IBM SPSS Modeler 18.2.2 Solution Publisher

Note

Before you use this information and the product it supports, read the information in “Notices” on page 33.

Product Information

This edition applies to version 18, release 2, modification 2 of IBM® SPSS® Modeler and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation .
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Preface

IBM SPSS Modeler is the IBM Corp. enterprise-strength data mining workbench. SPSS Modeler helps organizations to improve customer and citizen relationships through an in-depth understanding of data. Organizations use the insight gained from SPSS Modeler to retain profitable customers, identify cross-selling opportunities, attract new customers, detect fraud, reduce risk, and improve government service delivery.

SPSS Modeler's visual interface invites users to apply their specific business expertise, which leads to more powerful predictive models and shortens time-to-solution. SPSS Modeler offers many modeling techniques, such as prediction, classification, segmentation, and association detection algorithms. Once models are created, IBM SPSS Modeler Solution Publisher enables their delivery enterprise-wide to decision makers or to a database.

About IBM Business Analytics

IBM Business Analytics software delivers complete, consistent and accurate information that decision-makers trust to improve business performance. A comprehensive portfolio of business intelligence, predictive analytics, financial performance and strategy management, and analytic applications provides clear, immediate and actionable insights into current performance and the ability to predict future outcomes. Combined with rich industry solutions, proven practices and professional services, organizations of every size can drive the highest productivity, confidently automate decisions and deliver better results.

As part of this portfolio, IBM SPSS Predictive Analytics software helps organizations predict future events and proactively act upon that insight to drive better business outcomes. Commercial, government and academic customers worldwide rely on IBM SPSS technology as a competitive advantage in attracting, retaining and growing customers, while reducing fraud and mitigating risk. By incorporationg IBM SPSS software into their daily operations, organizations become predictive enterprises - able to direct and automate decisions to meet business goals and achieve measurable competitive advantage. For further information or to reach a representative visit https://www.ibm.com/mysupport/s/.

Technical support

Technical support is available to maintenance customers. Customers may contact Technical Support for assistance in using IBM Corp. products or for installation help for one of the supported hardware environments. To reach Technical Support, see the IBM Corp. website at https://www.ibm.com/mysupport/s/. Be prepared to identify yourself, your organization, and your support agreement when requesting assistance.

Chapter 1. About IBM SPSS Modeler

IBM SPSS Modeler is a set of data mining tools that enable you to quickly develop predictive models using business expertise and deploy them into business operations to improve decision making. Designed around the industry-standard CRISP-DM model, IBM SPSS Modeler supports the entire data mining process, from data to better business results.

IBM SPSS Modeler offers a variety of modeling methods taken from machine learning, artificial intelligence, and statistics. The methods available on the Modeling palette allow you to derive new information from your data and to develop predictive models. Each method has certain strengths and is best suited for particular types of problems.

SPSS Modeler can be purchased as a standalone product, or used as a client in combination with SPSS Modeler Server. A number of additional options are also available, as summarized in the following sections. For more information, see https://www.ibm.com/analytics/us/en/technology/spss/.

IBM SPSS Modeler Products

The IBM SPSS Modeler family of products and associated software comprises the following.

- IBM SPSS Modeler
- IBM SPSS Modeler Server
- IBM SPSS Modeler Administration Console (included with IBM SPSS Deployment Manager)
- IBM SPSS Modeler Batch
- IBM SPSS Modeler Solution Publisher
- IBM SPSS Modeler Server adapters for IBM SPSS Collaboration and Deployment Services

IBM SPSS Modeler

SPSS Modeler is a functionally complete version of the product that you install and run on your personal computer. You can run SPSS Modeler in local mode as a standalone product, or use it in distributed mode along with IBM SPSS Modeler Server for improved performance on large data sets.

With SPSS Modeler, you can build accurate predictive models quickly and intuitively, without programming. Using the unique visual interface, you can easily visualize the data mining process. With the support of the advanced analytics embedded in the product, you can discover previously hidden patterns and trends in your data. You can model outcomes and understand the factors that influence them, enabling you to take advantage of business opportunities and mitigate risks.

SPSS Modeler is available in two editions: SPSS Modeler Professional and SPSS Modeler Premium. See the topic “ IBM SPSS Modeler Editions” on page 2 for more information.

IBM SPSS Modeler Server

SPSS Modeler uses a client/server architecture to distribute requests for resource-intensive operations to powerful server software, resulting in faster performance on larger data sets.

SPSS Modeler Server is a separately-licensed product that runs continually in distributed analysis mode on a server host in conjunction with one or more IBM SPSS Modeler installations. In this way, SPSS Modeler Server provides superior performance on large data sets because memory-intensive operations can be done on the server without downloading data to the client computer. IBM SPSS Modeler Server also provides support for SQL optimization and in-database modeling capabilities, delivering further benefits in performance and automation.

IBM SPSS Modeler Administration Console

The Modeler Administration Console is a graphical user interface for managing many of the SPSS Modeler Server configuration options, which are also configurable by means of an options file. The console is included in IBM SPSS Deployment Manager, can be used to monitor and configure your SPSS Modeler Server installations, and is available free-of-charge to current SPSS Modeler Server customers. The application can be installed only on Windows computers; however, it can administer a server installed on any supported platform.

IBM SPSS Modeler Batch

While data mining is usually an interactive process, it is also possible to run SPSS Modeler from a command line, without the need for the graphical user interface. For example, you might have long-running or repetitive tasks that you want to perform with no user intervention. SPSS Modeler Batch is a special version of the product that provides support for the complete analytical capabilities of SPSS Modeler without access to the regular user interface. SPSS Modeler Server is required to use SPSS Modeler Batch.

IBM SPSS Modeler Solution Publisher

SPSS Modeler Solution Publisher is a tool that enables you to create a packaged version of an SPSS Modeler stream that can be run by an external runtime engine or embedded in an external application. In this way, you can publish and deploy complete SPSS Modeler streams for use in environments that do not have SPSS Modeler installed. SPSS Modeler Solution Publisher is distributed as part of the IBM SPSS Collaboration and Deployment Services - Scoring service, for which a separate license is required. With this license, you receive SPSS Modeler Solution Publisher Runtime, which enables you to execute the published streams.

For more information about SPSS Modeler Solution Publisher, see the IBM SPSS Collaboration and Deployment Services documentation. The IBM SPSS Collaboration and Deployment Services Knowledge Center contains sections called "IBM SPSS Modeler Solution Publisher" and "IBM SPSS Analytics Toolkit."

IBM SPSS Modeler Server Adapters for IBM SPSS Collaboration and Deployment Services

A number of adapters for IBM SPSS Collaboration and Deployment Services are available that enable SPSS Modeler and SPSS Modeler Server to interact with an IBM SPSS Collaboration and Deployment Services repository. In this way, an SPSS Modeler stream deployed to the repository can be shared by multiple users, or accessed from the thin-client application IBM SPSS Modeler Advantage. You install the adapter on the system that hosts the repository.

IBM SPSS Modeler Editions

SPSS Modeler is available in the following editions.

