# A Prototype Application Protocol for Ready-to-Wear Pattern Making

Y. Tina Lee & Howard T. Moncarz
Factory Automation Systems Division
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

## ABSTRACT

A Ready-to-Wear Pattern Making Information Model is introduced for extending the emerging international Standard for the Exchange of Product Model Data (STEP) to include the exchange of apparel pattern data. This model focuses on a representation of two-dimensional (flat) patterns generated by the traditional ready-to-wear pattern making and grading method. A testing methodology of the information model is also described in this paper.

## KEYWORDS

apparel, application protocol, APDES, CIM, data exchange, grading, pattern, PDES, product data, STEP

## PREFACE

The apparel industry has used computers to great advantage to automate many of its manufacturing processes. However, these advancements often stand alone as "islands of automation." Integrating these separate automated processes could greatly improve the effectiveness of the entire enterprise.

A set of manufacturing data standards that can enable integration of the functions in an apparel enterprise can be based on the Standard for the Exchange of Product Model Data (STEP). STEP is an emerging international standard[1] for representing the physical and functional characteristics of a product throughout the product's life cycle. As a standard, STEP will permit communications among computer environments, each of which performs various product life cycle functions. An advantage of STEP is that it will support the integration of the computer environments using a shared database.

Many of the information requirements as well as the software tools being developed to support STEP are applicable for any manufacturing industry. To serve the needs for a particular industry, Application Protocols (APs) are developed that designate the specific information and application requirements for that industry. The APs draw upon integrated resources[2] to share the same information among different APs.

In recent years, the National Institute of Standards and Technology (NIST) has been working on a project to develop a suite of APs to support computer integration of the apparel product life cycle. This project is sponsored by the Defense Logistics Agency (DLA), and the work is being carried out in cooperation with the American Apparel Manufacturers Association (AAMA). The project has been named the APDES project. APDES stands for Apparel Product Data Exchange Standard.

The APDES project is part of a substantial program sponsored by DLA to improve apparel manufacturing technology. The DLA program is advancing technology from traditional

---

[1] Refer to ISO 10303-1, *Industrial Automation Systems and Integration-Product Data Representation and Exchange-Overview and Fundamental Principles*, to be published.

[2] Integrated resources are "a set of STEP Parts which provide application-independent information models for widely-used types of information. Integrated resources support communication between diverse applications by providing an agreed upon set of definitions and meanings for data that are independent of specific application requirements" [KRA].

size-based methods (ready-to-wear) to methods that use body measurement data directly (made-to-measure). Additionally, the program is advancing production methods from fixed procedures based on standard products to flexible, computer-integrated manufacturing using product representation standards to communicate requirements. The new technologies developed will lead to better fit, higher product quality, economic unit production methods, and quick response. All told, the program is a broad evolution toward integrated enterprises, in which all phases of a product's life cycle are coordinated through a framework of standards, concurrent engineering practice, and supporting technology.[3]

The goal for the APDES project is to develop manufacturing data standards, based on STEP, that will support integration of the projects that DLA is sponsoring. The first objective, when the APDES project began, was to demonstrate the feasibility of using STEP for apparel. The objective was accomplished by developing an information model for pattern data using STEP technology [LEE1]. The information model was represented in the EXPRESS modeling language [ISO11]. The model was implemented in a computer program that exchanges pattern data between two proprietary industry formats [MON]. A neutral set of data structures, based on the information model developed, was used as the intermediary in this process. It was concluded that STEP APs can provide the information interfaces to integrate the apparel product life cycle.

The timeline required for integrating the DLA-sponsored research projects is insufficient to enable the development and integration of formal STEP APs. Instead, a suite of "prototype" APs will be developed.[4] A prototype AP will not require industry consensus, and it will consist of four main components:

Scope: general description of the information requirements and the applications supported

Application Reference Model (ARM): an information model that formally describes the information requirements and constraints for an application domain. The model uses terminology that is familiar to an expert from the application domain. In a prototype AP, the ARM is expressed in the formal computer

[3]Much of DLA's sponsored apparel research is published in the annual Academic Apparel Research Conference proceedings. The most recent conference was held February 17-18, 1992 [DLA].

[4]NIST is concurrently working on the development of formal STEP APs for the apparel industry [LEE2].

language, EXPRESS, and is used in implementing the application interfaces[5]
Conformance Testing (CT) Requirements: testing requirements to demonstrate that an application that implements the prototype AP does so correctly
Useage Guide: a manual that contains a written description of the Scope, the Application Reference Model, and the Conformance Testing Requirements to enable a developer to implement the AP into an application

A formal STEP AP that is developed according to ISO guidelines [PAL] requires a formal documentation and quality review process, as well as an extensive, consensus gathering effort. Additionally, a formal STEP AP requires one more component:

> "Application Interpreted Model (AIM): a model that describes the interpretation of the integrated resource constructs that provide functional equivalence to the AP's information requirements as specified in the application reference model. The form of an AIM is an EXPRESS schema" [PAL].

The intention for the APDES project was to develop CT requirements and procedures for the prototype APs, based on STEP methodology. Unfortunately, the CT methodology is still evolving. There is not yet a clear, documented consensus of what constitutes an abstract test suite[6] and the methodology to create it. There is not yet a documented methodology for

---

[5] In a formal STEP AP, the ARM can be described in one of three information modeling languages (EXPRESS, NIAM [NIJ], or IDEF1X [USA]), and is developed from the point of view of the application domain, without regard to the rest of STEP's resources. It is then mapped to the AIM, which is written in EXPRESS, and uses constructs from the STEP integrated resources. An application that implements the AP will be based directly on the EXPRESS version of the AIM.

[6] The following terminology is used in STEP Conformance Testing (quoted from [ISO31]):

| | |
|---|---|
| abstract test case | One or more files, encapsulating a test purpose and independent of both the implementation and the numerical values, which provide the formal basis from which executable test cases are derived |
| abstract test group | A named set of related abstract test cases |
| abstract test suite | A complete set of abstract test cases, possibly combined into nested abstract test groups, that is necessary to perform conformance testing for a standard or group of standards |
| executable test case | An instantiation of an abstract test case with values NOTE: The form of the realization is still under development [KEM] |
| executable test suite | A complete set of executable test cases that is necessary to perform conformance testing for a standard or group of standards |
| test purpose | A precise description of the objective which an abstract test case is designed |

translating abstract test suites to executable test suites. Consequently, the prototype AP specified in this report describes the procedures for Conformance Testing, but the formal test requirements are not defined.

This report presents the prototype Ready-to-Wear Pattern Making AP. The information model used is a refinement of a previous model developed at NIST [LEE1]. For a proposed STEP AP to be approved by the official STEP sanctioning organization–the International Organization for Standardization–an extensive, consensus gathering effort in the industry must be undertaken. The prototype APs that are currently being developed in the APDES project can be used as initial proposals for developing official STEP standards for apparel. NIST will continue to work with the AAMA to extend the prototype Ready-to-Wear Pattern Making AP to a formal AP. The work will be carried out in cooperation with standards organizations, the apparel manufacturing industry, and academia.

