Automated performance measure template with metadata

Derek McColl
DRDC – Toronto Research Centre

Defence Research and Development Canada

Reference Document
DRDC-RDDC-2017-D010
February 2017

Abstract

Defence Research and Development Canada is developing automated performance measures to support the Royal Canadian Air Force Simulation Strategy. These automated performance measures will assist instructors in making accurate assessments of trainee performance during simulations and exercises. Partnering nations in The Technical Cooperation Panel are also developing automated performance measures. To facilitate the sharing of automated performance measures between these nations, this Reference Document presents a template with metadata that will ensure automated performance measures are easy to understand and consistently developed. The template consists of two main components: the metadata and the programming code. The metadata includes information on the automated performance measure's reference data (e.g., title, author, date created), required software information (inputs, outputs), training objective details (performance standards and frameworks, skills, knowledge, etc.), and required operators and testing scenarios (crew and mission types). This template has been applied to five automated performance measures developed in the Coalition Performance Evaluation Tracking System's C++ application programming interface.

Résumé

Recherche et développement pour la défense Canada élabore actuellement des mesures automatisées du rendement en vue d'appuyer la Stratégie de simulation de l'Aviation royale canadienne. Ces mesures automatisées permettront aux instructeurs de produire une évaluation précise du rendement des stagiaires au cours de simulations et d'exercices. Les pays partenaires membres du Programme de coopération technique s'affairent également à mettre au point de telles mesures automatisées du rendement. Pour faciliter le partage de ces mesures entre pays membres, le présent document de référence décrit un modèle avec métadonnées pour faciliter leur compréhension et uniformiser leur élaboration. Le modèle comporte deux éléments principaux, soit les métadonnées et le code de programmation. Les métadonnées incluent l'information sur les données de référence de la mesure automatisée du rendement (son nom, son auteur, sa date de création, etc.), les renseignements nécessaires sur les logiciels (entrées, sorties) et les détails sur les objectifs de la formation (normes et cadres de rendement, habiletés, connaissances, etc.), ainsi que les opérateurs et les scénarios d'essai requis (types d'équipage et de mission). Le modèle a été appliqué à cinq mesures automatisées du rendement élaborées dans l'interface de programmation d'applications C++ du Système de suivi des évaluations du rendement de la Coalition.

Table of Contents

List of Tables

1 Introduction

Automated performance measures and associated metadata are being developed as a part of Defence Research and Development Canada's (DRDC's) Virtual Skies 03CE project to support the Royal Canadian Air Force (RCAF) Simulation Strategy. These automated performance measures address no fewer than six of ten strategic requirements for simulation focused training and will assist instructors in making accurate assessments of trainee performance during simulations and exercises. The development of automated performance measures is a task being undertaken by nations participating in The Technical Cooperation Program (TTCP). This Reference Document aims to facilitate sharing between these nations by presenting a common automated performance measure template with metadata that will ensure measures are consistently developed, easy to understand and implement.

The template and metadata presented in this Reference Document is based on the development of automated performance measures for The Coalition Performance Evaluation Tracking System (C-PETS) [1]. C-PETS developed by the United States Air Force Research Laboratory (AFRL), is a Distributed Interactive Simulation (DIS) network integrated software application that captures, displays, and stores objective warfighter performance data in real-time during a simulation. C-PETS reads and analyzes the DIS network simulation data in order to determine when simulation entities and/or objects (aircraft, land vehicles, weapons, munitions, etc.) meet predefined thresholds or objectives, herein referred to as measures. For example, C-PETS can notify the user when a friendly aircraft is within range of an enemy aircraft or a bomb was dropped too close to friendly ground units. C-PETS has a C++ Application Programming Interface (API) that allows any C-PETS user to create custom measures that the C-PETS application does not already support.

To date, AFRL has distributed C-PETS version 2.10.8 to The TTCP partners (Canada, US, United Kingdom, Australia, and New Zealand) for testing and use in nationally and internationally led simulations and experiments. With a large number of potential C-PETS users, we propose developing a repository of user generated C-PETS custom measures. Sharing the custom measures will reduce the time and cost each TTCP member spends on developing their own sets of measures by re-using and/or modifying measures that are already in the repository. In order to facilitate the sharing and use of C-PETS custom measures, all measures stored in the repository should follow the same format and have consistent and informative metadata. A common custom measures template with structured metadata will allow C-PETS users to quickly understand and implement custom measures from the repository into their own simulations.