SPSS Modeler Professional

SPSS Modeler Professional provides all the tools you need to work with most types of structured data, such as behaviors and interactions tracked in CRM systems, demographics, purchasing behavior and sales data.

SPSS Modeler Premium

SPSS Modeler Premium is a separately-licensed product that extends SPSS Modeler Professional to work with specialized data and with unstructured text data. SPSS Modeler Premium includes IBM SPSS Modeler Text Analytics:

IBM SPSS Modeler Text Analytics uses advanced linguistic technologies and Natural Language Processing (NLP) to rapidly process a large variety of unstructured text data, extract and organize the key concepts, and group these concepts into categories. Extracted concepts and categories can be combined

with existing structured data, such as demographics, and applied to modeling using the full suite of IBM SPSS Modeler data mining tools to yield better and more focused decisions.

IBM SPSS Modeler Subscription

IBM SPSS Modeler Subscription provides all the same predictive analytics capabilities as the traditional IBM SPSS Modeler client. With the Subscription edition, you can download product updates regularly.

Documentation

Documentation is available from the Help menu in SPSS Modeler. This opens the online Knowledge Center, which is always available outside the product.

Complete documentation for each product (including installation instructions) is also available in PDF format, in a separate compressed folder, as part of the product download. Or the latest PDF documents can be downloaded from the web at https://www.ibm.com/support/pages/spss-modeler-1822-documentation.

SPSS Modeler Professional Documentation

The SPSS Modeler Professional documentation suite (excluding installation instructions) is as follows.

- **IBM SPSS Modeler User's Guide.** General introduction to using SPSS Modeler, including how to build data streams, handle missing values, build CLEM expressions, work with projects and reports, and package streams for deployment to IBM SPSS Collaboration and Deployment Services or IBM SPSS Modeler Advantage.
- **IBM SPSS Modeler Source, Process, and Output Nodes.** Descriptions of all the nodes used to read, process, and output data in different formats. Effectively this means all nodes other than modeling nodes.
- **IBM SPSS Modeler Modeling Nodes.** Descriptions of all the nodes used to create data mining models. IBM SPSS Modeler offers a variety of modeling methods taken from machine learning, artificial intelligence, and statistics.
- **IBM SPSS Modeler Applications Guide.** The examples in this guide provide brief, targeted introductions to specific modeling methods and techniques. An online version of this guide is also available from the Help menu. See the topic "Application examples" on page 4 for more information.
- **IBM SPSS Modeler Python Scripting and Automation.** Information on automating the system through Python scripting, including the properties that can be used to manipulate nodes and streams.
- **IBM SPSS Modeler Deployment Guide.** Information on running IBM SPSS Modeler streams as steps in processing jobs under IBM SPSS Deployment Manager.
- **IBM SPSS Modeler CLEF Developer's Guide.** CLEF provides the ability to integrate third-party programs such as data processing routines or modeling algorithms as nodes in IBM SPSS Modeler.
- **IBM SPSS Modeler In-Database Mining Guide.** Information on how to use the power of your database to improve performance and extend the range of analytical capabilities through third-party algorithms.
- **IBM SPSS Modeler Server Administration and Performance Guide.** Information on how to configure and administer IBM SPSS Modeler Server.
- **IBM SPSS Deployment Manager User Guide.** Information on using the administration console user interface included in the Deployment Manager application for monitoring and configuring IBM SPSS Modeler Server.
- **IBM SPSS Modeler CRISP-DM Guide.** Step-by-step guide to using the CRISP-DM methodology for data mining with SPSS Modeler.
- **IBM SPSS Modeler Batch User's Guide.** Complete guide to using IBM SPSS Modeler in batch mode, including details of batch mode execution and command-line arguments. This guide is available in PDF format only.

SPSS Modeler Premium Documentation

The SPSS Modeler Premium documentation suite (excluding installation instructions) is as follows.

- **SPSS Modeler Text Analytics User's Guide.** Information on using text analytics with SPSS Modeler, covering the text mining nodes, interactive workbench, templates, and other resources.

Application examples

While the data mining tools in SPSS Modeler can help solve a wide variety of business and organizational problems, the application examples provide brief, targeted introductions to specific modeling methods and techniques. The data sets used here are much smaller than the enormous data stores managed by some data miners, but the concepts and methods that are involved are scalable to real-world applications.

To access the examples, click **Application Examples** on the Help menu in SPSS Modeler.

The data files and sample streams are installed in the `Demos` folder under the product installation directory. For more information, see "Demos Folder" on page 4.

Database modeling examples. See the examples in the *IBM SPSS Modeler In-Database Mining Guide*.

Scripting examples. See the examples in the *IBM SPSS Modeler Scripting and Automation Guide*.

Demos Folder

The data files and sample streams that are used with the application examples are installed in the `Demos` folder under the product installation directory (for example: `C:\Program Files\IBM\SPSS\Modeler\<version>\Demos`). This folder can also be accessed from the IBM SPSS Modeler program group on the Windows Start menu, or by clicking `Demos` on the list of recent directories in the **File** > **Open Stream** dialog box.

License tracking

When you use SPSS Modeler, license usage is tracked and logged at regular intervals. The license metrics that are logged are *AUTHORIZED_USER* and *CONCURRENT_USER*, and the type of metric that is logged depends on the type of license that you have for SPSS Modeler.

The log files that are produced can be processed by the IBM License Metric Tool, from which you can generate license usage reports.

The license log files are created in the same directory where SPSS Modeler Client log files are recorded (by default, `%ALLUSERSPROFILE%/IBM/SPSS/Modeler/<version>/log`).

Chapter 2. IBM SPSS Modeler Solution Publisher

IBM SPSS Modeler Solution Publisher is a powerful tool for integrating your data mining results into your business process to solve real-world problems. Using IBM SPSS Modeler Solution Publisher, you can create a "packaged" version of a stream that can be executed by an external Runtime engine or embedded in an external application. This enables you to deploy your data modeling streams in a production environment to support your everyday business processes and to empower your organization's decision makers with the knowledge gained from mining your data.

Using IBM SPSS Modeler Solution Publisher offers more power than simply exporting the model (as PMML), because it enables you to publish and deploy *complete IBM SPSS Modeler streams*. That means you can perform data preparation as well as record and field operations, such as aggregating data, selecting records, or deriving new fields, before creating predictions based on a model. You can then further process the model results before saving the data--all simply by executing the published stream.

Note: Publishing is achieved by using the Publish tab of the appropriate export node. See the topic "Publishing streams" on page 6 for more information. Streams saved in releases prior to version 11.0 of the product used a separate Publisher node. The node is still supported for backwards compatibility, but is no longer available on the node palette.

Licensing IBM SPSS Modeler Solution Publisher

Note: IBM SPSS Modeler Solution Publisher is distributed as part of the IBM SPSS Collaboration and Deployment Services Scoring Service, for which a separate license is required. http://www.ibm.com/software/analytics/spss/products/deployment/cds/

When you purchase a license, a separate IBM SPSS Modeler Solution Publisher Runtime is provided that enables you to execute published streams. See the topic "Installing the Runtime" on page 13 for more information.