---

to achieve

# TABLE OF CONTENTS

ABSTRACT . . . . . . . . . . . . . . . . . . . . . . . . . . .iii

KEYWORDS . . . . . . . . . . . . . . . . . . . . . . . . . .iii

PREFACE . . . . . . . . . . . . . . . . . . . . . . . . . . .v

1 INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . 1

2 SCOPE . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

3 INFORMATION MODEL . . . . . . . . . . . . . . . . . . . .2

4 SCHEMA . . . . . . . . . . . . . . . . . . . . . . . . . 4

5 SAMPLE EXCHANGE FILE . . . . . . . . . . . . . . . . . . 14

6 CONFORMANCE TESTING . . . . . . . . . . . . . . . . . . .15

APPENDIX A: REFERENCES . . . . . . . . . . . . . . . . . . . 18

APPENDIX B: GLOSSARY OF EXPRESS TERMS . . . . . . . . . . . . 19

APPENDIX C: SAMPLE STEP EXCHANGE FILE . . . . . . . . . . . . 22

# A Prototype Application Protocol for Ready-to-Wear Pattern Making

Y. Tina Lee & Howard T. Moncarz
Factory Automation Systems Division
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

## 1 INTRODUCTION

The Apparel Product Data Exchange Standard (APDES) is a series of specifications, currently under development, for the computer-sensible representation and exchange of apparel product data. The specifications will provide a mechanism capable of describing product data throughout the life cycle of an apparel product, independent from any particular system. The nature of this description makes it suitable not only for file exchange, but also as a basis for implementing and sharing product databases, and archiving.

APDES is based on, and in time will extend, another series of standards called the Standard for the Exchange of Product Model Data, or STEP, currently being developed by the International Organization for Standardization (ISO). In STEP, each International Standard in the series is called a Part and is published separately. Parts are grouped into the following classes: description methods, integrated resources, application protocols, implementation forms and conformance testing. The classes are described in the STEP Overview, ISO 10303-1.

All specifications in the APDES series are application protocols. The specification defined in this report addresses the industrial need to exchange ready-to-wear apparel pattern data between different apparel pattern making software systems, and also between pattern making and marker making systems.

## 2 SCOPE

This prototype AP specifies the information necessary to represent two-dimensional flat patterns for the purpose of facilitating communication between apparel CAD/CAM, ready-to-wear, pattern making systems. The AP supports the capabilities of representing the base pattern geometry, the sizing data, and the grading rules which are based on the traditional X-Y grading methods. It does not support the following capabilities: the relationship between pattern pieces, non x-y grading methods, and alteration options to the pattern piece.

# 3 INFORMATION MODEL

The information model is one of the STEP AP components. It defines the data types that can be used to define products. In this section, we introduce a Ready-to-Wear Apparel Pattern Making (RWPM) information model written in the EXPRESS language. The model describes the information necessary for a ready-to-wear apparel pattern. The model is independent of the integrated resources of STEP.

The EXPRESS information modeling language was developed by ISO as a way to precisely and completely describe all the data elements for defining objects. EXPRESS is described in the "EXPRESS Language Reference Manual" [ISO11]. The exchange medium for STEP product models is the STEP physical file. A STEP physical file contains instances of the various entities defined by the EXPRESS information model. The STEP exchange format, and the mapping from EXPRESS to the STEP physical file, are described in "Clear Text Encoding of the Exchange Structure" [ISO21]. These methodologies form the basis of the APDES project.

An EXPRESS schema is composed of type declarations, entities, and constraints. The RWPM schema is explained in detail in the next section. The following is the list of types and entities, with associated section numbers and page numbers of this documentation, used in this model.

TYPES:


- composite_curve_feature_type (Section 4.1.3; page 5)
- mark_feature_type (Section 4.1.1; page 4)
- measurement_unit (Section 4.1.4; page 5)
- orientation_constraint_type (Section 4.1.2; page 4)
- pattern_mirror_type (Section 4.1.5; page 5)


ENTITIES:


- annotation_feature (Section 4.2.10; page 9)
- arc (Section 4.2.12; page 9)
- basic_pattern_piece (Section 4.2.15; page 10)
- bounded_curve (Section 4.2.11; page 9)
- composite_curve_feature (Section 4.2.14; page 10)
- grade_data_at_point (Section 4.2.20; page 12)
- grade_delta (Section 4.2.19; page 12)
- grade_point (Section 4.2.18; page 11)
- grade_rules_of_piece (Section 4.2.21; page 12)

grade_rules_of_pattern (Section 4.2.22; page 13)
line (Section 4.2.3; page 6)
mark_feature (Section 4.2.5; page 7)
notch_feature (Section 4.2.6; page 7)
orientation_constraint (Section 4.2.9; page 8)
pattern (Section 4.2.17; page 11)
pattern_geometry_entity (Section 4.2.4; page 7)
pattern_piece (Section 4.2.16; page 11)
pattern_size (Section 4.2.1; page 6)
point (Section 4.2.2; page 6)
polyline (Section 4.2.13; page 9)
slit_notch (Section 4.2.8; page 8)
ready_to_wear_pattern (Section 4.2.23; page 13)
v_notch (Section 4.2.7; page 8)

The entities include a broad range of data types, from simple points to complex entities such as patterns. The way these entity classes are related is specified by the model schema, which is described next.

# 4 SCHEMA

This section describes the detailed information for the RWPM schema. Types and entities are defined formally here in EXPRESS. Appendix A contains the listing of EXPRESS keywords that are used in the RWPM schema. The concept of a type in EXPRESS is similar to that of a data type in a standard programming language. It defines the kind of values that an object may assume. Entities are the main building blocks of an EXPRESS information model. An entity declaration describes the information content of an object, as well as some of the constraints on the object. The schema is presented here in a "bottom-up" order: type definitions are presented first, followed by entity definitions. The more specific entity definitions are described before they are used in the definition of more complex entities. In the EXPRESS language, a "remark" is used for documentation and is not significant as a language element. The character pair, "(" and "*", is used to denote the start of an embedded remark, and the character pair, "*" and ")", is used to denote its end. An embedded remark may appear between any two tokens. In this report, the documentation is presented as embedded remarks. Consequently, this entire report can be read into an EXPRESS parser for further analysis [CLA].

```
*)

SCHEMA rwpm_schema;

(*
```

## 4.1 Type Definitions

This section contains the type definitions.

### 4.1.1 Mark Feature Type

A mark_feature_type provides the means to indicate a mark on the pattern. This is used as an aid for subsequent cutting and sewing procedures. Mark_feature_type is an enumeration of drill hole, lift and plunge point, stacking point, facing point, and cut entry point. It is used as the type of an attribute defined in the mark_feature entity.

```
    *)

    TYPE mark_feature_type = ENUMERATION OF (
                drill_hole,
```

```
                lift_and_plunge_point,
                stacking_point,
                facing_point,
                cut_entry_point);
END_TYPE;
```

(*

### 4.1.2 Orientation Constraint Type

An orientation_constraint_type provides a means of specifying the orientation of the pattern piece or supporting symmetrical pattern pieces. The pattern piece may be aligned with the fabric's grain, or with some feature of the fabric's decorative design. Orientation_constraint_type is an enumeration of grain reference line, stripe reference line, plaid reference line, and mirror line. It is used as the type of an attribute defined in the orientation_constraint entity.

