Volume

3

OPCAT SYSTEMS

OPCAT 4.0

RPG Conversion Guide

INSIGH 4.0

OPCAT SYSTEMS

OPCAT 4.0 RPG Conversion Guide

Table of Contents

Chapter 3

Purpose of this Guide

The purpose of this guide is to provide experienced OPCAT 4 users to create and understand models generated using OPCAT's RPG Reverse Engineering module. This guide is also designed for use by system administrators who need to install and run the Reverse Engineering process.

RPG Reverse Engineering

OPCAT 4.0 is designed, among other features, to allow the organization to capture the current status of the code fast and accurately with minimum human intervention. The Convertor is the first building block for enabling this capability. It is designed to specifically capture code written in RPG. After capturing, it displays its logic and interdependencies by translating the RPG code into an OPCAT model, consisting of a set of interconnected OPDs.

The Convertor takes as input well-formed XML files generated from the RPG source code into OPCAT model. The XML files must comply with OPCATPACKAGE.XSD schema. The Convertor then runs in batch mode and converts the XML files into an OPCAT model. The Convertor assumes that each batch is complete, i.e., that all the RPG files which depend on each other (by call, copied code, etc.) are included in this batch. The Convertor does not take into consideration files which had been previously converted. Therefore, no partial run is possible.

Before reading this guide we recommend reading the *OPCAT Server User Guide* with special attention to the *Reuse and Dependency Tracking* Chapter. This guide shall be read while reviewing the OPCAT Model.

OPCAT Navigation Tools

The models representing RPG programs and tables are large. Therefore, before digging into such large-scale models, we first briefly review some of the Navigation Tool of OPCAT, which help overcome the complexities of the models.

- **OPD Hierarchy** – OPCAT models are organized hierarchically in a top-down fashion. Each process may be in-zoomed into a more detailed level. The top-level OPD is called SD for System Diagram. By zooming into processes we create a hierarchy of SDs. Lower-level diagrams are automatically labeled SD1, SD1.1, etc. The SD set is the set of all the SDs in the model hierarchy. In addition, anything—objects or process—may be unfolded into a View, which details the thing's parts and/or characteristics. While SDs in the SD set are arranged hierarchically, Views may or may not be arranged in such a way. SDs and Views are collectively referred to as OPDs and the entire collection of diagrams, including all the SDs and all the Views, is the OPD set. Objects and processes that are refined—in-zoomed or unfolded—are marked in thick line. The OPD Hierarchy pane, found on the left hand side of the OPCAT screen, serves as an important navigation tool. You can jump to any OPD by double clicking on it in the OPD Hierarchy pane.

- **Things List** – All the things in the model are arranged alphabetically in the Things List, which accessible by clicking "Things List" in the OPD Hierarchy pane. Objects appear first and then processes. Double clicking on a thing opens a "Where Used" list at the bottom grid, specifying all the places where this thing appears. Clicking on any line will take you to the OPD where the thing appears. Note that if a thing is both exposed and used in the same model, the exposed thing and used thing will appear as two different things in the Things List.

- **Expose Reports** –to identify dependencies within the model, right click on an Exposed or Used thing and select Expose>Show Local Successors/Show Parent thing/Show Successors as needed. Note that these reports are also available by right-clicking on a thing at the Expose List. For more information about the Expose Reports please refer to the *OPCAT Server User Guide.*

- **Show Parameters** – ALL the parameters that objects and processes may have in the XML are kept in the OPCAT Model. To view these parameters, right-click on a thing in any OPD and select Parameters. The parameters are shown at the OPCAT bottom grid, arranged by Group, Sub Group, Name, and Value.

XML Schema and OPCAT Representation

Each XML schema may include a ProgramModel, which represents an RPG program, or a MessageMap, which represents a table. Table may be logical, physical, or a reference file. The following section describes how each XML element is represented in OPCAT.