Logging

The log file is controlled by the `log4cxx.properties` configuration file, located in the `config` folder of your IBM SPSS Modeler Solution Publisher installation directory. You can modify the `log4cxx.properties` configuration file to customize the logging output.

Note that if using Linux/UNIX, in order for the application to find the configuration file, you must define an environment variable before running the application (for example, `export MODELERRUNTIME=MSP_INSTALLATION_DIRECTORY`).

How IBM SPSS Modeler Solution Publisher Works

Deploying a solution using IBM SPSS Modeler Solution Publisher involves two phases: *publishing* a stream and *executing* a stream.

Publishing. As you work through the data mining process, you will eventually arrive at a model that provides a good solution to your business problem. At that point, you are ready to take that model and apply it to your business process. When you publish a stream, a detailed description of the stream is written to the disk (as an *image* file and a *parameter* file). See the topic "Publishing streams" on page 6 for more information.

Executing. After you've published the stream, you can re-create the process implemented in the stream by executing the published stream. This is done either by using the standalone IBM SPSS Modeler Runtime (*modelerrun.exe*) or by developing an application that uses the IBM SPSS Modeler Runtime Library to execute the stream. To execute streams outside of IBM SPSS Modeler (using the Runtime or a custom application), you must first install the IBM SPSS Modeler Solution Publisher Runtime. See the topic "Installing the Runtime" on page 13 for more information.

Publishing streams

Publishing streams is done directly from IBM SPSS Modeler using any of the standard export nodes: Database, Flat File, Statistics Export, Extension Export, Data Collection Export, SAS Export, Excel, and XML Export nodes. The type of export node determines the format of the results to be written each time the published stream is executed using the IBM SPSS Modeler Solution Publisher Runtime or external application. For example, if you want to write your results to a database each time the published stream is run, use a Database export node.

To publish a stream

1. Open or build a stream in the normal manner and attach an export node at the end.
2. On the Publish tab in the export node, specify a root name for the published files (that is, the file name to which the `.pim`, `.par`, and `.xml` extensions will be appended).
3. Click **Publish** to publish the stream, or select **Publish the stream** to automatically publish the stream each time the node is executed.

Published name. Specify the root name for the published image and parameter files.

- The **image file** (`*.pim`) provides all of the information needed for the Runtime to execute the published stream exactly as it was at the time of export. If you are confident that you will not need to change any of the settings for the stream (such as the input data source or the output data file), you can deploy the image file only.
- The **parameter file** (`*.par`) contains configurable information about data sources, output files, and execution options. If you want to be able to control the input or output of the stream without republishing the stream, you will need the parameter file as well as the image file.
- The **metadata file** (`*.xml`) describes the inputs and outputs of the image and their data models. It is designed for use by applications which embed the runtime library and which need to know the structure of the input and output data.

  Note: This file is only produced if you select the **Publish metadata** option.

Publish parameters. If required, you can include stream parameters in the `*.par` file. You can change these stream parameter values when you execute the image either by editing the `*.par` file or through the runtime API.

This option enables the **Parameters** button. The Publish Parameters dialog box is displayed when you click the button.

Choose the parameters you want to include in the published image by selecting the relevant option in the **Publish** column.

On stream execution. Specifies whether the stream is automatically published when the node is executed.

- **Export data.** Executes the export node in the standard manner, without publishing the stream. (Basically, the node executes in IBM SPSS Modeler the same way it would if IBM SPSS Modeler Solution Publisher were not available.) If you select this option, the stream will not be published unless you do so explicitly by clicking **Publish** in the export node dialog box. Alternatively, you can publish the current stream using the Publish tool on the toolbar or by using a script.
- **Publish the stream.** Publishes the stream for deployment using IBM SPSS Modeler Solution Publisher. Select this option if you want to automatically publish the stream every time it is executed.

Note:

- If you plan to run the published stream with new or updated data, it is important to note that the order of fields in the input file must be the same as the order of fields in the source node input file specified in the published stream.
- When publishing to external applications, consider filtering extraneous fields or renaming fields to conform with input requirements. Both can be accomplished using a Filter node prior to the export node.

Chapter 3. Running published streams

The main effect of running a published stream is always to generate a set of data. The data may be stored to a disk file or written to a database. Published streams can't generate graphs, tables, generated models, or other non-data output.

Streams published using IBM SPSS Modeler Solution Publisher can run using the IBM SPSS Modeler Solution Publisher Runtime program. The Runtime program, `modelerrun.exe`, is started from the command line, with options indicating the image file and (optional) parameter file to run. The Runtime command is as follows:

```
modelerrun -nobanner -p <parameter-file> -o <options> <image-file>
```

where the command entries are shown in the following table.

Table 1. Runtime command entries

Command Entry	Description
`-nobanner`	suppresses the startup text message for the Runtime (optional).
`<parameter-file>`	is the filename of the parameter file (optional).
`<options>`	is a comma-separated list of `option=value` pairs specifying run options for the Runtime. Options are detailed below (optional). **Note:** You must not have any empty spaces after a comma.
`<image-file>`	is the filename of the image file to execute (required).

The available run options are shown in the following table.

Table 2. Valid Runtime options

Name	Default Value	Description
`max_file_size`	–1	Maximum size of saved files. A value of –1 indicates no limit.
`max_sql_string_length`	2,048	Maximum length of a string imported from the database with SQL. String values longer than this are truncated on the right without warning. The valid range is between 1 and 65,535 characters.
`memory_usage`	100	Multiplier for dynamic memory allocation. Adjust this value up or down to regulate the server's total memory usage.
`temp_directory`	"	Specifies the directory to be used for temporary files.
`request_passwords`	Y	Controls whether you are prompted for a database password when running a published stream that requires database access (Y/N).

Note: When you publish a stream that contains a user input node, note that SPSS Modeler Solution Publisher requires an input file for the node (whereas in SPSS Modeler you define the field and its data directly in the node). The default file name is `userIput.dat`, and you can change it in the parameter file.

Note: When running a stream containing a CPLEX Optimization node in **SPSS Modeler Solution Publisher**, by default the embedded Community edition CPLEX library is used. It has a limitation of 1000 variables and 1000 constraints. If you install the full edition of IBM ILOG CPLEX and want to use the full edition's CPLEX engine instead, which doesn't have such limitations, complete the following step for your platform.

- On Windows, add the OPL library path as a command line argument for `modelerrun.exe`. For example:

```
-o cplex_opl_lib_path="<CPLEX_path>\opl\bin\<Platform_dir>"
```

  Where <CPLEX_path> is the CPLEX installation directory such as `C:\Program Files\IBM\ILOG\CPLEX_Studio127`, and <Platform_dir> is the platform-specific directory such as `x64_win64`.
- On Linux, edit `modelerrun` and add the OPL library path. For example:

```
CPLEX_OPL_LIB_PATH=<CPLEX_path>/opl/bin/<Platform_dir>
```

  Where <CPLEX_path> is the CPLEX installation directory such as `/root/Libs_127_FullEdition/Linux_x86_64`, and <Platform_dir> is the platform-specific directory such as `x86-64_linux`.