This DRDC Reference Document briefly describes the basic process of creating a C-PETS custom measure then proposes a template with metadata for sharing C-PETS custom measures between TTCP partners. Some example C-PETS measures created using the proposed template have also been included.

2 Developing a Custom Measure in C-PETS

Creating a new custom measure for C-PETS requires a C++ development environment, e.g., Microsoft Visual Studio 2012, and the example custom measure project titled "MeasurementAuthoringExample" that comes with C-PETS.

The "MeasurementAuthoringExample" project includes 3 C++ software files that need to be edited to create a custom measure: "MeasurementAuthoringExample.cpp", "AuthoringRegistration.cpp", and "MeasurementAuthoringExample.h". These three C++ files are used to define which entities in a simulation the custom measure acts on, the name of the custom measure, and the logic to determine when a measure should be triggered or stopped.

The simulation entity or entities that you want to apply the custom measure to are defined in the "IsHandled" method of "AuthoringRegistration.cpp" using either their DIS enumeration or DIS entity ID. For example, if you wanted to apply a custom measure to all Canadian Griffon Helicopters in a simulation, the following C++ code is used for the "IsHandled" method:

```
virtual bool IsHandled(const EntityType& type, const EntityID& id) const
        {
                return (theirtype.GetKind() == 1) && (theirtype.GetDomain() == 2) &&
                        (theirtype.GetCountry() == 39) && (theirtype.GetCategory() == 21) &&
                        (theirtype.GetSubcategory() == 2);
        }
```

The custom measure's name, as used in the code, is defined in "MeasurementAuthoringExample.h", under the set of protected variables of the class "MeasurementAuthoringExample: public Entity" using the following code.

```
std::unique_ptr<Violation> custommeasurename;
```

The logic that describes how the custom measure functions is defined in "MeasurementAuthoringExample.cpp". This file has a method, called "DoInternalMeasurements()", for running custom measures that rely only on information relating specifically to C-PETS current entity. Another method, called "DoExternalMeasurements()", is used for running custom measures that require information about other entities in the simulation. For example, if a custom measure only needed the speed and altitude of an aircraft, the logic for that measure would be written in the "DoInternalMeasurements()" method. If a custom measure need the distance between the current entity and other entities in the simulation, the logic for that measure would be written in the "DoExternalMeasurements()" method. The code for the logic of a custom measures can vary widely from measure to measure, however the lines of code that inform C-PETS that a measure has been triggered or terminated are always the same. To trigger a custom measure to appear in the C-PETS Graphical User Interface (GUI) use the following code:

```
custommeasurename = std::unique_ptr<Violation>( new Violation(*this, *this, "Custom Measure is
        Occuring") );
```

To stop a custom measure from appearing in the C-PETS GUI use the following code:

```
custommeasurename->End();
custommeasurename.release();
custommeasurename.reset();
```

The following section will describe the proposed template with metadata, which has been designed to easily communicate the intent and goals of a custom measure as well as share the C++ code needed for implementing it.

3 Custom Measure Template

To simplify the sharing of custom measures, the metadata and the changes to the three project files described in the previous section are placed in a single text file (.txt) as C++ code with appropriate commenting. This allows for all the metadata and code for a custom measure to be shared with a single file, that can be used by simply copying and pasting code and comments into the appropriate C-PETS custom measure files.

The metadata for a custom measure is divided into four sets of tags. These sets of tags include i) Reference Information—bibliographic information on the creation of the measure; ii) C-PETS Information—C-PETS specific data on inputs and outputs of the measure; iii) Training Objectives—the training objectives, frameworks, crew knowledge and skills assessed with the custom measure; and iv) Required Crew and Scenario—the personnel and simulation scenario required for practical implementation of the custom measure. The complete list of metadata tags and descriptions of each tag are included in Table 1. It should be noted that not every custom measure will need to use every metadata tag. In such cases the metadata tag can be marked not applicable (NA), left blank, or removed from the custom measure text file.

Each custom measure text file should start with the metadata written as C++ commented lines with the name of each of the metadata tags.