1. Tables

Tables are represented by XML files called MessageMaps. An OPCAT model of a MessageMap contains the table object and several child objects, one for each field in the table. To avoid crowded diagrams, during the conversion process we split the fields to separate Views. The number of fields per each View can be set at the convertor configuration file (See below at **שגיאה! מקור ההפניה לא נמצא.** chapter). The XML file includes additional parameters about each field. This information can be viewed by right-clicking the field and selecting "Show Parameters". The field parameters list will appear at the bottom grid.

You will find that the parent object – the table – is Exposed Public (EPB). This means that the table and all its fields can be used by other programs or tables. All the tables created during the conversion process are marked as EPB.

In many cases, a field in a table originates from a reference table. In such cases, this information will appear in the parameters list at the bottom grid. To view the reference table, double click on the link which appears at the parameters list grid.

2. Undefined Tables and Programs

As noted, we assume that the conversion batch includes all the dependent programs and tables. If this is not the case, the Convertor will try to find missing tables and programs and build dummy models for those missing programs or tables. The models are not usable but are there in order to mark that those programs and/or table XML files are missing from this batch. It is recommended that after the conversion process is done, you will visit the Undefined Programs and Undefined Tables directories and analyze why those programs and tables where marked as missing.

Note that undefined models are built for missing tables that are called by IOs, missing Sub-Procedures and programs that are called at Call, and subroutines that are called by an Execute Subroutine. For the latter case, we only build the missing subroutine if there is only one Copied Code model missing in the current model. Otherwise, there is no way to know what is the Copied Code model to which the executed subroutine should be added.

Note that a reference file that does not have an XML file will not be created unless such reference field is a table in its own right, which is used by at least one program.

3. Code Blocks and Parameters

3.1 Code

RPG Code is tagged as Code at the XML files. Each code block has a Parent, which is anther process in this program. Each code block also has an Execution Order. Accordingly, the convertor will attach the code blocks to their respective parents and will place the code at the right place according to the specified execution order. Code block for atomic processes (processes to are not in-zoomed) will appear at the Body of the process. To view this code, double click on the process and go to the Details tab. For any non-atomic process (which is in-zoomed), the code blocks are added as Code process inside the (in-zoomed) process according to their execution order. Such code blocks are interleaved among other processes that may already be inside. The code for each specific code process can be found in the Body.

3.2 Copied Code

Copied code of type Code is a bulk of RPG code, which is copied from another program, in all or in part, into a specific place in the current program. During the conversion, we know that some code was copied from a specific program, but we do not know exactly which part of the other program's code was copied. Therefore, in such a case, you will find the main process of the other program as Used Exposed process, inserted at the right place according to the execution order. Note that the process is not marked in dashed line like a Call Program. This marks the difference between calling an external program, which is environmental to your program, and copying code, which makes the code internal to your program. To see the code for this program open the other model by right-clicking on the process>expose>Open Parent Model.

3.3 Code Aggregation

Each process may be recursively in-zoomed to enclose additional lower-level processes, and each enclosed process may have its own code. In order to provide a complete view, we aggregate the code of all the internal processes into each parent process in the correct execution order. If a parent process has a code block of its own, then this code block will appear first. From that perspective, if we look at the top process, we actually see all the executable code (without the declarations). A Used Process, such as Execute Subroutine (Subroutine Used Private) or Call (Sub-Procedure Used Private) has the call statement in the Body. To view the code of the subroutine, Sub-Procedure, or program that is being called or copied, you will need to go to the source subroutine. To do so, open the other model by right-clicking on the process>expose>Show Parent Thing.

4. Program Model Content

As noted, a Program Model represents an RPG program. A ProgramModel includes the following elements.

4.1 Top Level Objects

At SD, the top-level diagram of the Program Model, we will find the following things, which are created for each program regardless of its content:

4.1.1 Parent Object

- XML Tag: None
- Description: For each program the Convertor adds a parent process carrying the name of this program.
- Expose Status: Expose Public. When other program use things in the Subroutines Set it will Use Public this object (see detailed explanation below at the Subroutines from Copied Code section).