Running a local Apache Spark stream in IBM SPSS Modeler Solution Publisher

When you run a local Spark stream in IBM SPSS Modeler Solution Publisher, you must specify the following in the `modelerrun` command (where `C:/Anaconda3/python.exe` is the path to your Python executable). Anaconda 3 is required (Anaconda 2 isn't supported).

```
-o eas_pyspark_python_path=C:/Anaconda3/python.exe
```

Using the Parameter File to Customize Stream Execution

When a stream is published, the image file contains information about the data source and output data destination that were selected in the original stream. However, it is often convenient to run a stream against data from a different data source or to reroute the output data to a different destination. This can be done by editing the stream's parameter file.

This is particularly important when you will be executing the published stream on a computer different from the one on which it was created. You will need to update the locations of the input and output files to reflect the structure of the target computer. Note that published streams do not support relative paths, making it especially important to verify the data locations in your parameter file. (If you specify relative paths in the Export node options when publishing the stream, IBM SPSS Modeler will convert them to absolute paths before creating the image and parameter files for the published stream.)

You can also use the parameter file to set execution options and to change the values of stream parameters.

Parameter values are specified in the parameter file as <name.attribute>=<value> pairs. For some parameters, the parameter name contains an ID digit to distinguish nodes of the same type in the stream. For example, in a stream where both input and output refer to a database connection, the parameters describing the input connection might be `dbconn0.datasource`, `dbconn0.user`, etc., and the parameters associated with the output connection might be `dbconn1.datasource`, `dbconn1.user`, etc. Values are always enclosed in double quotes. The parameter file contains the following parameters, all of which can be changed to customize stream execution.

Table 3. Execution parameters

Parameter	Purpose
`dbconn`*x*`.datasource`	Data source name (DSN).

Table 3. Execution parameters (continued)	
Parameter	**Purpose**
dbconn*x*.user	User name for restricted access databases.
dbconn*x*.password	Password for restricted access databases.
dbconn*x*.epassword	Encoded password for restricted access databases. To generate an encoded password, select **Encode Password** from the Tools menu of the IBM SPSS Modeler user interface. Copy and paste the encoded password as needed.
dbconn*x*.password_required	Flag indicating whether to prompt for a password for this database connection. (The value is "Y" or "N".) Automatically set to "N" when there is no password specified at publish time.
dbconn*x*.catalog	Catalog associated with the database connection.
dbtable*x*.name	Base table name. *Note*: The order of fields in a new table must be the same as the order of fields in the originally published stream.
dbtable*x*.schema	Schema name.
dbtable*x*.catalog	Catalog associated with the current table.
file*x*.name	Filename for data file. *Note*: The order of fields in a new input file must be the same as the order of fields in the originally published stream.
file*x*.path	File location (directory name).
file*x*.field_names_included	For text format files, the flag indicating whether the field names are included as the first line of the file. (The value is "Y" or "N".)
file*x*.field_separator	For text format files, the character(s) used to separate field values.
file*x*.decimal_separator	For text format files, the character used for the decimal point.
options.angle_in_radians	Flag indicating whether radians are used as the unit of measurement in trigonometric CLEM expressions. (The value is "Y" or "N".)
options.date_2digit_baseline	Defines the century for dates specified with two-digit years.
options.date_baseline	The baseline year (always January 1) used by CLEM date functions that work with a single date.
options.time_rollover	Flag indicating whether negative time differences refer to the past.
options.decimal_separator	The default decimal point character in text files.
options.time_format	Time format used when strings are interpreted as times by CLEM time functions.
options.date_format	Date format used when strings are interpreted as dates by CLEM date functions.
options.timestamp_format	Format used when reading timestamp fields as strings from ODBC data sources.

Table 3. Execution parameters (continued)	
Parameter	**Purpose**
`paramx.value`	Stream parameter value. *Note*: The parameter name, description, and storage type are included for reference but cannot be changed. The value must be valid for the type.

Embedding IBM SPSS Modeler Solution Publisher into Applications

In addition to the standalone Runtime engine, IBM SPSS Modeler Solution Publisher provides a Runtime programming library (CLEMRTL) that allows other programs to control IBM SPSS Modeler Solution Publisher stream execution. You can call CLEMRTL procedures in client programs written in C and C++. To use the CLEMRTL, you need to include the header file *clemrtl.h* (available from the *..\installation\clemrtl\include* folder) and be sure to link the appropriate library file for your development platform into your application when you build it. The clemrtl.dll file is available from the *..\installation\bin* folder.

Any source file that references library procedures must include the header *clemrtl.h*. This header file provides ANSI C prototypes for the library procedures and defines useful macros. It does not require any other headers to be included beyond what your program requires. To protect against name clashes, all library type and function names start with `clemrtl_` and all macro names are prefixed with `CLEMRTL_`.

CLEMRTL produces reports containing useful information that should be communicated back to the application in some way. The CLEMRTL provides mechanisms for dealing with such messages:

- The application can retrieve details of the last error using the `clemrtl_getErrorDetail()` function.
- The application can provide its own report-handling procedure using the `clemrtl_setReportHandler()` function.

The Runtime system has a localized message catalog for reports. Any report passed to the application would include the localized message string. Reports also include the report code so that the application could choose to interpret and present the message differently. In order to get detailed (localized) messages, the application needs to ship the appropriate *messages.cfg* file in a *config* directory. Different versions of *messages.cfg* for different locales can be found in the directory *<installdir>/config/<locale>*, where *<installdir>* is the directory in which you installed IBM SPSS Modeler Solution Publisher, and *<locale>* is the desired locale. Select the appropriate version of *messages.cfg* and add it to a *config* subdirectory in your project directory.

The Runtime system needs no special environment or registry settings to operate correctly. Dependent libraries have to be distributed with an application linked to the CLEMRTL. These files are included in the *<installdir>/bin* directory.

Requirements

The following table shows the platforms and compilers tested with the Runtime programming library.

Table 4. Platforms and compilers	
Platform	**Suggested compiler(s)**
Linux 64-bit	gcc 3.2.3, Intel C++ compiler 9.1.043
Windows	Visual Studio 2008 R2, Visual Studio 2015

IBM SPSS Modeler Solution Publisher Sample Application

An example application, *dlltest.c*, is included in the *\clemtrl\demo* folder in the IBM SPSS Modeler Solution Publisher installation directory. Use the example application to help you get started embedding the Runtime library into your own applications.

The information used by the example application is included in the *\clemtrl\demo* folder; see the *README.txt* file for instructions. Compiling an application varies by platform, so example makefiles are also included.

Chapter 4. Installing IBM SPSS Modeler Solution Publisher Runtime

Installing the Runtime

Installing the IBM SPSS Modeler Solution Publisher Runtime allows you to execute published streams. Published streams are independent of the operating system, so you can execute any published stream on any platform supported by the Runtime.

The functionality that allows you to publish streams from IBM SPSS Modeler is installed with IBM SPSS Modeler. Note that published files must be re-exported with each release of IBM SPSS Modeler. For example, a file exported using an earlier version will not work with the current version of IBM SPSS Modeler Solution Publisher Runtime.

Installing on Windows

The following section describes the procedure for installing the SPSS Modeler Solution Publisher Runtime on the Windows operating system.