*)

```
TYPE orientation_constraint_type = ENUMERATION OF (
                grain_reference_line,
                stripe_reference_line,
                plaid_reference_line,
                mirror_line);
END_TYPE;
```

(*

### 4.1.3 Composite Curve Feature Type

A composite_curve_feature_type provides a means of expressing the purpose of a curve on a pattern piece. This type is an enumeration of boundary cut, internal cut out, seam line, and sew line. It is used as the type of an attribute defined in the composite_curve_feature entity.

*)

```
TYPE composite_curve_feature_type = ENUMERATION OF (
                boundary_cut,
                internal_cut_out,
```

```
                    seam_line,
                    sew_line);
END_TYPE;

(*
```

### 4.1.4 Measurement Unit Type

A measurement_unit provides a means to define the units of length. The units of length refers to the basic units of measurement employed for the given model data. Measurement_unit is an enumeration of centimeter and inch.

```
*)

TYPE measurement_unit = ENUMERATION OF (
                    centimeter,
                    inch);
END_TYPE;

(*
```

### 4.1.5 Pattern Mirror Type

A pattern_mirror_type provides the means to identify the mirror information of the pattern piece. It is an enumeration of basic, horizontal mirror, and vertical mirror. A mirror pattern is a mirror-image of the basic pattern in the horizontal or vertical direction. Pattern_mirror_type is used as the type of an attribute defined in the pattern_piece entity.

```
*)

TYPE pattern_mirror_type = ENUMERATION OF (
                    basic,
                    horizontal_mirror,
                    vertical_mirror);
END_TYPE;

(*
```

## 4.2 Entity Definitions

This section defines the entities for the RWPM schema. All pattern geometry is defined in a Cartesian coordinate system.

### 4.2.1 Pattern Size

A garment size designation is merely an arbitrary name or number for a given compilation of anthropometric measurements. With this designation, the garment will fit someone whose measurements lie within certain range limits of the size measurements. A pattern_size entity is represented in terms of an abstract size number (a real number) and optionally, an alternate size (a text string). Size number may be the same as an actual body dimension (e.g., neck size for men's shirts) or maybe a number that is not the same as the body measurements used to establish the size (e.g., women's dress size). Alternate size may be used to define a size group or to define some body dimensions that are important to the type of garment but not specified in the size number. Size group is used to specify a group which may be based on length (e.g., short, regular, or long), or may be based on proportion (e.g., Junior, Miss, or Woman). Body dimensions may include circumferential designation and longitudinal designation.

```
*)

ENTITY pattern_size;
        size_number: REAL;
        alternate_size: OPTIONAL STRING;
END_ENTITY;

(*
```

### 4.2.2 Point

A point entity specifies a location on a pattern piece. It consists of an X value and a Y value. Coordinates are defined from an unspecified origin, determined by the application.

```
*)

ENTITY point;
        x, y: REAL;
END_ENTITY;
```

(*

### 4.2.3 Line

A line entity defines a line segment. It consists of two points.

*)

```
ENTITY line;
        location1, location2: point;
END_ENTITY;
```

(*

### 4.2.4 Pattern Geometry Entity

A pattern_geometry_entity is an abstraction of mark_feature entity, notch_feature entity, orientation_constraint entity, annotation_feature entity, and composite_curve_feature entity. (Note: The abstract supertype entity is defined for classification purposes only, and is not ever directly instantiated except through its specific subtypes.)

*)

```
ENTITY pattern_geometry_entity
    ABSTRACT SUPERTYPE OF (
        ONEOF (
            mark_feature,
            notch_feature,
            orientation_constraint,
            annotation_feature,
            composite_curve_feature));
END_ENTITY;
```

(*

### 4.2.5 Mark Feature

A mark_feature entity, a subtype of pattern_geometry_entity, specifies a drill hole, a lift and plunge point, a stacking point, a facing point, or a cut entry point on the pattern piece. It is a point together with a mark_feature_type.

*)

```
ENTITY mark_feature
	SUBTYPE OF (pattern_geometry_entity);
		feature_type: mark_feature_type;
		location: point;
END_ENTITY;
```

(*

### 4.2.6 Notch Feature

A notch_feature entity, a subtype of pattern_geometry_entity, specifies a notch on the pattern piece. Notches are represented in a variety of ways. A notch base point may be given with angle and depth information to define an angled slit notch. Alternately, a v-notch is defined by the notch base point with depth and width. Thus a notch_feature is an abstraction of v_notch entity and slit_notch entity. Both sub-types inherit the notch base point attribute.

*)

```
ENTITY notch_feature
	SUBTYPE OF (pattern_geometry_entity)
		ABSTRACT SUPERTYPE OF (
			ONEOF (v_notch,
			       slit_notch));
		notch_base_point: point;
END_ENTITY;
```

(*

### 4.2.7 V Notch

A v_notch entity, a subtype of notch_feature entity, specifies a v-shaped cut on the pattern piece. It is defined by depth and width of the notch.

*)

```
ENTITY v_notch
        SUBTYPE OF (notch_feature);
                depth, width: REAL;
END_ENTITY;

(*
```

### 4.2.8 Slit Notch

A slit_notch entity, a subtype of notch_feature entity, specifies an angled slit cut on the pattern piece. It is defined by a depth value and an angle of a notch.

```
*)

ENTITY slit_notch
        SUBTYPE OF (notch_feature);
                depth, angle: REAL;
END_ENTITY;

(*
```

### 4.2.9 Orientation Constraint

An orientation_constraint entity, a subtype of pattern_geometry_entity, is a direction specification on the pattern piece. It is characterized by a line,that defines the location of the orientation on the drawing, together with an orientation_constraint_type.

```
*)

ENTITY orientation_constraint
        SUBTYPE OF (pattern_geometry_entity);
                feature_type: orientation_constraint_type;
                location: line;
END_ENTITY;

(*
```

### 4.2.10 Annotation Feature

An annotation_feature entity, a subtype of pattern_geometry_entity, is an annotation feature on the pattern piece presented for informational purposes; it does not represent a feature of the cut piece. It consists of a text string and a line for locating and orienting text. The annotation_feature entity may be used to define style lines, user defined internal lines, etc.

```
*)

ENTITY annotation_feature
        SUBTYPE OF (pattern_geometry_entity);
                text: STRING;
                location: line;
END_ENTITY;

(*
```

### 4.2.11 Bounded Curve

A bounded_curve entity is an abstraction of arc entity and polyline entity.

```
*)

ENTITY bounded_curve
        ABSTRACT SUPERTYPE OF (
                ONEOF (arc,
                       polyline));
END_ENTITY;

(*
```