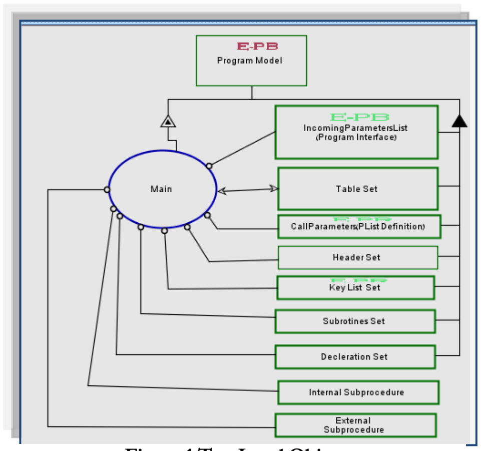

Figure 1:Top Level Objects

4.1.2 Main Process

- **XML Tag:** General (no specific tag for this process)
- **Description:** Each program has a main process. This is the logical starting point of the program. The user may manually insert the program name or any other text in this tag in the RPG program. Unless tagged differently manually, the name of the main process will be "Main".
- **Expose Status:** Expose Public. When other program will call this program, this main process will be inserted (Use Public).

4.1.3 Header Set

- **XML Tag:** Header
- **Description:** – Header Set represents the header of the program.
- **Expose Status:** Not Exposed

4.1.4 Table Set

- **XML Tag:** Table
- **Description:** The Table Set includes all the Tables found in the RPG declarations. At the Unfolding View of the Table Set you will find all the tables which are declared in this program. Right-click on the Table Set and select "Unfold" to view all the tables. Note that the Tables appear as UPB – Use Public. This means that the connection to the Table is maintained. Each

time an IO will appear later on in the program, the Table will be copied to the diagram where the IO appears.

- **Expose Status:** Use Public

4.1.5 Declaration Set

- **XML Tag:** Declaration
- **Description:** The Declaration Set includes the Variable Set, Constant Set, Internal Data Structure (Internal DS) Set, External Data Structure (External DS) Set and Copied Code. Each of these objects is unfolded. At the unfolded view you will find all the fields or parameters related to this object. Note that under Copied Code you will find only Copied Code of type Declaration. Copied Code of type Subroutine will appear under Subroutines Set.
- **Expose Status:** Not Expose

4.1.6 Incoming Parameters

- **XML Tag:** Incoming Parameters
- **Description:** These are Parameters that are input to the RPG program.
- **Expose Status:** The parameters are both Exposed Public and Exposed Private, as they may be used both in the program and by other programs calling this program.

4.1.7 Call Parameters

- **XML Tag:** Call Parameters
- **Description:** Incoming Parameters of the RPG program.
- **Expose Status:** The parameters are Exposed Private as they may be used in the program.

4.1.8 Key List Set

- **XML Tag:** Key List
- **Description:** The unfolded view of the Key List Set includes all the Key Lists used in this program. Note that the Key List may appear in the declaration or at the Copied Code. We can find out about the Key List originating from Copied Code when an IO is using a Key List which cannot be found in the declaration. In such case during the conversion process, the convertor will find this Key List at the models of the Copied Code and will add it to the Key List.

- **Expose Status:** Key Lists are exposed both Public and Private, as they may be used in this program by IO process or by other programs.

4.1.9 Subroutine Set

Subroutines Set is an object added to aggregate the internal subroutines and Subroutines from Copied Code. Both can be seen in the Unfolded View of the Subroutines Set.

4.1.9.1 INTERNAL SUBROUTINE SET

- **XML Tag:** Program Model>Internal Subroutine
- **Description:** An Internal Subroutine is a Subroutine which is part of the current program. The Subroutines are arranged in the Internal Subroutines Set view. A Subroutine may be in-zoomed into additional processes.
- **Expose Status:** Internal Subroutines are Exposed Private. They may be used though by other programs by using Property Advisor (see explanation below for Subroutines from Copied Code). Use of Internal Subroutine in the program is done by Use Private.
- **Code:** The Subroutine code can be found in a separate code process in the in-zoomed view of the subroutine.