1. Launch the IBM SPSS Modeler Solution Publisher installation.
2. In the installation wizard, click **Next** to begin.
3. Follow the instructions that appear on the screen. To continue, click **Next**.
4. Once you have specified all options, you are ready to install. Click **Install** to begin transferring files.
5. After all files have been installed, click **Finish**.

Installing on UNIX

The following sections describe additional system requirements and the procedure for installing the IBM SPSS Modeler Solution Publisher Runtime on the UNIX operating system.

Additional Requirements for UNIX

You must ensure that kernel limits on the system are sufficient for the operation of IBM SPSS Modeler Server. The data, memory, file, and processes ulimits are particularly important and should be set to unlimited within the IBM SPSS Modeler Server environment. To do this:

1. Add the following commands to `modelersrv.sh`:

```
ulimit –d unlimited
```

```
ulimit –m unlimited
```

```
ulimit –f unlimited
```

```
ulimit –u unlimited
```

   In addition, set the stack limit to the maximum allowed by your system (`ulimit -s XXXX`), for example:

```
ulimit -s 64000
```

2. Restart IBM SPSS Modeler Server.

You also need the *gzip* file compression utility and *GNU cpio* installed and on the PATH in order for the installer to be able to decompress the installation files. In addition, on the machine running SPSS Modeler Server, you should set the locale to EN_US.UTF-8.

UNIX Installation Procedure

These installation instructions apply to the IBM SPSS Modeler Solution Publisher Runtime for UNIX. You need read and write permissions to the target installation directory, so log on with an account that has sufficient permissions.

Note: You must be logged in as *root* to install.

1. From the installation package, open the *modelrun* directory.
2. Change to the relevant platform directory.
3. Run the *.bin* install script. For example:

```
./modelersolutionpublisherhpia64.bin -i console
```

4. Introduction details are displayed. Press Enter to continue.
5. Licensing information is displayed. Read the license, type 1 to accept it, and press Enter to continue.
6. A list of available languages is displayed. Enter the number for the language version that you want to install and press Enter to continue.
7. You are prompted to type the installation location. To use the default directory, */usr/IBM/SPSS/ModelerSolutionPublisher<nn>* (where *<nn>* is the version number), press Enter. The Runtime will be installed in the specified directory.
8. You are prompted to confirm the installation location. When it is correct, type y and press Enter.
9. A pre-installation summary is displayed to confirm your entries so far. Press Enter to continue.
10. A message is displayed to say the installation routine is ready to run. Press Enter to continue.
11. A progress bar is displayed whilst the installation routine runs. When the installation is complete, press Enter to exit from the installer.

Configuring ODBC on UNIX

To configure ODBC to start with IBM SPSS Modeler Solution Publisher Runtime

When you can successfully connect to the database from IBM SPSS Modeler Server, you can configure an IBM SPSS Modeler Solution Publisher Runtime installation on the same server by referencing the same `odbc.sh` script from the startup script of IBM SPSS Modeler Solution Publisher Runtime.

1. Edit the `modelerrun` script file in IBM SPSS Modeler Solution Publisher Runtime to add the following line immediately above the last line of the script:

```
. <odbc.sh_path>
```

   where `odbc.sh_path` is the full path to the `odbc.sh` file. For example:

```
. /usr/spss/odbc/odbc.sh
```

   Note: The syntax is important here. Be sure to leave a space between the first period and the path to the file.
2. Save the `modelerrun` script file.
3. By default, the DataDirect Driver Manager is not configured for IBM SPSS Modeler Solution Publisher Runtime to use ODBC on UNIX systems. To configure UNIX to load the DataDirect Driver Manager, enter the following commands (where `sp_install_dir` is the installation directory of Solution Publisher Runtime):

```
cd sp_install_dir
rm -f libspssodbc.so
ln -s libspssodbc_datadirect.so libspssodbc.so
```

Troubleshooting an Installation

Invalid digital signature on installation

IBM SPSS Modeler products use IBM-issued certification for digital signing. In certain circumstances you may see the following error on trying to install SPSS Modeler products:

```
Error 1330. A file that is required cannot be installed because the cabinet file filename has
an invalid
digital signature...
```

All Windows users

You see this message if you try to install SPSS Modeler products on a machine that has no Internet connection and does not have the correct certificate installed. Use the following procedure to correct this problem.

1. Click **OK** to acknowledge the message.
2. Click **Cancel** to exit from the installer.
3. If the machine on which you want to install has no Internet connection, perform the next step on an Internet-connected machine and copy the `.cer` file to the machine where you want to install.
4. Go to https://support.symantec.com, search for **VeriSign Class 3 Primary Certification Authority - G5 root certificate**, and download it. Save it as a `.cer` file.
5. Double-click the `.cer` file.
6. On the General tab, click **Install Certificate**.
7. Follow the instructions in the Certificate Import Wizard, using the default options and clicking **Finish** at the end.
8. Retry the installation.

Starting the Runtime

Once you have installed the Runtime, you can use it to execute streams that have been published from IBM SPSS Modeler using any of the export nodes. See the topic "Publishing streams" on page 6 for more information. Start the Runtime from the command line, with options indicating the name of the published image file and an optional parameter file to execute.

To execute the Runtime, type the following line at the command prompt:

```
modelerrun -p <parameter-file> -o <options> <image-file>
```

where the command entries are shown in the following table.

Table 5. Runtime command entries

Command Entry	Description
`<parameter-file>`	is the filename of the published parameter file (optional).
`<options>`	is a comma-separated list of `option=value` pairs specifying execution options for the Runtime. **Note:** You must not have any empty spaces after a comma.
`<image file>`	is the filename of the published image file (**.pim*) to execute.

Note: When using multibyte languages such as Chinese or Japanese in a UNIX environment, you must specify the language codes to be used. The following example shows the language codes for Simplified Chinese:

```
./modelerrun -o locale="zh_CN.GB18030",encoding="GB18030" -p
```

where -p is the location of the pim and par files.

See Chapter 3, “Running published streams,” on page 7 for more information.

Changing the Temp Directory

Some operations performed by the IBM SPSS Modeler Solution Publisher Runtime may require temporary files to be created. By default, the Runtime use the system temporary directory to create temp files. You can alter the location of the temporary directory using the following steps.

Note: It is very important to use the correct syntax, being especially careful of spaces and the directory path.

Suppose you are using the IBM SPSS Modeler Solution Publisher Runtime with the following published files—*webtest.pim* and *webtest.par*. In this situation, you want modify the location of temporary files to use the directory *C:\published\temp*. To alter the location of the temp directory used by the executable:

1. From the directory where the *.pim* and *.par* files are saved, run the command:

```
modelerrun -o temp_directory="C:\\published\\temp" -p webtest.par webtest.pim
```

This will direct the Runtime to use *"C:\published\temp"* as the temp directory for that specific process only.

Uninstalling IBM SPSS Modeler Solution Publisher Runtime

The following section describes the procedures for uninstalling the IBM SPSS Modeler Solution Publisher Runtime.

