式で出力された。

ステップ2：修正方針（コードはまだ生成しません）

目的のおさらい：

- 既存集計に加え「月ごとにすべての店舗売上の合計」を計算し、出力ファイルの列に「全店舗売上合計」を追加する。
- 出力列順（要件）： 売上年月, 全店舗売上合計, 最大売上店舗, 最大売上金額, ･･･（既存の他列は保持）

以下は「修正個所ごとの理由 / 修正方針 / 影響範囲」をファイル･段落単位で整理した表（Markdown）。

注意：この段階ではコードは生成しません。以下の方針を確認いただき、承認を得てから実装（ステップ3）に進めます。

ファイル名	段落名	修正理由	修正方針（どこに何を追加するか）	影響範囲
SSML02.cbl	WORKING-STORAGE / 新規段落（追加）	「全店舗売上合計」を格納･計算するための変数が現状にないため追加が必要	- 新しい WS 変数を追加（例：`W-TOTAL-MONTH` PIC 9(12) COMP/または表示用 PIC）。 - 初期化（グループ開始時にゼロクリア）を `DD-DATA-SET` に追加。	既存変数名を変更しない。WS に変数を追加するため他部は影響小。

～　以下、省略（全８か所）～

複数の実装方法の提案：☆

修正方針として、効率性と保守性の２つの実装方法の提案あり。

実装方針の選択肢（短め）

2つの実装アプローチが考えられます。どちらを採るか承認時に指定ください。

A) 累積時に同時加算（推奨：効率的）

- `DD-DATA-SUM` 内で、`W-TOTAL-MONTH = W-TOTAL-MONTH + SS-DATA01-02-02(...)` のように同時累積。
- `DD-DATA-TOTAL-CALC` は不要または簡易チェック用に残すのみ。

長所：1パスで計算、実装箇所が少ない。
短所：`DD-DATA-SUM` のロジックに手を入れるため変更箇所がやや増える。

B) 出力直前に配列を走査して合算（安全･可読性重視）

- `DD-DATA-WRT` の直前で `PERFORM DD-DATA-TOTAL-CALC` を行い `W-TOTAL-MONTH` を算出（`W-DATA02-01-02(i)` を走査）。
- `DD-DATA-SUM` は配列のみ更新（現状のまま）。

長所：既存の加算ロジックをほぼ変更せず安全に実装できる。
短所：出力時に追加ループが発生（データ量が少なければ問題なし）。

推奨は B（互換性と保守性を優先）。承認時に A を希望されればその方針で実装します。

凡例）○ ：指示通り
△ ：指示通りではないが、許容できる
× ：指示通りではなく、要件を満たさない
☆ ：指示していないが、AIエージェントからの提案

HITACHI

ステップ３ ：ソース修正（AIエージェントへの指示）

現状分析 | 修正方針策定 | ソース修正 | ビルド確認 | サンプルテスト

ステップ3: 修正済みコードの出力（承認後）

作業要件

1. COBOLソースファイルに修正を入れてください。
2. 新しい集計処理を独立した段落として追加し、必要に応じてPERFORM文で制御フローに組み込んでください。
3. コメントは *> 形式で、修正理由と概要を簡潔に記載してください。
4. 生成コードは、1行あたり、空白文字も含めて72文字以内です。行先頭の7文字にはコードを出力しないでください。
5. COBOLソースファイルのコードの文字コードはShift-JISです。生成コードはShift-JISで出力し保存してください。