4.1.9.2 SUBROUTINES FROM COPIED CODE

- **XML Tag:** ProgramModel>CopiedCode>CopyType <Subroutine>
- **Description:** Includes Subroutines originating from other programs (usually called CopyBooks). During the conversion process, the Convertor identifies the points where Execute Subroutine resides. In such cases it tries to identify the subroutines at the Internal Subroutines Set. If the Subroutine does not exist, then the Convertor tries to locate it at the other ProgramModels from type CopiedCode <Subroutine>. After identifying the subroutine the Convertor Use Public the Parent Object from the other program. It then applies Properties Advisor which brings from the other model the Subroutine Set>Internal Subroutine Set(the remote one) and the Subroutine. It then Private Exposes the Subroutine and adds it by Use Private at the relevant OPD where the Execute Subroutine resides.
- **Expose Status:** Expose Private

4.1.10 Internal Sub-Procedure

- **XML Tag**: ProgramModel>Sub-Procedure
- Description: Subprocedures in this program. Subprocedures are connected to a prototype. – Internal subprocedures prototypes are taken from Declarations with DeclarationType=InternalProcedurePrototype. The linkage between the subprocedure to the prototype is defined through the procedure “ProcedureInterfaceName” attribute. Each

Subprocedure has Return Value and may have Passed Value Subprocedure may have Declaration Set of its own. Internal Subprocedures will appear at the unfolded view of Internal Subprocedure.

- **Expose Status:** Expose Private, Expose Public as they may be used by other programs

4.1.11 External Sub-Procedure

- **XML Tag:** ProgramModel>Sub-Procedure
- **Description:** Subprocedures from other programs which are being called by this program. Subprocedures are connected to a prototype. Each Subprocedure has Return Value and may have Passed Value Subprocedure may have Declaration Set of its own. The Return Value and Passed Value are brought from the Subprocedure's original ProgramModel using Properties Advisor. External Subprocedures will appear at the unfolded view of External Subprocedure.
- **Expose Status:** Use Public, Expose Private

4.2 Process Types

4.2.1 General

- **XML Tag**: General
- **Description**: this tag is used for logical processes entered by users that have no specific RPG meaning. Each General process includes at least one process that may be of any type.
- **Expose Status**: Not Exposed
- **Code Location**: Code blocks whose General process is their parent will appear as Code processes in the in-zoomed OPD of the General process.

4.2.2 IF

- **XML Tag:** IF
- **Description:** IF is a process that in-zooms into more processes of any type, multiple ELSE IF and single ELSE. Each IF statement has an Expression. AnIF process is connected to the Expression's "true" state with a Condition Link. An ELSE process is connected to the Expression's "false" state**.**
- **Expose Status:** Not Exposed
- **Code Location:** Code blocks whose IF process is their parent will appear as Code processes in the in-zoomed view of the IF and will be placed among the other processes according to the Execution Order provided by the XML file.

4.2.3 Loop

- **XML Tag:** Loop
- **Description:** Loop refers to RPG Loop. A Loop may have multiple processes of any type in its in-zoomed view. In case of LoopTypes of DoWhile, an OPM process called "Check Condition" is added at the top of the process (i.e., the condition is checked first). In the cases of DoUntil and For, the "Check Condition" process is added at the bottom of the in-zoomed process (i.e. the condition is checked last). No OPM process is added for LoopType of DoInfinite.
- **Expose Status:** Not Exposed
- **Code Location:** Code blocks with Loop, ELSEIF or Else CHECK CAPUTALIZATION processes as their parent will appear as Code processes in the in-zoomed view of the respective process and will be placed among the other processes according to the Execution Order provided by the XML file.