Following the metadata should be a copy of the custom measure's IsHandled function from "AuthoringRegistration.cpp" followed by a line break and preceded with the commented header:

//In "AuthoringRegistration.cpp"

Following the IsHandled function should be the custom measure's line of code from "MeasurementAuthoringExample.h" that defines the name of the custom measure as it is used in the logic. This line should be followed with a line break and be preceded with this header:

//In "MeasurementAuthoringExample.h" define the custom measure variable as

Finally the logic that actually defines the trigger and termination of the custom measure is put into the custom measure text file. The first line should be a commented header that informs the user where the logic code should go. For example, if the logic code is for an internal measure (requires only information about the current entity) then the header should be

//In "MeasurementAuthoringExample.cpp" in "DoInternalMeasurements()"

or if the logic code is an external measure (needs information about other entities) then the header should be

//In "MeasurementAuthoringExample.cpp" in "DoExternalMeasurements()".

Following the header, the logic code should be put into the file and commented appropriately.

The title of the text file should be the primary author's name, the title of the custom measure, and the version.

Table 1: *Custom Measure Metadata.*

Metadata Tag	Description
Reference Information	
Title	C-PETS custom measure title
Version	Current version number
Author(s)	Author(s) of custom measure
Contact	Email of contact author
Organization	Author's affiliated organization
Country	Country of origin of measure authors
Date Created	Date measure was created
Date Modified	Date measure was updated
Description	Short (one or two line) description of measure
C-PETS Information	
Platform(s)	Platforms that are required for measure
Inputs	Variables and units required to perform measure calculation
Outputs	Any output produced, e.g., text file, variables
GUI Name	Name of measure as shown in C-PETS Graphical User Interface (GUI)
Training Objectives	
Performance Standard	Any formal standard or specification addressed by the measure, e.g., MOA, BTS
Performance Framework	Methodology used to create performance standard, e.g., CFITES, MEC, TCTNA
Training Objs	Training objectives associated with measure
Training Audience	Personnel whose performance is being measured

Metadata Tag	Description
Skills	Compiled actions required to complete task
Knowledge	Facts and information required to complete task
Collective Levels	Level of team or collective behaviour associated with measure
Required Crew and Scenario	
Crew	Personnel required for testing measure
Scenario	Scenario details needed for initiating measure

4 Summary

This Reference Document presents a template with metadata to facilitate the sharing of C-PETS automated performance measures. Although the template was developed primarily for the C-PETS API, the presented metadata tags can be applied to automated performance measures developed in any software environment.

DRDC has already used this template to share five automated performance measures with TTCP nations. In the near future, this template will be used by DRDC to develop measures for the RCAF's distributed simulation exercise "Exercise Virtual" in 2017 and TTCP's Joint and Coalition Training and Rehearsal and Exercise Research (JCTR) experiment in 2017.

The following sections include three examples of applying the custom measures template with metadata, two internal custom measures and one external custom measure.

4.1 Example 1: Derek McColl—Minimum Safe Altitude—v1_1.txt

```
//Reference Information ----------
//Title// Minimum Safe Altitude for American Built Aircraft
//Version// 1.1
//Authors// Derek McColl
//Contact// derek.mccoll@drdc-rddc.gc.ca
//Organization// Defence Research and Development Canada
//Country// Canada
//Date Created// Aug 2015
//Date Modified// Sept 2016
//Desciption// This C-PETS custom measure determines when an American built aircraft is flying below a pre-defined minimum altitude.

//C-PETS Information ----------
//Platform// American built aircraft
//Inputs// Aircraft MSL altitude in ft
//Outputs// C-PETS GUI Notification
//GUI Name// Below Min Safe Alt

//Training Objectives ----------
//Performance Std// Battle Task Standard
//Performance Frmwrk// NA
//Training Objs// Flying at a safe altitude, following ACOs
//Training Audience// Aircraft pilot
//Skills// NA
//Knowledge// NA
//Collective Levels// Level 1: individual

//Required Crew and Scenario ----------
//Crew// Aircraft pilot(s)
//Scenario// Mission requiring low altitude flying

//In "AuthoringRegistration.cpp"
virtual bool IsHandled(const EntityType& type, const EntityID& id) const
        {
                return (type.GetKind() == 1) && (type.GetDomain() == 2) && (type.GetCountry() == 225);
                        based on DIS enumeration
        }

//In "MeasurmentAuthoringExample.h" define the custom measure variable as
std::unique_ptr<Violation> altitudeViolation;

//In "MeasurmentAuthoringExample.cpp" in “DoInternalMeasurements()"

//get MSL altitude of current entity
double alt = this->GetPosition().GetAltMSL();

//trigger measure if MSL is less than XXX ft
if (!altitudeViolation && (alt < XXX)) {
        altitudeViolation = std::unique_ptr<Violation>( new Violation(*this, *this, "Below Min Safe Alt") );
}
//stop measure if MSL is greater than or equal to XXXft
else if (altitudeViolation && (alt >= XXX)) {
        altitudeViolation->End();
        altitudeViolation.release();
        altitudeViolation.reset();
}
```