修正箇所の出力形式

6. 修正したファイルと修正箇所の一覧をMarkdown形式の表で提示してください（列名：ファイル名,修正箇所,修正内容）。

完了条件

- すべてのソースコードの修正が完了。ビルド、テストは実行しない。

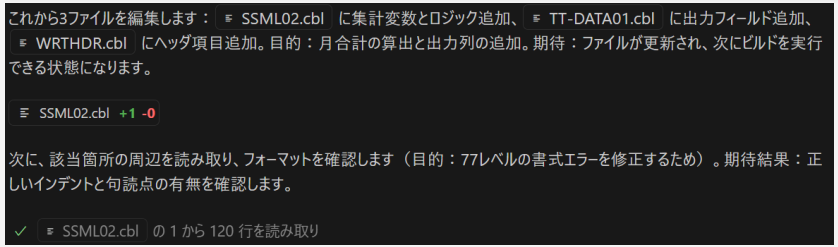
HITACHI

ステップ３ ：ソース修正 （AIエージェントの応答）

現状分析 | 修正方針策定 | **ソース修正** | ビルド確認 | サンプルテスト

ソース修正：○

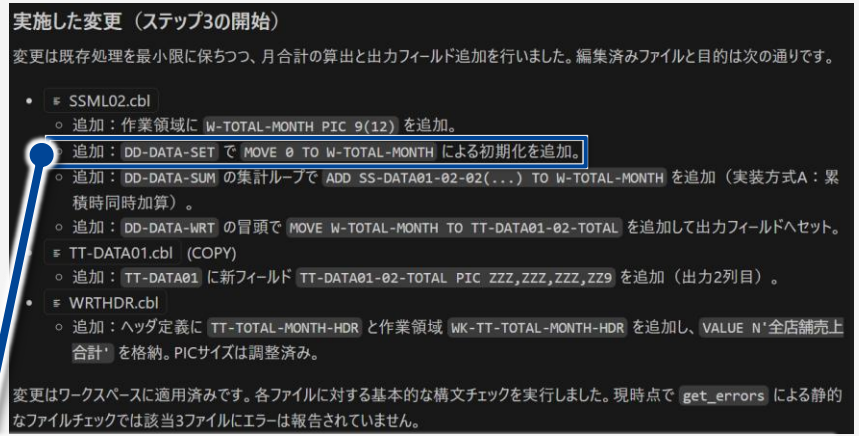
ソース修正と構文エラーのチェックを実施。初回修正で先頭７カラムのコードに対し構文エラーを検出し自動修正。
ただし、以降の修正では同件エラーなし。



～　以下、省略　～

修正箇所一覧の提示：△

表形式での出力を指示したが、箇条書きで出力された。



凡例）○ ：指示通り
△ ：指示通りではないが、許容できる
× ：指示通りではなく、要件を満たさない
☆ ：指示していないが、AIエージェントからの提案

修正内容：×（１箇所だけ）

DD-DATA-SETセクションで初回レコード加算処理が漏れていた。
ただし、ステップ５でAI自身がこの修正漏れを検出し自動修正する。

HITACHI

ステップ４ ：ビルド確認 （AIエージェントへの指示）

現状分析 | 修正方針策定 | ソース修正 | ビルド確認 | サンプルテスト

ステップ4: ビルド確認

作業要件

1. ビルド実行し、ビルドエラーがないことを確認してください。
2. ビルドエラーがある場合は修正してください。

完了条件

完了条件

ビルドエラーなくビルド完了

HITACHI

ステップ４ ：ビルド確認（AIエージェントの応答）

現状分析 → 修正方針策定 → ソース修正 → **ビルド確認** → サンプルテスト

ビルド実行・エラー確認：○

ビルド用スクリプトを実行し、ビルドエラーなしの確認まで実施された。

次のアクション（ステップ4）

ビルドを実行して実際にコンパイル／リンクエラーがないか確認します。ビルドコマンドは指定の Build_all.bat です。実行してもよろしければ実行します（Windows のデフォルトシェルは cmd.exe なのでそのまま実行します）。

実行許可をください。実行後、ビルド出力（成功／エラー）を報告し、もしエラーがあれば修正を行います。

～　途中省略　～

次に、ワークスペースのバッチを絶対パスで直接実行してビルドを試みます（why/what/outcome: バッチがタスク経由で相対パス実行できなかったため、直接パスで実行しビルドログを取得する）。実行します。

✓ c:\demo\CBL_SAMPLE_SSML02_Agent\02_Scripts\Build_all.bat

ソース修正の妥当性確認：☆

指示していないが、テスト用スクリプトを実行し、期待通りの出力になっているかの確認を実施。
出力ファイルに新列『全店舗売上合計』が出力できているところまで確認実施。

これからテスト実行スクリプトを直接実行し、改修後の出力ファイルが期待どおりか確認します（why/what/outcome：Run_all.bat を実行して出力CSVを生成し、内容を取得）。結果ログと生成された出力ファイルを報告します。