4.2.4 Select

- **XML Tag:** Select
- **Description:** Select refers to RPG Select. Select may have multiple "When" processes and a single "Other" process in its in-zoomed view. "When" may be in-zoomed into any type of process. Select has an Expression. When is connected to the "true" state of the Expression, while Other is connected to the "false" state of the Expression.
- **Expose Status:** Not Exposed
- **Code Location:** Code blocks whose Select, When or Other processes are their parent will appear as Code processes in the in-zoomed view of their respective process and will be placed among the other processes according to the Execution Order provided by the XML file.

4.2.5 IO

- **XML Tag:** IO
- **Description:** IO is an atomic process (with no in-zoom). It is connected to a table or a field in a table. The table would appear at the Table Set. It is then copied to the OPD where the IO resides.
    - **FieldName -** If the IO includes a FieldName, then the Field object from the Table's original model will be used via the Property Advisor (to learn more about Property Advisor please refer to *OPCAT Server User Guide*). The field will be connected to the IO with an Effect Link. If no field name exists, then the IO will be connected to the Table object with an Effect Link.

  - **Key List and Key Field -** IO may be connected to Key List or KeyField objects. If Key List is used, then it is first added at the Key List Set, Exposed Private and then used at the IO location as Use Private. If KeyField appears, then a corresponding object will be generated in that OPD. Note that if KeyList or KeyField appear with no name, an empty object with no name will be added by the Convertor.

  - **TableName, FormatName, RenamedFormatName -** If the IO refers to a FormatName rather than a table name, the Convertor will try to locate the table according to the RenameFormatName that exists at the Table Set View. If such an element is not found, then the Convertor will try to find a Table with this Format Name in all the tables that appear at the Table Set. If none was found, as in the case of Copied Code that expects to act on a local RenameFormatName, which can be found only in the target program, then the Convertor will add to the original table an object with no reference. In this case, the parameters will indicate that the table is missing.

- **Expose Status:** Not Exposed (used KeyList is Used Private)

- **Code Location**: The IO Code is found in the Body of the process. To see the code, double-click on the process and select the Details tab. No additional in-zoom and Code processes exist for IO.

4.2.6 Execute Subroutine

- **XML Tag:** Execute Subroutine

- **Description:** This tag marks that a Subroutine is executed at a particular place. In this case the Subroutine will be added from the Internal Subroutine Set or External Subroutine Set by Use Private. No specific "Execute Subroutine" process is presented in OPCAT. The External Subroutine Set is built when the Execute Subroutines executes a subroutine. If the subroutine cannot be found at the Internal Subroutine Set, the Convertor scans the Copied Code of type Subroutines and locates the Subroutine. Once found, the Convertor builds the Subroutine at the External Subroutine Set, Exposes it Private, and Uses it Private where the Execute Subroutine appears. If the Convertor cannot locate the Subroutine at any of the Copied Code programs mentioned at the Declarations (usually as a result of a missing XML file), then a process will be added, carrying the name of the Subroutine. Its parameters will state that this is s Dummy subroutine. Note that the in-zoomed view of a Subroutine is found at the Internal or External Subroutine Set views. To locate the Subroutine source code, right-click on it and select Expose>Show Parent Thing.

- **Expose Status:** Use Public, Expose Private, Use Private

- **Code Location:** Code blocks appear as Code processes within the Subroutine's in-zoomed view. Note that this view is found at the Subroutine External/Internal Set where it is exposed. At the ExecuteSubroutine location, the Use Private subroutine will carry the “execution” statement only in the Body of the process.

4.2.7 Tag

- **XML Tag**: Tag
- **Description**: A Tag is a break point which can be returned to during the program either from GOTO or CABXX.
- **Expose Status**: Not Exposed
- **Code Location**: Tag code is found in the Body of the process. Double click on the process and select the Details tab. No additional in-zoom and Code processes exist for Tag.

4.2.8 SQL

- **XML Tag**: SQL
- **Description**: SQL refers to an SQL process within the program.
- **Expose Status**: Not Exposed
- **Code Location**: SQL code is found in the Body of the process. Double click on the process and select the Details tab. No additional in-zoom and Code processes exist for SQL.