4.2 Example 2: Derek McColl—Minimum Safe Speed and Altitude for Ch-146—v1_1.txt

```
//Reference Information ----------
//Title// Minimum Safe Speeds and Altitudes for Griffon Helicopter
//Version// 1.1
//Authors// Derek McColl
//Contact// derek.mccoll@drdc-rddc.gc.ca
//Organization// Defence Research and Development Canada
//Country// Canada
//Date Created// Aug 2015
//Date Modified// Sept 2016
//Desciption// This C-PETS custom measure determines when a Griffon helicopter is flying too low and fast.

//C-PETS Information ----------
//Platform// At least one CH-146
//Inputs// i) Aircraft altitude above ground level in feet, ii) Aircraft groundspeed in knots
//Outputs// C-PETS GUI Notification
//GUI Name// Heli Alt/Speed Vio

//Training Objectives ----------
//Performance Std// Battle Task Standard
//Performance Frmwrk// NA
//Training Objs// Flying at a safe speed at low altitude
//Training Audience// CH-146 pilots
//Skills// NA
//Knowledge// NA
//Collective Levels// LEVEL 3: Sub-sub-unit, LEVEL 1: Individual

//Required Crew and Scenario ----------
//Crew// At least one CH-146 pilot
//Scenario// Mission requiring low altitude flying

//In "AuthoringRegistration.cpp"
virtual bool IsHandled(const EntityType& type, const EntityID& id) const
        {
                return (theirtype.GetKind() == 1) && (theirtype.GetDomain() == 2) &&
                        (theirtype.GetCountry() == 39) && (theirtype.GetCategory() == 21) &&
                        (theirtype.GetSubcategory() == 2); //based on DIS enumeration
        }

//In "MeasurmentAuthoringExample.h" define the custom measure variable as
std::unique_ptr<Violation> speedaltitudeViolation;

//In "MeasurmentAuthoringExample.cpp" in “DoInternalMeasurements()"
//get helicopter speed and AGL
double alt = this->GetPosition().GetAltAGL();
double speed = this->GetPosition().GetGroundspeed();

//if speed is greater than X knots when altitude is less than Y ft set off custom measure
if (!speedaltitudeViolation && (alt < X) && (speed > Y)) {
        speedaltitudeViolation = std::unique_ptr<Violation>( new Violation(*this, *this, "Heli Alt/Speed Vio") );
}
```

```
//if speed is slower than X knots or altitude is greater than Y ft stop measure
else if (speedaltitudeViolation && ((alt >= X1 ) || (speed <= Y1))) {
        speedaltitudeViolation ->End();
        speedaltitudeViolation.release();
        speedaltitudeViolation.reset();
}
```