Windows Uninstallation Procedure

1. In the Windows Control Panel, open **Add or Remove Programs**.
2. From the list, select **IBM SPSS Modeler Solution Publisher <nn>**.
3. Click **Remove**.
4. Follow the instructions that appear on the screen to remove the Runtime.

Note: You must be logged on to your computer with administrator privileges to remove programs.

UNIX Uninstallation Procedure

Remove the installation directory, including all of the installed files.

Chapter 5. IBM SPSS Embedded Predictive Modeling API

The IBM SPSS Embedded Predictive Modeling API defines a simple Java interface to let you integrate IBM SPSS Modeler designed artifacts with your applications in an embedded fashion. Complete documentation in Javadoc format is included with IBM SPSS Modeler Solution Publisher for coders who are using the library.

Note the following restrictions when using the Embedded Predictive Modeling (EPM) API:

- **Unique source and terminal node labels are required.** The EPM API uses node labels. Although the labels are always in a Source or Terminal node context, they can be ambiguous.
- **Super nodes are not supported as source or terminal nodes.** To provide common source for Super nodes in all operations, they are not supported as Source nodes or Terminal nodes
- **Input field name restrictions.** Some characters cannot be used in field names.

See the following information for an overview of API functionality. For further details, see the Javadoc included with the product.

Predictive Model Embedded Scoring

Usage: Used by the application to generate predictive analytics on demand as new data comes in. The application must handle refreshed models. Any source nodes—as well as the scoring branch's terminal node—are replaced by application inputs and outputs in this mode of scoring.

Action: Prepares the scoring branch you designed in your IBM SPSS Modeler stream once for repeated calls that quickly and efficiently generate the predictive analytics from application-managed input data.

Predictive Model Refresh

Usage: Perform the initial training of the model algorithms in the stream design using historical data. Also, periodically "refresh" the configured predictive model algorithms with the most current history data.

Action: Runs all predictive model builder nodes in the IBM SPSS Modeler stream, which trains the predictive model algorithm as configured using data mining techniques. Updates the model applier nuggets linked to each model builder node. Stream parameters and source/terminal node properties can be used to implement some degree of dynamics.

Predictive Model Evaluation

Usage: Used by an application to communicate the accuracy, confidence, or other qualitative evaluation of a trained predictive model as designed in the IBM SPSS Modeler stream. This information is normally used to control promotion of a newly refreshed predictive model into scoring in the application. The primary reason the Model Refresh returns a new `PredictiveModel` object is to enable the comparison of evaluation data for the newly refreshed model with the original.

Action: Runs all document builder nodes (primarily the non-interactive output nodes on the IBM SPSS Modeler Output and Chart pallets) in the IBM SPSS Modeler stream returning the resulting documents. Stream parameters and source/terminal node properties can be used to implement some degree of dynamics.

Predictive Model Stream Execution

Usage: All use cases that do not fit in "refresh," "evaluation," or the two scoring modes, must be handled using stream execution.

Action: Execution as controlled by IBM SPSS Modeler stream options. Either in "all terminal nodes" mode similar to clicking the big green arrow button in IBM SPSS Modeler Client, or as coded in the stream-level

script. Stream parameters and source/terminal node properties can be used to implement some degree of dynamics.

Predictive Model Scoring External Data

Usage: Commonly referred to as a "batch" process due to the manner in which a complete batch of input data is processed.

Action: Runs the scoring branch as designed. Inputs from configured data source node(s) and generated predictive analytics are persisted as controlled by the terminal node. Stream parameters and source/terminal node properties can be used to implement some degree of dynamics.

Predictive Model Stream Parameters and Specification of Source Node and Terminal Node Properties

Usage: One common way is to define parameters that are referenced in expressions in branches of the stream. Another point of control is to modify certain key properties of a source or terminal node to alter the file path used, etc.

Action: Certain aspects of IBM SPSS Modeler stream execution can be dynamically modified by an application.

Note: Parameter changes must be made before creating the `PredictiveScorer` (see the embedded scoring description above), and source and terminal node properties have no meaning to this object.

Chapter 6. IBM SPSS Modeler Solution Publisher Runtime Library API Reference

Introduction

The CLEMRTL API allows you to control the execution of published streams from your application. This section describes the available API functions.

The following are general issues in using the API:

- The API has C-linkage for maximum compatibility, but the library has C++ dependencies. On some platforms, this might mean that it can be used only with a C++-aware linker.
- `clemrtl_initialise_ext()` must be called before any other functions in the library are used.
- The type `clemrtl_image_handle` is used as an image identifier.
- Every function returns a status indicator code. The values of the status indicator are shown in the following table.

Table 6. Status indicator values

Result	Description
`CLEMRTL_OK`	Success
`CLEMRTL_FAIL`	Failed with no further details available
`CLEMRTL_ERROR`	Failed with additional information about the error available through the `clemrtl_getErrorDetail()` function

API Process Overview

The general outline of an application using the API is as follows:

1. Initialize the library using `clemrtl_initialise_ext()`.
2. Check and change execution options using `clemrtl_getOption()` and `clemrtl_setOption()`.
3. Open an image using `clemrtl_openImage()` and receive an image handle.
4. Check and change image parameters using `clemrtl_enumerateParameters()`, `clemrtl_getParameter()`, and `clemrtl_setParameter()`.
5. Execute the image using `clemrtl_execute()`. If the same image is to be executed multiple times without changing parameters, use `clemrtl_prepare()` first.
6. Close the image using `clemrtl_closeImage()`.
7. To cancel an execution in progress, use `clemrtl_interrupt()`.
8. To retrieve information about the last error, use `clemrtl_getErrorDetail()`. To receive error messages and other diagnostic messages as they arrive, use `clemrtl_setReportHandler()`.

The library is designed for multithreaded use under the following conditions:

- Global functions, those not taking an image handle as an argument, cannot be called concurrently.
- Image-local functions, those taking an image handle as an argument, cannot be called concurrently when applied to the same image handle, with the exception of `clemrtl_interrupt()`.
- Image-local functions can be called concurrently when applied to different image handles.

In particular, this means that separate threads can prepare and execute images concurrently, provided that they are using different image handles. Different image handles may refer to the same image

(`clemrtl_openImage()` creates a new handle each time it is called, even for the same image), but be sure when executing multiple instances of the same image to change the image parameters to redirect output as required. If two images executing concurrently both try to write to the same output file or database table, the results will be unpredictable.

API Functions

Following is a complete list of functions exposed in the API.

initialise

This function is deprecated and is equivalent to

```
initialise_ext(flags, 0, 0);
```

New programs should call *initialise_ext*.

initialise_ext

```
int clemrtl_initialise_ext(
    unsigned flags,
    int arg_count,
    const clemrtl_init_arg* args);
```

Initializes the IBM SPSS Modeler Solution Publisher Runtime. *This function must be called before any other API functions.*

Table 7. initialise_ext parameters

Parameter	Description
`flags`	Controls some aspects of the initialization process. The value is constructed as a bitwise OR of the flags described below.
`arg_count`	The number of additional initialization arguments. This must be less than or equal to the length of the `args` array.