4.2.9 Call

- **XML Tag**: Call
- **Description**: Call is used for calling another program or Sub-Procedure. Call does not appear in the diagram as a process in its own right. Rather, the Sub-Procedure or program which is called is Used Public (for programs) or Private (for Sub-Procedures). Programs will also be marked with dashed line to emphasize that this is called program is external. Sub-Procedures are connected to the Return Value and Parameter Set. A Program may use a Plist object, which appears at the Call Parameters. In such a case, the Plist will be Exposed Private at the Call Parameters View and Used Private at the OPD in which the Call appears. In another case, a program may be connected to a Local object <PlistType=Local>. In this case, the Convertor will add this Local object during the conversion process.
- **Expose Status**: Use Private/Use Public
- **Code Location**: Code for Call is found in the Body of the process. Double-click on the process and select the Details tab. Note that the Sub-Procedure code is found at the External or Internal Sub-Procedure Set. Code of a called program is found in the model of the called program. At the location of the Call there is a Used Private Sub-Procedure, or a Used Public Main process of another program, which has only the "Call" statement in the Body of the process. To view the code of the Sub-Procedure, right-click on the process and select Expose>Show Parent Thing. To view the code of the program which is called, right-click on the process and select Expose>Show Parent Model.

4.2.10 CASXX

- **XML Tag**: CASXX
- **Description**: CASXX is a decision process that determines whether to execute a subroutine based on two Factor inputs. The Factor inputs are read from the XML file. The result of the process is a Boolean object. If true, then it is connected with condition link to a subroutine. The subroutine is Used Private from its original location at the Internal or External Subroutine Set.
- **Expose Status**: Not Exposed (Subroutine is Used Private)
- **Code Location**: CASXX code is found in the Body of the process. Double-click on the process and select the Details tab. No additional in-zoom and Code processes exist for CASXX.

4.2.11 CABXX

- **XML Tag**: CABXX
- **Description**: CABXX is a decision process that determines whether to go to a Tag process based on two factors. The Factor inputs are read from the XML file. The result of the process is a Boolean object carrying the name of the Tag.
- **Expose Status**: Not Exposed
- **Code Location**: CABXX code is found in the Body of the process. Double-click on the process and select Details tab. No additional in-zoom and Code processes exist for CABXX.

4.2.12 Commit

- **XML Tag**: Commit
- **Description**: Commit refers to RPG Commit.
- **Expose Status**: Not Exposed
- **Code Location:** Code for Commit is found in the Body of the process. Double-click on the process and select the Details tab. No additional in-zoom and Code processes exist for Commit.

4.2.13 Rollback

- **XML Tag**: RollBack
- **Description**: Rollback refers to RPG RollBack.
- **Expose Status**: Not Exposed

- **Code Location**: Code is found in the Body of the process. Double-click on the process and select the Details tab. No additional in-zoom and Code processes exist for Rollback.

4.2.14 Goto

- **XML Tag:** Goto
- **Description:** Goto Interrupts the order of the program execution and sends the program control to a specific Tag. Goto is a process which yields an object. This object is connected with event link to the Tag specified in the Goto. Note that the Tag process may not be at the same OPD.
- **Expose Status:** Not Exposed
- **Code Location:** Code for Goto is found in the Body of the process. Double-click on the process and select the Details tab. No additional in-zoom and Code processes exist for GOTO.

4.2.15 Return

- **XML Tag**: Return
- **Description**: Return refers to the RPG Return command.
- **Expose Status**: Not Exposed
- **Code Location**: Code is found in the Body of the process. Double-click on the process and select the Details tab. No additional in-zoom and Code processes exist for Return.

4.2.16 Leave

- **XML Tag**: Leave
- **Description**: Leave refers to the RPG Leave command.
- **Expose Status**: Not Exposed
- **Code Location**: Code is found in the Body of the process. Double-click on the process and select the Details tab. No additional in-zoom and Code processes exist for Leave.