4.3 Example 3: Derek McColl—American Aircraft within Range of Enemy SA—v1_1.txt

```
//Reference Information ----------
//Title// American Aircraft within Range of Enemy SA
//Version// 1.1
//Authors// Derek McColl
//Contact// derek.mccoll@drdc-rddc.gc.ca
//Organization// Defence Research and Development Canada
//Country// Canada
//Date Created// Aug 2015
//Date Modified// Sept 2016
//Desciption// This C-PETS custom measure determines when an American aircraft gets within range of an //enemy surface to air (SA) weapon system.

//C-PETS Information ----------
//Platforms// At least one American aircraft and one of SA-3, SA-6, SA-8, SA-15, or SA-17
//Inputs// i) Aircraft altitude and position, ii) SA altitude and positions
//Outputs// C-PETS GUI Notification
//GUI Name// In SA Range

//Training Objectives ----------
//Performance Std// JCAS AP MOA 2004-01
//Performance Frmwrk// JTAC MECs
//Training Objs// JTAC successfully avoids/mitigates S-A threats
//Training Audience// JTACs
//Skills// Employ S-A Threat Mitigation
//Knowledge// Surface-to-Air Threats, Surface-to-Air Threat Mitigation Tactics
//Collective Levels// LEVEL 1: Individual

//Required Crew and Scenario ----------
//Crew// JTAC and live, virtual or constructive aircraft pilots
//Scenario// Mission with supporting LVC aircraft in or near enemy territory


//In "AuthoringRegistration.cpp"
virtual bool IsHandled(const EntityType& type, const EntityID& id) const
        {
                return (type.GetKind() == 1) && (type.GetDomain() == 2) && (type.GetCountry() == 225);
                        based on DIS enumeration
        }

//In "MeasurmentAuthoringExample.h" define the custom measure variable as
//std::unique_ptr<Violation> saTooClose;

//In "MeasurmentAuthoringExample.cpp" in “DoExternalMeasurements()"
// make list of red ground entities
std::vector<Entity> redGNDEntites = petsDatabase->
        GetObjects(PETSDatabase::Types::OBJECT_ENTITY_RED, Entity::State::RUNNING);
```

```
//flag for measure start/end
bool sacloseflag = false;
//loop through red entities to find SAs
for (Entity redgnd : redGNDEntites)
{
	//get DIS type to isolate SA
	EntityType theirtype = redgnd.GetDISType();

	//if an SA - 3
	if ((theirtype.GetKind() == 1) && (theirtype.GetDomain() == 1) && (theirtype.GetCountry() == 222) &&
	(theirtype.GetCategory() == 28) && (theirtype.GetSubcategory() == 2) && (theirtype.GetSpecific() == 3))
	{
		//get their position
		Position theirPos = redgnd.GetPosition();
		//get distance to aircraft
		double range = GetPosition().GetRange2(theirPos);
		//if any aircraft is too close flag a measure, if none are too close than flag will not be tripped
		if ((range < X1) && (range>1) && (this->GetPosition().GetAltMSL() < Y1)) {
			sacloseflag = true;
		}
	}
	//if an SA - 6
	else if ((theirtype.GetKind() == 1) && (theirtype.GetDomain() == 1) && (theirtype.GetCountry() == 222)
	&& (theirtype.GetCategory() == 28) && (theirtype.GetSubcategory() == 6) && (theirtype.GetSpecific() ==
	2))
	{
		//get their position
		Position theirPos = redgnd.GetPosition();
		//get distance to aircraft
		double range = GetPosition().GetRange2(theirPos);
		//if any aircraft is too close flag a measure, if none are too close than flag will not be tripped
		if ((range < X2) && (range>1) && (this->GetPosition().GetAltMSL() < Y2)) {
			sacloseflag = true;
		}
	}
	//if an SA - 8
	else if ((theirtype.GetKind() == 1) && (theirtype.GetDomain() == 1) && (theirtype.GetCountry() == 222)
	&& (theirtype.GetCategory() == 28) && (theirtype.GetSubcategory() == 7) && (theirtype.GetSpecific() ==
	2))
	{
		//get their position
		Position theirPos = redgnd.GetPosition();
		//get distance to aircraft
		double range = GetPosition().GetRange2(theirPos);
		//if any aircraft is too close flag a measure, if none are too close than flag will not be tripped
		if ((range < X3) && (range>1) && (this->GetPosition().GetAltMSL() < Y3)) {
			sacloseflag = true;
		}
	}
```

```
//if an SA-15
	else if ((theirtype.GetKind() == 1) && (theirtype.GetDomain() == 1) && (theirtype.GetCountry() == 222)
	&& (theirtype.GetCategory() == 28) && (theirtype.GetSubcategory() == 10)) // && (theirtype.GetSpecific()
	== 1))
	{
		//get their position
		Position theirPos = redgnd.GetPosition();
		//get distance to aircraft
		double range = GetPosition().GetRange2(theirPos);
		//if any aircraft is too close flag a measure, if none are too close than flag will not be tripped
		if ((range < X4) && (range>1) && (this->GetPosition().GetAltMSL() < Y4)) {
			sacloseflag = true;
		}
	}

//if an SA-17
	else if ((theirtype.GetKind() == 1) && (theirtype.GetDomain() == 1) && (theirtype.GetCountry() == 222)
	&& (theirtype.GetCategory() == 28) && (theirtype.GetSubcategory() == 11)) // && (theirtype.GetSpecific()
	== 2))
	{
		//get their position
		Position theirPos = redgnd.GetPosition();
		//get distance to aircraft
		double range = GetPosition().GetRange2(theirPos);
		//if any aircraft is too close flag a measure, if none are too close than flag will not be tripped
		if ((range < X5) && (range>1) && (this->GetPosition().GetAltMSL() < Y5)) {
			sacloseflag = true;
		}
	}
}
//do measure logic, since this measure is dependent only on the flag,
if (!saTooClose && (sacloseflag == true)) {
	saTooClose = std::unique_ptr<Violation>( new Violation(*this, *this, "In SA Range") );
	}
else if (saTooClose && (sacloseflag == false)) {
	saTooClose->End();
	saTooClose.release();
	saTooClose.reset();
}
```