### 4.2.12 Arc

An arc entity, a subtype of bounded_curve entity, specifies an arch-shaped segment of a curve. It consists of three points: a start point, an intermediate point, and an end point.

```
*)

ENTITY arc
```

```
        SUBTYPE OF (bounded_curve);
                start_point, intermediate_point, end_point: point;
END_ENTITY;

(*
```

### 4.2.13 Polyline

A polyline entity, a subtype of bounded_curve entity, is a bounded curve of line segments. It consists of an ordered collection of points that are connected using straight lines.

```
*)

ENTITY polyline
        SUBTYPE OF (bounded_curve);
                points: LIST [2:?] of point;
END_ENTITY;

(*
```

### 4.2.14 Composite Curve Feature

A composite_curve_feature entity, a subtype of pattern_geometry_entity, is a geometry entity that defines a curve in the drawing. It is a composite curve (an ordered collection of bounded_curves joined end to end) together with a composite_curve_feature_type.

```
*)

ENTITY composite_curve_feature
        SUBTYPE OF (pattern_geometry_entity);
                feature_type: composite_curve_feature_type;
                composite_curve: LIST [1:?] of UNIQUE bounded_curve;
END_ENTITY;

(*
```

### 4.2.15 Basic Pattern Piece

A basic_pattern_piece entity defines the base shape of one pattern piece of a garment for a particular size. It has identification information, a set of pattern_geometry_entities, and optionally, a tolerance value (a real number). The identification information includes a piece name and optionally, a description. The piece name is a unique pattern piece name within a pattern. The description is a piece identification for the operator to use. If a tolerance value is defined in a pattern entity instance, the value is applied to all pattern pieces of the garment. However, the basic_pattern_piece allows this value to be redefined. Thus if a tolerance value is defined in a basic_pattern_piece entity instance, it overrides the tolerance value that is given at the pattern entity instance. A basic_pattern_piece may be repeated either as is or in a mirrored form for one garment (e.g. identical, but mirrored, pieces for shirt sleeves).

```
*)

ENTITY basic_pattern_piece;
        piece_name: STRING;
        description: OPTIONAL STRING;
        geometry_entities: SET [1:?] OF pattern_geometry_entity;
        tolerance: OPTIONAL REAL;
        UNIQUE   piece_name;
END_ENTITY;

(*
```

### 4.2.16 Pattern Piece

A pattern_piece entity defines a pattern piece of a garment for a particular size and the total number of this piece required in the garment. It is defined by a basic_pattern_piece with a pattern_mirror_type, and a quantity of the pattern piece.

```
*)

ENTITY pattern_piece;
        piece: basic_pattern_piece;
        mirror_type: pattern_mirror_type;
        quantity: INTEGER;
END_ENTITY;

(*
```

### 4.2.17 Pattern

A pattern entity belongs to a garment and defines the garment shape for a particular size. It is defined by a garment style name, a garment description for the operator to use, a set of pattern_pieces, and optionally, a tolerance value. If a tolerance value is defined in a pattern entity instance, the value is applied to all pattern pieces of the garment. However, the basic_pattern_piece allows this value to be redefined.

```
*)

ENTITY pattern;
        style_name: STRING;
        description: OPTIONAL STRING;
        tolerance: OPTIONAL REAL;
        pattern_pieces: SET [1:?] OF pattern_piece;
END_ENTITY;

(*
```

### 4.2.18 Grade Point

A grade_point entity specifies a grade point on a pattern piece. Points on a pattern piece may or may not be subject to a grading rule. A grade_point is defined by a point, and a unique grade point identifier.

*)

```
ENTITY grade_point;
        location: point;
        identifier: STRING;
        UNIQUE location, identifier;
END_ENTITY;
```

(*

### 4.2.19 Grade Delta

A grade_delta entity specifies a displacement vector on a drawing. It consists of a delta-X value and a delta-Y value. The delta-X and delta-Y values are the amounts of growth in the X and Y directions at the grading point between two grading sizes.

*)

```
ENTITY grade_delta;
        delta_x, delta_y: REAL;
END_ENTITY;
```

(*

### 4.2.20 Grade Data At Point

A grade_data_at_point entity is an ordered collection of the displacements of the specified grading point for a set of predefined grading sizes. It is defined by a grade_point, a smooth option (which provides a means of determining that the grading curve will be generated with or without smoothing at the grade point), a set of grade_delta, and optionally, an alternate grade reference line. If an alternate grade reference line is defined in an entity instance, it overrides the grade reference line of the pattern piece that is given at the grade_rules_of_piece entity instance.

*)

```
ENTITY grade_data_at_point;
```

```
point: grade_point;
smooth_option: BOOLEAN;
grade_deltas: LIST [1:?] OF grade_delta;
alternate_grade_reference_line: OPTIONAL line;
END_ENTITY;

(*
```

### 4.2.21 Grade Rules Of Piece

A grade_rules_of_piece entity contains the information for grading a pattern piece for a set of predefined sizes. It consists of a piece name, a set of grade_data_at_point, and optionally, a rules identifier and a grade reference line. If a grade reference line is defined in an entity instance, it is applied to all grade points of the pattern piece.

```
*)

ENTITY grade_rules_of_piece;
        piece_name: STRING;
        identifier: OPTIONAL STRING;
        grade_reference_line: OPTIONAL line;
        rules: SET OF grade_data_at_point;
        UNIQUE piece_name, identifier;
END_ENTITY;

(*
```

### 4.2.22 Grade Rules Of Pattern

A grade_rules_of_pattern entity contains the information for grading a pattern for a set of predefined sizes. It consists of an ordered collection of pattern sizes that include the base size, a set of grade_rules_of_piece, and optionally, a rules identifier.

```
*)

ENTITY grade_rules_of_pattern;
        identifier: OPTIONAL STRING;
```

```
        pattern_sizes: LIST [2:?] OF UNIQUE pattern_size;
        rules: SET OF grade_rules_of_piece;
END_ENTITY;

(*
```

### 4.2.23 Ready-To-Wear Pattern

A ready_to_wear_pattern entity defines a graded pattern for all sizes. It is defined by a pattern, a pattern_size (the size of the base pattern), and optionally, a grade_rules_of_pattern. The ready_to_wear_pattern is defined with a common unit of measurement.

```
*)

ENTITY ready_to_wear_pattern;
        unit: measurement_unit;
        base_size: pattern_size;
        base_pattern: pattern;
        grade_rules: OPTIONAL grade_rules_of_pattern;
END_ENTITY;


END_SCHEMA; -- end rwpm_schema

(*
```