✓ c:\demo\CBL_SAMPLE_SSML02_Agent\02_Scripts\Run_all.bat

次に、生成された出力ファイル（`SSML02_out` または `SSML02_OUT` など）を確認します。出力は 01_Exec に出力される想定です。まずディレクトリ一覧を取得して出力ファイル名を確認します。

✓ 01_Exec の読み取り

これから生成された `SSML02_output.csv` を読み、追加した「全店舗売上合計」列が含まれているか確認します（what: ファイル内容を表示、outcome: 出力列を解析して期待と照合）。

凡例）○ ：指示通り
△ ：指示通りではないが、許容できる
× ：指示通りではなく、要件を満たさない
☆ ：指示していないが、AIエージェントからの提案

HITACHI

ステップ５ ：サンプルテスト（AIエージェントへの指示）

現状分析 > 修正方針策定 > ソース修正 > ビルド確認 > サンプルテスト

ステップ5: サンプルテスト

作業要件

1. 新しい集計処理に対するテスト用のサンプル入力と、想定される出力ファイル内容を提示してください。
2. 想定される出力ファイル内容になっているか確認してください。
3. 想定される出力ファイル内容になっていない場合は、ソース修正案を提示してください。

完了条件

完了条件

- テスト実行時にエラーがない
- 出力ファイルが想定通りであるか評価する
- 出力ファイル想定通りでない場合は、修正案を提示する

HITACHI

ステップ５ ：サンプルテスト（AIエージェントの応答）

現状分析 > 修正方針策定 > ソース修正 > ビルド確認 > サンプルテスト

想定される出力ファイル内容か確認：◯

テスト用入力ファイルから、改修後アプリで想定される出力ファイル内容を算出するスクリプトを自動生成し実行。想定する内容と、改修後アプリで出力した内容を比較。
差分を発見し想定通りでないことを確認。

これから Step5（サンプルテスト）を実行します：小さなテスト入力ファイルを作り、既存のバイナリで実行して出力を検証します（目的：新集計の正しさを確認）。まず既存の入力ファイルの場所と名前を確認します。I'll read Run_all.bat and Build scripts to see how files are passed.

～　途中省略　～

これから Python スクリプトを実行して 202101 の合計を求め、出力ファイルの該当行の合計と照合します（what: python 実行、outcome: 合致確認）。実行します。

✓ `python c:\demo\CBL_SAMPLE_SSML02_Agent\04_Test\01_Exec\sum_month.py c:\demo\CBL_SAMPLE_SSML02_Agent\04_Test\01_Exec\SSML02_input.csv 202101`

出力ファイルの202101行に表示されている「全店舗売上合計」は "56,136,719" となっているが、スクリプトで計算した合計は 71,248,972 となった。差分があるため、原因を調査する。可能性：入力の取り扱い（店舗数30件のうち一部のみ集計）、あるいは既存プログラムが何らかの単位（千円）で出力しているなど。次に SSML02.cbl の金額 PIC と出力フォーマットを見る。

ソース修正：◯

差分原因調査→修正方法提示→ソース修正→再検証し、エラーなく想定する内容になっていることを確認した。
（ステップ３での修正漏れへの対応）

これから DD-DATA-SET に初期レコード分の月合計加算を追加します（目的：方式Aで初回レコードが合算されず差が出ていたため）。変更後に再ビルド→再実行して出力を検証します。

SSML02.cbl +1 -0

ビルドスクリプトを実行して修正後のビルドを確認します（what: Build_all.bat を直接実行、outcome: ビルドログ）。実行します。

✓ `c:\demo\CBL_SAMPLE_SSML02_Agent\02_Scripts\Build_all.bat`

ビルド成功のようです。次にテスト実行を再度行い、生成された SSML02_output.csv の先頭行を再確認して合計が期待通りに変わったか確認します。実行します。

✓ `c:\demo\CBL_SAMPLE_SSML02_Agent\02_Scripts\Run_all.bat`

凡例）◯ ：指示通り
△ ：指示通りではないが、許容できる
× ：指示通りではなく、要件を満たさない
☆ ：指示していないが、AIエージェントからの提案

HITACHI

試行結果

指示通りに成功

- ✓ すべてのステップを AIエージェントが自律的に遂行
- ✓ 要件定義レベルの指示から 期待通りの実装を達成
- ✓ AIエージェント自身が 修正漏れを検知し対処まで実施