`args`	Additional initialization arguments. Arguments are described by the type:
	`typedef struct _clemrtl_init_arg {`
	`const char* name;`
	`const char* value;`
	`} clemrtl_init_arg;`

The `name` field is the name of a configuration property and the `value` field is its value. The configuration properties shown in the following table are recognized.

Table 8. Configuration properties

Name	Value
`installation_directory`	The full path to the IBM SPSS Modeler Solution Publisher installation folder. The installation folder is that containing the `bin` and `config` folders.

Returns one of the error codes shown in the following table.

Table 9. Error codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed to initialize, with no further details available.

The flags parameter provides control over certain aspects of the initialization process. A value of 0 (or CLEMRTL_INIT_DEFAULTS) specifies the default behavior as follows:

- Set the default time zone by calling tzset().
- Set the LC_NUMERIC locale to “C”.
- Set the global new handler to throw an exception when memory is exhausted.

This behavior can be modified by specifying combinations of the flags shown in the following table.

Table 10. Flags	
Flag	**Description**
CLEMRTL_INIT_NO_TZ	Don’t set the time zone.
CLEMRTL_INIT_NO_LOCALE	Don’t change the locale.
CLEMRTL_INIT_NO_NEW_HANDLER	Don't set the new handler.
CLEMRTL_INIT_LOCAL_NEW_HANDLER	Localize the new handler within each API call.

The library requires that the LC_NUMERIC locale should be set to “C” in order to properly convert between numbers and strings (the IBM SPSS Modeler number format uses the rules of the “C” locale). This includes the conversion of numbers read from and written to text files.

- Specify NO_LOCALE if your application depends on the LC_NUMERIC locale and does not require conversion between numbers and strings.
- NO_NEW_HANDLER and LOCAL_NEW_HANDLER are mutually exclusive. The library requires that new should throw an exception when memory is exhausted.
- Specify NO_NEW_HANDLER if your application sets its own new handler or if you are certain that your compiler/Runtime system throws an exception by default.

Specify LOCAL_NEW_HANDLER only if your application relies on new returning a null pointer when memory is exhausted. *Note*: This is not safe for use in a multithreaded environment.

getOption

```
int clemrtl_getOption(
    const char* name,
    char* value,
    int value_size);
```

Retrieves the value of an execution option.

Table 11. getOption parameters	
Parameter	**Description**
name	The option name.
value	A buffer to receive the option value.
value_size	The size of the value buffer.

Returns one of the status codes shown in the following table.

Table 12. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed with no further details available.
CLEMRTL_ERROR	Failed with further details available.

The operation fails if the option name is not one of the execution option names listed above—for example, memory_usage—or if the value buffer is not large enough to receive the value including a terminating null character. Option values are always strings, even when the interpretation is numeric. See Chapter 3, "Running published streams," on page 7 for more information.

setOption

```
int clemrtl_setOption(
    const char* name,
    const char* value);
```

Sets the value of an execution option.

Table 13. setOption parameters	
Parameter	**Description**
name	The option name.
value	The option value.

Returns one of the status codes shown in the following table.

Table 14. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed with no further details available.
CLEMRTL_ERROR	Failed with further details available.

The operation fails if the option name is not one of the execution option names listed above—for example, memory_usage. Option values are always strings, even when the interpretation is numeric.

openImage

```
int clemrtl_openImage(
    const char* image_file,
    const char* param_file,
    clemrtl_image_handle* handle);
```

Opens a published stream image.

Table 15. openImage parameters	
Parameter	**Description**
image_file	The image filename.
param_file	The parameter filename.
handle	Receives the image handle.

Returns one of the status codes shown in the following table.

Table 16. Status codes

Result	Description
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed with no further details available.
CLEMRTL_ERROR	Failed with further details available.

The parameter file can be NULL if a separate parameter file is not required. The operation fails if the image file is not specified, if either filename is invalid, or if the content is unreadable. If the operation succeeds, the returned image handle can be used to identify the image instance in subsequent API calls.

closeImage

```
int clemrtl_closeImage(
    clemrtl_image_handle handle);
```

Closes an image handle and frees its resources.

Table 17. closeImage parameters

Parameter	Description
handle	The image handle.

Returns one of the status codes shown in the following table.

Table 18. Status codes

Result	Description
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed with no further details available.
CLEMRTL_ERROR	Failed with further details available.

The operation fails if another API call is in progress on the image, particularly if the image is still executing. If the operation succeeds, the image is closed and the handle cannot be used in any subsequent API call except another call of `clemrtl_closeImage()`, which has no effect.

enumerateParameters

```
int clemrtl_enumerateParameters(
    clemrtl_image_handle handle,
    clemrtl_parameter_proc proc,
    void* data);
```

Applies a callback procedure to each image parameter name and value.

Table 19. enumerateParameters parameters

Parameter	Description
handle	The image handle.
proc	The parameter callback procedure.
data	User-specified data for the callback.

Returns one of the status codes shown in the following table.

Table 20. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_ERROR	Failed with further details available.

The callback procedure has the following type:

```
typedef void (*clemrtl_parameter_proc)(
    void* data,
    const char* name,
    const char* value);
```

Table 21. Callback procedure parameters	
Parameter	**Description**
data	User-specified data passed to clemrtl_enumerateParameters().
name	The parameter name.
value	The parameter value.

The procedure is applied exactly once to each image parameter in an arbitrary order.

getParameter

```
int clemrtl_getParameter(
    clemrtl_image_handle handle,
    const char* name,
    char* value,
    int value_size);
```

Retrieves the value of an image parameter.

Table 22. getParameter parameters	
Parameter	**Description**
handle	The image handle.
name	The parameter name.
value	A buffer to receive the parameter value.
value_size	Size of the value buffer.

Returns one of the status codes shown in the following table.

Table 23. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed with no further details available.
CLEMRTL_ERROR	Failed with further details available.

The operation fails if the parameter name does not match the name of any parameter in the image or if the value buffer is not large enough to receive the parameter value including a terminating null character.

Parameter names use the `name.attribute` format described above—for example, `file0.name`—and parameter values are always strings, even when the interpretation is numeric.

setParameter

```
int clemrtl_setParameter(
    clemrtl_image_handle handle,
    const char* name,
    const char* value);
```

Sets the value of an image parameter.

Table 24. setParameter parameters

Parameter	Description
handle	The image handle.
name	The parameter name.
value	The parameter value.

Returns one of the status codes shown in the following table.

Table 25. Status codes

Result	Description
CLEMRTL_OK	Success.
CLEMRTL_ERROR	Failed with further details available.

The operation fails if the image handle is invalid or if the parameter name does not match the name of any parameter in the image. Parameter names use the `name.attribute` format described above—for example, `file0.name`—and parameter values are always strings, even when the interpretation is numeric.

getFieldCount

```
int clemrtl_getFieldCount(
    clemrtl_image_handle handle,
    const char* key,
    size_t* field_count);
```

Returns the number of fields in an input source or an output target.

Table 26. getFieldCount parameters

Parameter	Description
handle	The image handle.
key	The name of the input or output to be examined, as used in the parameters file. The key may refer to a file or a database.
field_count	Receives the number of fields.

Returns one of the status codes shown in the following table.