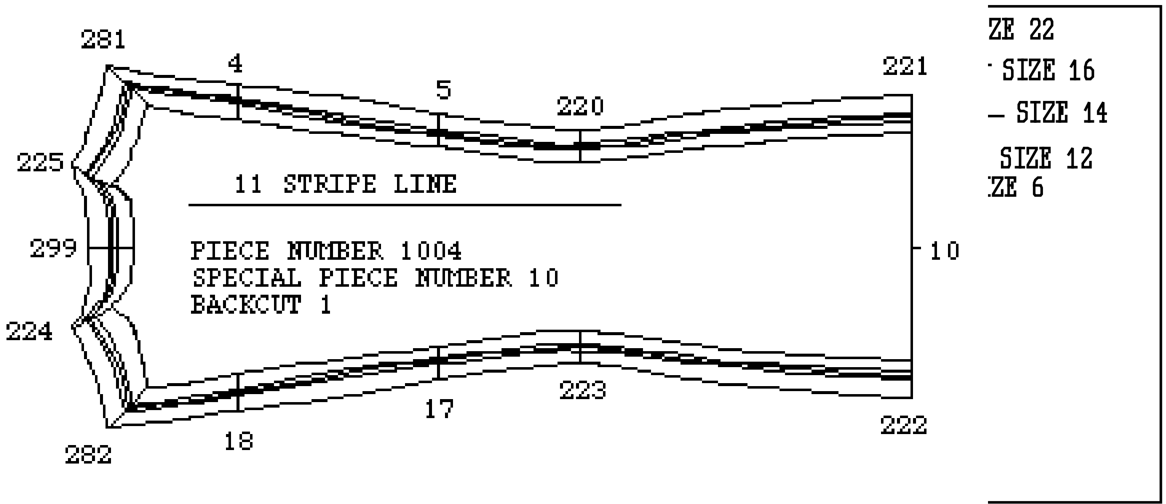
# 5 SAMPLE EXCHANGE FILE

A sample pattern piece is provided for illustrating an example of a STEP exchange structure physical file (STEP file). Figure 1 presents a sketch of the sample pattern piece. The same piece was represented in a STEP file based on the RWPM schema presented above. Appendix B contains the listing of the STEP file. The file contains both the data for the base pattern piece as well as grading information for that pattern piece.

Each STEP file consists of two sections: the HEADER section and DATA section. The file shall begin with "ISO-10303-21;", and be terminated by "END-ISO-10303-21;". The HEADER section contains information that is applicable to the entire exchange structure. The DATA section contains the product data to be transferred, it contains instances of entities which correspond to an EXPRESS schema. The STEP file format is considered to be a continuous stream of characters from the basic alphabet. These characters are collected into recognizable sequences of characters called tokens (keywords or simple data types). When printing the STEP file, the print control directives (such as a new line) are used to aid readability. A STEP entity instance name consists of a "number" sign (#) followed by an unsigned integer of 1 to 9 digits. Entity instance names are used as identifiers when they appear in the exchange file just before the equal sign in an entity instance. Entity instance names are also used as references to other entities when they appear inside of an entity's attribute list. Both forward and backward references are permitted.

**Nest of "Pattern Piece 1004" for base size (14)**
**and four graded sizes (6, 12, 16, 22)**

**Figure 1: Sketch of Sample Pattern Piece**




# 6 CONFORMANCE TESTING

This section specifies requirements and procedures for conformance testing of applications that implement the RWPM Prototype AP. Note that the Conformance Testing (CT) procedures specified do not require special software to be used to assist in the testing. However, special CT software could improve the efficiency and quality of testing.

Section 6.1 introduces a number of new acronyms and definitions. Section 6.2 specifies the requirements of the application that is being tested for conformance with the AP. The application being tested is called the Implementation Under Test (IUT). Section 6.3 specifies the requirements of the system that must be used to test conformance of the IUT. Finally, section 6.4 specifies the responsibilities of the Test Site and the Developer, and specifies the procedures that will be used to test the IUT.

## 6.1 Acronyms And Definitions

| | |
|---|---|
| AP file | STEP physical file for RWPM AP |
| CT | Conformance Testing |
| Developer | Vendor who is developing application software |

| | |
|---|---|
| IUT | Implementation Under Test. The application software that has implemented the AP and is subject to conformance testing |
| Test Site | Location where CT is done |

## 6.2 IUT Requirements

### 6.2.1 General

• Fully conforms with AP specifications (partial conformance not acceptable)

• Conforms with STEP physical file format as described in the STEP documentation [ISO21]

### 6.2.2 AP Input

• Reads properly formatted and syntactically correct AP files

• Recognizes improperly formatted or syntactically incorrect AP files

• Recognizes errors in entity values and relationships as specified by the AP

### 6.2.3 AP Output

• Generates AP files that are properly formatted and syntactically correct

• Can generate every entity specified by the AP

### 6.2.4 AP File Integrity

• Recognizes all AP entities (i.e. ability to use them)[7]

• Generates correct entity values and relationships (i.e. modifications made by IUT on entities

[7]Ability to modify every AP entity gives some indication of satisfying this requirement.

and entity relationships are done correctly within bounds specified by the AP)

## 6.3 Requirements for System to Test IUT

The system to test IUTs will have two components:

- AP file generator to generate test AP files for the IUT to read[8]

- AP file tester to test the AP files that the IUT generates

### 6.3.1 AP File Generator

- Generates set of AP files

- Generates all AP entities within set of files produced

- Generates IUT test procedures that are customized to set of files produced[9]

### 6.3.2 AP File Tester

- Configures testing based on test procedures produced by AP file generator

- Checks if test set of AP output files generated by IUT during CT has correct syntax

- Checks that every entity is in test set

- Checks entity and relationship correctness

- Produces output report:

---

[8]AP files generated by the Test Site's file generator or by the Developer's IUT are written to PC-DOS formatted diskettes. Other formats can be negotiated.

[9]The document includes procedures that specify operations that the IUT should do after inputting the AP input files, and before outputting the AP output files. The output files will be tested by the AP file tester.

- Pass or fail
- If fail, reasons for failure

## 6.4 Conformance Testing Procedure[10]

### 6.4.1 Test Site Responsibilities

- Maintain a CT system accredited by suitable authority and agreeable to all parties

- Use personnel for IUT testing who are unbiased towards Developer and who are sufficiently versed in IUT application area to conduct CT with brief training

- Provide Developer with test results and archive at Test Site (not for distribution)

- Certify IUTs that have passed CT, and authorize Developers to publicize that fact

- Maintain confidentiality of test results

### 6.4.2 Developer Responsibilities

- Deliver IUT to Test Site. Delivery must include hardware if other suitable arrangements cannot be made (e.g. if hardware required is unavailable at Test Site)

- Demonstrate system to Test Site personnel sufficiently, so that latter can run CT tests

- May display CT certification only on versions of products that have passed CT

### 6.4.3 Procedures

The following procedures are conducted at the Test Site.

[10]The Conformance Testing procedure for STEP APs will be standardized in ISO 10303-32 [ISO32].