AIエージェントがCOBOLソースを理解し、改修要件を満たすことができた

失敗して作業中断（紹介していないケース）

× Shift-JISソースが別の文字コードで編集される。

【人手での対策】
指示内容にソースファイルの文字コードを追記。

× AIエージェントが結果検証用に生成したスクリプトが実行エラー。AIエージェント自身がエラー解消を繰り返し試みたが解消できず。

【人手での対策】
指示全文をはじめから再実行。
再生成されたスクリプトはエラーなく検証完了。

HITACHI

考察 ー試行結果から見る、AIエージェントによる課題解決の可能性ー

課題 COBOL資産のブラックボックス化

- ✓ AIエージェントがCOBOLソースを調査・理解した。
- ⇒ 大規模なCOBOLソースの調査効率化にも期待！

課題 COBOL技術者不足

- ✓ 日本語の指示でCOBOLソースの調査や改修を実施できた。
- ✓ AIエージェントがCOBOL言語やプログラム仕様を理解し、要件に応じた改修を実現できた。
- ✓ AIエージェントがCOBOLソースを解説してくれた。
- ⇒ COBOL言語の経験が浅い開発者でも効率的に開発や学習が可能に！

AIエージェント活用の留意点

- 処理結果は実行ごとに変動する。 ⇒ 結果を安定させるために必要十分な指示が必要。
- 結果の誤りや失敗するケースもある。 ⇒ 結果の正しさは人の確認が必要。

AIエージェント活用がCOBOL開発・保守の課題解決の有力な選択肢に！

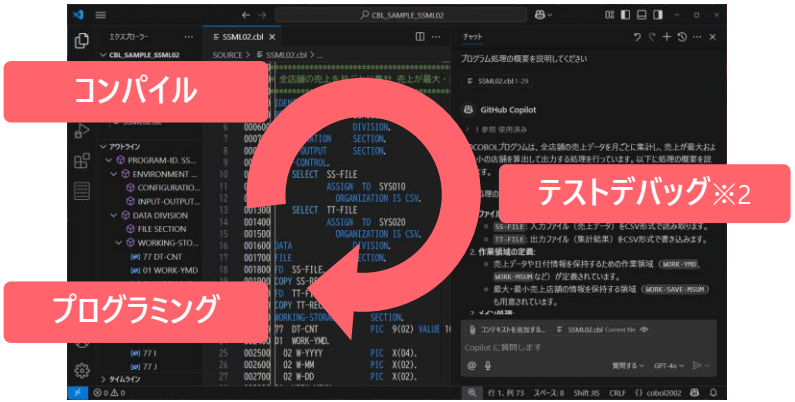
4. 生成AI活用に向けた日立COBOL2002の対応

HITACHI

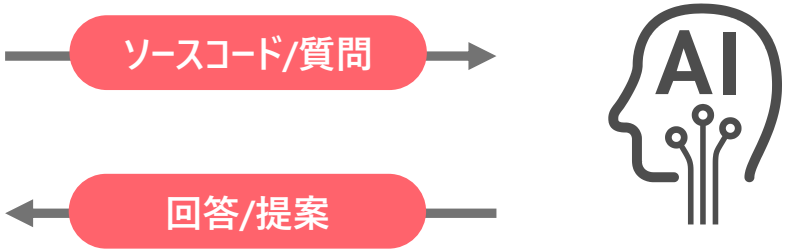
生成AI支援開発を実現するVisual Studio Code(※1)をサポート

✓ 生成AIとシームレスに連携しながら一連の開発作業が可能！

Visual Studio Code



生成AIサービス



※1 Visual Studio Code は Microsoft社が提供するコードエディタです。
Windows版 COBOL2002 Version 5で提供する拡張機能をVisual Studio Codeにインストールすることで利用できます。
※2 テストデバッグは2026年度対応予定。