This page intentionally left blank.

References

[1] Warfighter Readiness Science and Technology Program, Air Force Research Laboratory (2016) Coalition Performance Evaluation Tracking System version 2.10.8 [Computer Software]. Wright-Patterson AFB, OH.

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g., Centre sponsoring a contractor's report, or tasking agency, are entered in Section 8.)

DRDC – Toronto Research Centre
Defence Research and Development Canada
1133 Sheppard Avenue West
P.O. Box 2000
Toronto, Ontario M3M 3B9
Canada

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC DECEMBER 2013

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Automated performance measure template with metadata

4. AUTHORS (last name, followed by initials – ranks, titles, etc., not to be used)

McColl, D.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
February 2017	22	1

7. DESCRIPTIVE NOTES (The category of the document, e.g., technical report, technical note or memorandum. If appropriate, enter the type of report, e.g., interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Reference Document

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

DRDC – Toronto Research Centre
Defence Research and Development Canada
1133 Sheppard Avenue West
P.O. Box 2000
Toronto, Ontario M3M 3B9
Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC-RDDC-2017-D010	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Defence Research and Development Canada is developing automated performance measures to support the Royal Canadian Air Force Simulation Strategy. These automated performance measures will assist instructors in making accurate assessments of trainee performance during simulations and exercises. Partnering nations in The Technical Cooperation Panel are also developing automated performance measures. To facilitate the sharing of automated performance measures between these nations, this Reference Document presents a template with metadata that will ensure automated performance measures are easy to understand and consistently developed. The template consists of two main components: the metadata and the programming code. The metadata includes information on the automated performance measure's reference data (e.g., title, author, date created, etc.), required software information (inputs, outputs), training objective details (performance standards and frameworks, skills, knowledge, etc.), and required operators and testing scenarios (crew and mission types). This template has been applied to five automated performance measures developed in the Coalition Performance Evaluation Tracking System's C++ application programming interface.

Recherche et développement pour la défense Canada élabore actuellement des mesures automatisées du rendement en vue d'appuyer la Stratégie de simulation de l'Aviation royale canadienne. Ces mesures automatisées permettront aux instructeurs de produire une évaluation précise du rendement des stagiaires au cours de simulations et d'exercices. Les pays partenaires membres du Programme de coopération technique s'affairent également à mettre au point de telles mesures automatisées du rendement. Pour faciliter le partage de ces mesures entre pays membres, le présent document de référence décrit un modèle avec métadonnées pour faciliter leur compréhension et uniformiser leur élaboration. Le modèle comporte deux éléments principaux, soit les métadonnées et le code de programmation. Les métadonnées incluent l'information sur les données de référence de la mesure automatisée du rendement (son nom, son auteur, sa date de création, etc.), les renseignements nécessaires sur les logiciels (entrées, sorties) et les détails sur les objectifs de la formation (normes et cadres de rendement, habiletés, connaissances, etc.), ainsi que les opérateurs et les scénarios d'essai requis (types d'équipage et de mission). Le modèle a été appliqué à cinq mesures automatisées du rendement élaborées dans l'interface de programmation d'applications C++ du Système de suivi des évaluations du rendement de la Coalition.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g., Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

metadata; performance measures; template