- Receive delivery of IUT from Developer
    - Install IUT with Developer's assistance
    - Receive Developer training to enable IUT test personnel to conduct CT

- Create input files for IUT
    - Generate set of AP files
        - Contains all AP entities
        - Includes incorrect files
    - Generate IUT test procedures document (archive)

- Execute IUT
    - Execute procedures from test procedures document
    - Generate output specified from the procedures document

- Check the AP files produced by the IUT
    - Use input consisting of:
        - AP input files to IUT (generated by AP File Generator)
        - AP output files, generated in previous step by IUT
        - Test procedures document, likewise generated in previous step
    - Generate report that documents test results

- Process results
    - Send report of test results to Developer
    - Send acceptance certification to Developer if IUT passed CT
    - Archive test results at Test Site

## APPENDIX A: REFERENCES:


[CLA] Clark, Stephen N., *An Introduction to the NIST PDES Toolkit,* NISTIR 4336, National Institute of Standards and Technology, Gaithersburg, MD, May 1990.

[DLA] Defense Logistics Agency, *Third Annual Academic Apparel Research Conference*, Manufacturing Technology Information Analysis Center, 10 West 35th Street, Chicago, IL 60616-3799, February 1992.

[ISO11] ISO DIS 10303-11, *Product Data Representation and Exchange - Part 11: The EXPRESS Language Reference Manual*, ISO, August 1992.

[ISO21] ISO CD 10303-21, *Product Data Representation and Exchange - Part 21: Clear Text Encoding of the Exchange Structure*, ISO, March 1991.

[ISO31] ISO CD 10303-31, *Product Data Representation and Exchange - Part 31: Conformance Testing Methodology and Framework: General Concepts*, ISO, January 14, 1992.

[ISO32] ISO Working Draft, *Product Data Representation and Exchange - Part 32: Conformance Testing Methodology and Framework: Requirements on Testing Laboratories and Clients*, ISO TC184/SC4/WG6 N 52, ISO, December 1992.

[KEM] Kemmerer, S., editor, *Requirements and Recommendations for STEP Conformance Testing*, NISTIR 4743 (Revised), National Institute of Standards and Technology, Gaithersburg, MD, June 1992.

[KRA] Kramer, T. R., et al., *Issues and Recommendations for a STEP Application Protocol Framework*, NISTIR 4755, National Institute of Standards and Technology, Gaithersburg, MD, January 1992.

[LEE1] Lee, Y. T., *On Extending the Standard for the Exchange of Product Data to Represent Two-Dimensional Apparel Pattern Pieces*, NISTIR 4358, National Institute of Standards and Technology, Gaithersburg, MD, June 1990.

[LEE2] Lee, Y. T., "Apparel Product Data Exchange Standard," Proceedings of the *Third Annual Academic Apparel Research Conference on Implementing Advanced Technology*, Atlanta, GA, February 1992.

[MON] Moncarz, H. T. and Lee, Y. T., *Apparel STEP Translator*, NISTIR 4612, National Institute of Standards and Technology, Gaithersburg, MD, June 1991.

[NIJ] Nijssen, G. M. and Halpin, T. A., *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*, Prentice Hall, 1989.

[PAL] Palmer, M., *Guidelines for the Development and Approval of STEP Application Protocols*, ISO TC184/SC4/WG4 N 25 (P5), ISO, September 1991.

[USA] U.S. Air Force Wright Aeronautical Laboratories, *Information Modeling Manual IDEF1-Extended (IDEF1x)*, Report AFWAL-TR-86-4006, Volume 5, Part 4, Manufacturing Technology Directorate, U.S. Air Force Wright Aeronautical Laboratories 1986.

## APPENDIX B: GLOSSARY OF EXPRESS TERMS

The following is the list of EXPRESS keywords that are used in the RWPM model. Brief definitions of these keywords are derived from the "The EXPRESS Language Reference Manual" [ISO11] and are presented below for the reader's convenience.

ABSTRACT SUPERTYPE An ABSTRACT SUPERTYPE is a SUPERTYPE entity that is not intended to be directly instantiated.

BAG The key word BAG is used to specify a bag data type. A bag data type represents an unordered collections of like elements. The number of elements that can be held in a bag can optionally be specified. If the size is not specified, the bag can hold any number of elements. Duplicate elements are allowed in a bag.

BOOLEAN A BOOLEAN data type represents a TRUE or FALSE value.

END_ENTITY The key word END_ENTITY is used to terminate an entity declaration.

END_SCHEMA The key word END_SCHEMA is used to terminate a schema declaration.

END_TYPE The key word END_TYPE is used to terminate a type declaration.

| | |
|---|---|
| ENTITY | The key word ENTITY is used to specify an entity type. An entity type characterizes a collection of real-world physical or conceptual objects which have common properties. Any entity declared in a schema can be used as the data type of an attribute, local variable or formal parameter. Using an entity as an attribute's data type establishes a relationship between the two entities. |
| ENUMERATION | The key word ENUMERATION is used to specify an enumeration data type. An enumeration data type is an ordered set of values represented by names. Each enumeration item belongs only to the data type which defines it and must be unique within that type definition. |
| INTEGER | The key word INTEGER is used to specify an integer data type. An integer data type represents a value of an integer number, the magnitude of which is unconstrained. |
| LIST | The key word LIST is used to specify a list data type. A list data type represents an ordered collections of like elements. The number of elements that can be held in a list can optionally be specified. If the size is not specified, the list can hold any number of elements. Duplicate elements are allowed in a list. |
| OF | The key word OF is used together with other keywords such as BAG, LIST, SET, ENUMERATION, SUBTYPE, SUPERTYPE, etc. |
| ONEOF | The key word ONEOF is used to define the constraint on the relationship between the subtypes of a particular supertype. The ONEOF constraint is used when the subtypes are mutually exclusive. |
| OPTIONAL | The key word OPTIONAL is used to indicate that the attribute need not have a value in order for an instance of that entity to be valid. In a given entity instance, an attribute marked as optional may have no actual value, in which case the value is said to be null. The null value function (NVL) which returns |

| | |
|---|---|
| | either the input value or an alternate value in the case where the input has a null value may be used when a null value is unacceptable. |
| REAL | The key word REAL is used to specify a real data type. A real data type represents rational, irrational, and scientific real numbers. Rational and irrational numbers have infinite resolution and are exact. Scientific numbers represent values which are known only to a specified precision. |
| SCHEMA | The key word SCHEMA is used to specify a schema type. A schema declaration creates a new scope in which the following objects may be declared: constant, entity, function, procedure, rule, and type. |
| SET | The key word SET is used to specify a set data type. A set data type represents an unordered collections of like elements. The number of elements that can be held in a set can optionally be specified. If the size is not specified, the set can hold any number of elements. No two elements of a set can have the same value. |
| STRING | The key word STRING is used to specify a string data type. A string data type represents a sequence of zero or more characters. |
| SUBTYPE | The key words SUBTYPE and SUPERTYPE are used for classification purposes. A subtype is a more specific type than its supertype(s). A supertype is a more general type than its subtype(s). Thus, every instance of a subtype is an instance of its supertype(s). An entity is a subtype if and only if it contains a non-empty subtype clause. An entity does not have to declare itself to be a supertype. The subtype/supertype relationship is transitive. An entity is a supertype if it is named in the subtype clause of at least one other entity or declares itself to be an abstract supertype. A subtype may have more than one supertype, and a supertype may have more than one subtype. A supertype may itself be a subtype of one or more other entity types. Furthermore, a subtype cannot be the supertype of any type in the list of all its supertypes. |

| | |
|---|---|
| SUPERTYPE | See "SUBTYPE". |
| TYPE | The key word TYPE is used to specify a defined data type. A defined data type is a user extension to the set of standard data types. A defined data type can be used as any other data type by referencing the name given to it. |
| UNIQUE | The key word UNIQUE is used to specify a unique rule. A unique rule specifies either a single attribute name or a list of two or more attribute names. A rule which specifies a single attribute name is a "simple uniqueness constraint", requiring that any value of that attribute is associated with only one instance of that entity type. A rule which specifies two or more attribute names is a "joint uniqueness constraint", requiring that any set of values, one from each of the named attributes, is associated with only one instance of that entity type. |