HITACHI

COBOLアプリ開発保守に役立つ機能をサポート

表示機能

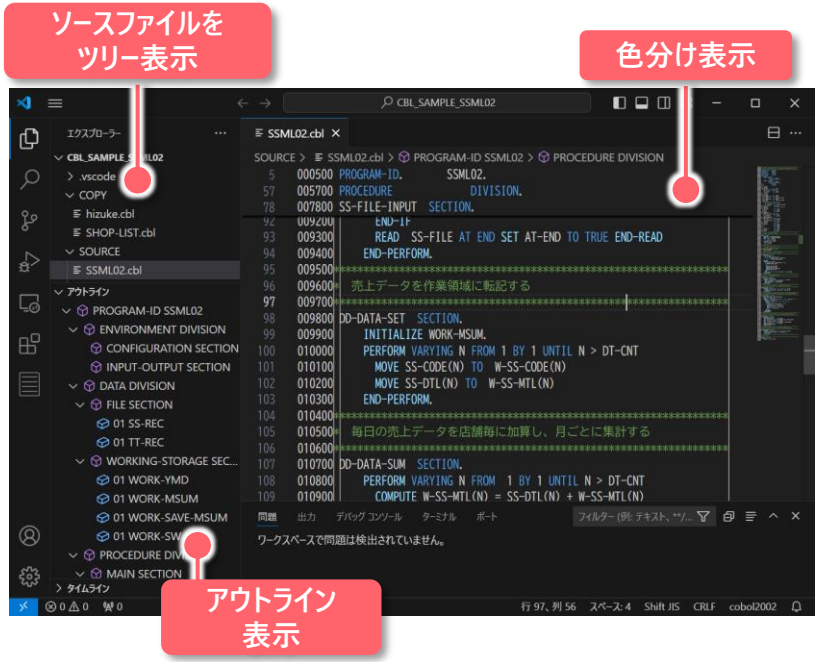
COBOLソースの構文要素ごとに色分け表示やマーカ表示を行い、可読性や入力ミスの発見を支援します。エラー箇所には波線が表示され、視認性が向上します。

コーディング支援機能

構文チェックやキーワード補完、構文テンプレートの挿入など、COBOLプログラムの記述を効率化し、ミスを減らす支援機能を提供します。

コードナビゲーション機能

定義や参照へのジャンプ、アウトライン表示、ホバー情報など、ソース内の構造把握や移動を容易にし、保守性や理解度を高めます。



※ 製品の改良により予告なく記載されている仕様が変更になることがあります。

HITACHI

日立COBOL2002は
生成AIなどの最新技術を活用しながら
お客さまがCOBOL資産を安心して
継続的に活用できる環境を提供していきます。

✓ 2026年2月 COBOL2002 05-10 リリース予定

付録. COBOL2002製品Webサイト ご紹介

「COBOL2002」の情報はこちらをご覧ください。

COBOL2002製品の情報については製品Webサイトなどに掲載していきます。
ぜひご覧ください。

URL: https://www.hitachi.co.jp/soft/cobol/

または 日立COBOL2002

セミナー内容に関するお問い合わせ先

お問い合わせいただく前に、「個人情報保護に関して(*)」をお読みいただき、記載されている内容に関して
ご同意いただく必要があります。ご同意いただけない場合には、お問い合わせに回答できない場合があります。
お問い合わせへの個人情報のご提供は、ご本人さまの任意です。ご提供されない場合は、お問い合わせに
回答できない場合があります。「個人情報保護に関して(*)」をよくお読みいただき、ご同意いただける場合のみ、
下記のアドレスに送付ください。

お問い合わせ先：株式会社 日立製作所 マネージド＆プラットフォームサービス事業部 ミドルウェア本部
ＣＯＢＯＬ担当　Ｅ-ｍａｉｌ：cobol2002@itg.hitachi.co.jp

(*)「個人情報保護に関して」：https://www.hitachi.co.jp/utility/privacy/index.html

他社商品名、商標等の引用に関する表示および免責事項

《他社所有名称に対する表示》

- Microsoft、Windows、VS Code は、マイクロソフトグループの企業の商標です。
- GitHub Copilotは、GitHub Inc.の商標または登録商標です。
- その他記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。

《免責事項》

- 画面表示をはじめ、製品仕様は、改良などのため変更することがあります。
- 本製品を輸出される場合には、外国為替および外国貿易法の規制ならびに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。なお、不明な場合は、当社担当営業にお問い合わせください。


HITACHI