## APPENDIX C: SAMPLE STEP EXCHANGE FILE

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('This file contains a sample RWPM STEP file'),
'RWPM_LEVEL 1.0');
FILE_NAME('Example RWPM File No.1',
'1992-09-18 T15:05:00',
('Tina Lee ','NIST Building 220, Room A-127','Gaithersburg, MD 20899'),
('National Institute of Standards and Technology',
'Factory Automation Systems Division'),
'RWPM Version 1.0',
'APDES RWPM Release 1.0',
'Approved by Tina Lee');
FILE_SCHEMA(('RWPM_SCHEMA'));
ENDSEC;
DATA;
#1001=POINT(23.62,16.24);
#1002=POINT(22.88,18.76);
#1003=POINT(24.10,21.90);
#1004=POINT(27.91,21.29);
#1005=POINT(34.86,20.24);
#1006=POINT(39.92,19.73);
#1007=POINT(51.40,20.77);
#1008=POINT(51.49,16.24);
#1009=POINT(51.40,11.71);
#1010=POINT(39.92,12.75);
#1011=POINT(34.86,12.24);
#1012=POINT(27.91,11.19);
#1013=POINT(24.10,10.58);
#1014=POINT(22.88,13.72);
#1015=POINT(26.36,17.73);
#1016=POINT(23.62,17.13);
#1017=POINT(23.55,17.52);
#1018=POINT(23.43,17.94);
#1019=POINT(23.14,18.42);
#1020=POINT(23.37,19.52);
#1021=POINT(23.69,20.23);
#1022=POINT(23.91,20.92);
```

```
#1023=POINT(25.99,21.60);
#1024=POINT(30.19,20.93);
#1025=POINT(32.42,20.59);
#1026=POINT(36.71,19.99);
#1027=POINT(38.69,19.76);
#1028=POINT(39.40,19.72);
#1029=POINT(41.17,19.83);
#1030=POINT(43.07,20.10);
#1031=POINT(45.09,20.37);
#1032=POINT(47.42,20.57);
#1033=POINT(49.38,20.69);
#1034=POINT(49.38,11.79);
#1035=POINT(47.42,11.91);
#1036=POINT(45.09,12.11);
#1037=POINT(43.07,12.38);
#1038=POINT(41.17,12.65);
#1039=POINT(39.40,12.76);
#1040=POINT(38.69,12.72);
#1041=POINT(36.71,12.49);
#1042=POINT(32.42,11.89);
#1043=POINT(30.19,11.55);
#1044=POINT(25.99,10.88);
#1045=POINT(23.91,11.56);
#1046=POINT(23.69,12.25);
#1047=POINT(23.37,12.96);
#1048=POINT(23.14,14.06);
#1049=POINT(23.43,14.54);
#1050=POINT(23.55,14.96);
#1051=POINT(23.62,15.35);
#1052=POINT(41.56,17.76);
#1201= &SCOPE
        #101=POLYLINE((#1001,#1016,#1017,#1018,#1019,#1002));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#101);
#1202= &SCOPE
        #102=POLYLINE((#1002,#1020,#1021,#1022,#1003));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#102);
#1203= &SCOPE
```

```
        #103=POLYLINE((#1003,#1023,#1004));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#103);
#1204= &SCOPE
        #104=POLYLINE((#1004,#1024,#1025,#1005));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#104);
#1205= &SCOPE
        #105=POLYLINE((#1005,#1026,#1027,#1028,#1006));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#105);
#1206= &SCOPE
        #106=POLYLINE((#1006,#1029,#1030,#1031,#1032,#1033,#1007));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#106);
#1207= &SCOPE
        #107=POLYLINE((#1007,#1008));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#107);
#1208= &SCOPE
        #108=POLYLINE((#1008,#1009));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#108);
#1209= &SCOPE
        #109=POLYLINE((#1009,#1034,#1035,#1036,#1037,#1038,#1010));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#109);
#1210= &SCOPE
        #110=POLYLINE((#1010,#1039,#1040,#1041,#1011));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#110);
#1211= &SCOPE
        #111=POLYLINE( (#1011,#1042,#1043,#1012));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#111);
#1212= &SCOPE
        #112=POLYLINE((#1012,#1044,#1013));
        ENDSCOPE
        COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#112);
```

```
#1213= &SCOPE
              #113=POLYLINE((#1013,#1045,#1046,#1047,#1014));
              ENDSCOPE
             COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#113);
#1214= &SCOPE
              #114=POLYLINE((#1014,#1048,#1049,#1050,#1051,#1001));
              ENDSCOPE
              COMPOSITE_CURVE_FEATURE(.BOUNDARY_CUT.,#114);
#1215= &SCOPE
              #115=LINE(#1015,#1052);
              ENDSCOPE
              ORIENTATION_CONSTRAINT(.STRIPE_REFERENCE_LINE.,#115);
#1216=V_NOTCH(#1001,0.1875,0.25);
#1217=V_NOTCH(#1004,0.1875,0.25);
#1218=V_NOTCH(#1005,0.1875,0.25);
#1219=V_NOTCH(#1006,0.1875,0.25);
#1220=V_NOTCH(#1008,0.1875,0.25);
#1221=V_NOTCH(#1010,0.1875,0.25);
#1222=V_NOTCH(#1011,0.1875,0.25);
#1223=V_NOTCH(#1012,0.1875,0.25);
#1301=BASIC_PATTERN_PIECE('BACK CUT 1',
                $,
                (#1201,#1202,#1203,#1204,#1205,#1206,#1207,#1208,
                 #1209,#1210,#1211,#1212,#1213,#1214,#1215,#1216,
                 #1217,#1218,#1219,#1220,#1221,#1222,#1223),
                $);
#1401=PATTERN_PIECE(#1301,.BASIC.,1);
#1501=PATTERN('style_name',$,$,(#1401));
#2001=GRADE_POINT(#1001,'299');
#2002=GRADE_POINT(#1002,'225');
#2003=GRADE_POINT(#1003,'281');
#2004=GRADE_POINT(#1004,'4');
#2005=GRADE_POINT(#1005,'5');
#2006=GRADE_POINT(#1006,'220');
#2007=GRADE_POINT(#1007,'221');
#2008=GRADE_POINT(#1008,'10');
#2009=GRADE_POINT(#1009,'222');
#2010=GRADE_POINT(#1010,'223');
#2011=GRADE_POINT(#1011,'17');
```

```
#2012=GRADE_POINT(#1012,'18');
#2013=GRADE_POINT(#1013,'282');
#2014=GRADE_POINT(#1014,'224');
#2015=GRADE_POINT(#1015,'11');
#2101= &SCOPE
            #201=GRADE_DELTA(-0.0517,0.0000);
            #202=GRADE_DELTA(-0.0650,0.0000);
            #203=GRADE_DELTA(-0.0600,0.0000);
            #204=GRADE_DELTA(-0.0683,0.0000);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2001,.F.,(#201,#202,#203,#204),$);
#2102= &SCOPE
            #205=GRADE_DELTA(-0.0567,0.0317);
            #206=GRADE_DELTA(-0.0650,0.0300);
            #207=GRADE_DELTA(-0.0600,0.0300);
            #208=GRADE_DELTA(-0.0633,0.0317);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2002,.F.,(#205,#206,#207,#208),$);
#2103=  &SCOPE
             #209=GRADE_DELTA(-0.0633,0.0633);
             #210=GRADE_DELTA(-0.0650,0.0650);
             #211=GRADE_DELTA(-0.0600,0.0600);
             #212=GRADE_DELTA(-0.0467,0.0467);
             ENDSCOPE
             GRADE_DATA_AT_POINT(#2003,.F.,(#209,#210,#211,#212),$);
#2104= &SCOPE
            #213=GRADE_DELTA(0.0000,0.0367),
            #214=GRADE_DELTA(0.0000,0.0550);
            #215=GRADE_DELTA(0.0000,0.0550);
            #216=GRADE_DELTA(0.0000,0.0567);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2004,.T.,(#213,#214,#215,#216),$);
#2105= &SCOPE
           #217=GRADE_DELTA(0.0000,0.0367),
           #218=GRADE_DELTA(0.0000,0.0550);
           #219=GRADE_DELTA(0.0000,0.0550);
           #220=GRADE_DELTA(0.0000,0.0567);
           ENDSCOPE
           GRADE_DATA_AT_POINT(#2005,.T.,(#217,#218,#219,#220),$);
```

```
#2106= &SCOPE
            #221=GRADE_DELTA(0.0000,0.0367),
            #222=GRADE_DELTA(0.0000,0.0550);
            #223=GRADE_DELTA(0.0000,0.0550);
            #224=GRADE_DELTA(0.0000,0.0567);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2006,.T.,(#221,#222,#223,#224),$);
#2107= &SCOPE
            #225=GRADE_DELTA(0.0000,0.0367),
            #226=GRADE_DELTA(0.0000,0.0550);
            #227=GRADE_DELTA(0.0000,0.0550);
            #228=GRADE_DELTA(0.0000,0.0567);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2007,.F.,(#225,#226,#227,#228),$);
#2108= &SCOPE
            #229=GRADE_DELTA(0.0000,0.0000),
            #230=GRADE_DELTA(0.0000,0.0000);
            #231=GRADE_DELTA(0.0000,0.0000);
            #232=GRADE_DELTA(0.0000,0.0000);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2008,.T.,(#229,#230,#231,#232),$);
#2109= &SCOPE
            #233=GRADE_DELTA(0.0000,-0.0367),
            #234=GRADE_DELTA(0.0000,-0.0550);
            #235=GRADE_DELTA(0.0000,-0.0550);
            #236=GRADE_DELTA(0.0000,-0.0567);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2009,.F.,(#233,#234,#235,#236),$);
#2110=  &SCOPE
            #237=GRADE_DELTA(0.0000,-0.0367),
            #238=GRADE_DELTA(0.0000,-0.0550);
            #239=GRADE_DELTA(0.0000,-0.0550);
            #240=GRADE_DELTA(0.0000,-0.0567);
            ENDSCOPE
            GRADE_DATA_AT_POINT(#2010,.T.,(#237,#238,#239,#240),$);
#2111= &SCOPE
           #241=GRADE_DELTA(0.0000,-0.0367),
           #242=GRADE_DELTA(0.0000,-0.0550);
           #243=GRADE_DELTA(0.0000,-0.0550);
```

```
        #244=GRADE_DELTA(0.0000,-0.0567);
        ENDSCOPE
        GRADE_DATA_AT_POINT(#2011,.T.,(#241,#242,#243,#244),$);
#2112= &SCOPE
        #245=GRADE_DELTA(0.0000,-0.0367),
        #246=GRADE_DELTA(0.0000,-0.0550);
        #247=GRADE_DELTA(0.0000,-0.0550);
        #248=GRADE_DELTA(0.0000,-0.0567);
        ENDSCOPE
        GRADE_DATA_AT_POINT(#2012,.T.,(#245,#246,#247,#248),$);
#2113= &SCOPE
         #249=GRADE_DELTA(-0.0633,-0.0633),
         #250=GRADE_DELTA(-0.0650,-0.0650);
         #251=GRADE_DELTA(-0.0600,-0.0600);
         #252=GRADE_DELTA(-0.0467,-0.0467);
         ENDSCOPE
        GRADE_DATA_AT_POINT(#2013,.F.,(#249,#250,#251,#252),$);
#2114= &SCOPE
         #253=GRADE_DELTA(-0.0567,-0.0317),
         #254=GRADE_DELTA(-0.0650,-0.0300);
         #255=GRADE_DELTA(-0.0600,-0.0300);
         #256=GRADE_DELTA(-0.0633,-0.0317);
         ENDSCOPE
         GRADE_DATA_AT_POINT(#2014,.F.,(#253,#254,#255,#256),$);
#2115= &SCOPE
         #257=GRADE_DELTA(0.0000,0.0000),
         #258=GRADE_DELTA(0.0000,0.0000);
         #259=GRADE_DELTA(0.0000,0.0000);
         #260=GRADE_DELTA(0.0000,0.0000);
         ENDSCOPE
         GRADE_DATA_AT_POINT(#2015,.F.,(#257,#258,#259,#260),$);
#2201=PATTERN_SIZE(6.0,'REGULAR');
#2202=PATTERN_SIZE(12.0,'REGULAR');
#2203=PATTERN_SIZE(14.0,'REGULAR');
#2204=PATTERN_SIZE(16.0,'REGULAR');
#2205=PATTERN_SIZE(22.0,'REGULAR');
#2301=GRADE_RULES_OF_PIECE('BACK_CUT_1',$,$,
            (#2101,#2102,#2103,#2104,#2105,#2106,#2107,#2108,#2109,
             #2110,#2111,#2112,#2113,#2114,#2115));
```

```
#2401=GRADE_RULES_OF_PATTERN($,(#2201,#2202,#2203,#2204,#2205),(#2301));
#3301=READY_TO_WEAR_PATTERN(.INCH.,#2203,#1501,#2401);
ENDSEC;
END-ISO-10303-21;

*)
```