Table 27. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed with no further details available.
CLEMRTL_ERROR	Failed with further details available.

getFieldTypes

```
int clemrtl_getFieldTypes(
    clemrtl_image_handle handle,
    const char* key,
    size_t field_count,
    int* field_types);
```

Returns the field types for an input source or an output target.

Table 28. getFieldTypes parameters	
Parameter	**Description**
handle	The image handle.
key	The name of the input or output to be examined, as used in the parameters file. The key may refer to a file or a database.
field_count	The number of fields to examine. This must be less than or equal to the length of the field_types array.
field_types	An array of length at least field_count which receives the field types. The types of the first field_count fields in the input or output are copied into the array. If field_count is greater than the actual number of fields the extra elements in the array are left undefined. Values for the field types are listed in the Data Types table below.

Data Types

The field_types must be one of the data types shown in the following table.

Table 29. Data types		
Type	**Interpretation**	**Typical 'C' Declarator**
STRING	UTF-8 null-terminated character string.	const char*
INTEGER	32-bit signed integer.	int
LONG	64-bit signed integer.	long long
REAL	64-bit floating point.	double
TIME	64-bit signed integer (seconds since midnight).	long long
DATE	64-bit signed integer (seconds since midnight 01/01/1970).	long long
TIMESTAMP	64-bit signed integer (seconds since midnight 01/01/1970).	long long

Returns one of the status codes shown in the following table.

Table 30. Status codes

Result	Description
CLEMRTL_OK	Success.
CLEMRTL_FAIL	Failed with no further details available.
CLEMRTL_ERROR	Failed with further details available.

setAlternativeInput

```
int clemrtl_setAlternativeInput(
    clemrtl_image_handle handle,
    const char* key,
    size_t field_count,
    int field_types,
    void** (*iterator)(void* arg),
    void* arg);
```

Replaces a file input source with an alternative input source.

Table 31. setAlternativeInput parameters

Parameter	Description
handle	The image handle.
key	The name of the original input source as used in the parameters file. The input source must be a file so the key will always have the form "*fileN*" for some integer *N >= 0*.
field_count	The number of fields in the input. The value must match exactly the number of fields in the original file input or the call will fail.
field_types	An array of types of length at least `field_count`. The value `field_types[i]` is the type of the i'th field and must be one of the values listed in the Data Types table below. The types must be compatible with those of the original file input or the call will fail.
iterator	A function which produces the alternative input data. The function is applied to its argument as follows: `void** row = iterator(arg);` The function is called during execution (inside a call to `clemrtl_execute`) and is called once for each input record. A return value of NULL indicates the end of input in which case the function is not called again and execution will eventually terminate. Otherwise, the result is an array of data of length at least `field_count` where `row[i]` provides the value of the i'th field. A value may be NULL, otherwise it must be a pointer to a datum whose type is determined by the corresponding `field_types[i]`. Pointers must remain valid until the next call of the iterator, or until the end of execution if execution terminates prematurely.

Table 31. setAlternativeInput parameters (continued)	
Parameter	**Description**
`arg`	An opaque argument which is passed to the iterator on each call.

Data Types

The `field_types` must be one from a restricted list of types. See the topic "getFieldTypes" on page 26 for more information.

Returns one of the status codes shown in the following table.

Table 32. Status codes	
Result	**Description**
`CLEMRTL_OK`	Success.
`CLEMRTL_FAIL`	Failed with no further details available.
`CLEMRTL_ERROR`	Failed with further details available.

The operation fails if the field types are not compatible with those of the original file input, or if the field count does not exactly match the number of fields in the original file input.

setAlternativeOutput

```
int clemrtl_setAlternativeOutput(
    clemrtl_image_handle handle,
    const char* key,
    size_t field_count,
    int field_types,
    void (*iterator)(void* arg, void** row),
    void* arg);
```

Replaces a file output target with an alternative output target.

Table 33. setAlternativeOutput parameters	
Parameter	**Description**
`handle`	The image handle.
`key`	The name of the original output target as used in the parameters file. The output target must be a file so the key will always have the form "*fileN*" for some integer *N >= 0*.
`field_count`	The number of fields in the output. The value must match exactly the number of fields in the original file output or the call will fail.
`field_types`	An array of types of length at least `field_count`. The value `field_types[i]` is the type of the i'th field and must be one of the values listed in the Data Types table below. The types must be compatible with those of the original file output or the call will fail.

Table 33. setAlternativeOutput parameters (continued)	
Parameter	**Description**
`iterator`	A function which consumes the image output. The function is applied to its argument and to a data row as follows: `iterator(arg, row);` The function is called during execution (inside a call to `clemrtl_execute`) and is called once for each result row produced by the image. A `row` value of NULL indicates the end of output after which the function is not called again; an application should not rely on this final call and should flush and close any external resources, etc. when execution has terminated. Otherwise, the `row` is an array of data of length at least `field_count` where `row[i]` provides the value of the i'th result. A value may be NULL, otherwise it must be a pointer to a datum whose type is determined by the corresponding `field_types[i]`. The function must copy any data values it needs because the memory may not be preserved after the call has returned.
`arg`	An opaque argument which is passed to the iterator on each call.

Data Types

The `field_types` must be one from a restricted list of types. See the topic "getFieldTypes" on page 26 for more information.

Returns one of the status codes shown in the following table.

Table 34. Status codes	
Result	**Description**
`CLEMRTL_OK`	Success.
`CLEMRTL_FAIL`	Failed with no further details available.
`CLEMRTL_ERROR`	Failed with further details available.

The operation fails if the field types are not compatible with those of the original file output, or if the field count does not exactly match the number of fields in the original file output.

execute

```
int clemrtl_execute(clemrtl_image_handle handle);
```

Executes an image.

Table 35. execute parameters	
Parameter	**Description**
`handle`	The image handle.

Returns one of the status codes shown in the following table.

Table 36. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_ERROR	Failed with further details available.

If the image has not been prepared, it is prepared first using the current parameter values. The operation fails if the image handle is invalid or if an error occurs during preparation or execution. The call does not return until execution is complete.

prepare

```
int clemrtl_prepare(clemrtl_image_handle handle);
```

Prepares an image for execution.

Table 37. prepare parameters	
Parameter	**Description**
handle	The image handle.

Returns one of the status codes shown in the following table.

Table 38. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_ERROR	Failed with further details available.

The operation fails if the image handle is invalid or if an error occurs during preparation.

An image must be prepared before it is executed. Preparing an image freezes parameter values into the image. `clemrtl_execute()` prepares an image automatically on each call if it hasn't been prepared already; using `clemrtl_prepare()` to prepare an image is useful if the image is to be executed multiple times with the same parameter values, and it can significantly improve performance. Once an image has been prepared, subsequent changes to parameter values are ignored; use `clemrtl_prepare()` again to update the image with the new parameter values.

interrupt

```
int clemrtl_interrupt(clemrtl_image_handle handle);
```

Terminates execution in progress on an image.

Table 39. interrupt parameters	
Parameter	**Description**
handle	The image handle.

Returns one of the status codes shown in the following table.

Table 40. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_ERROR	Failed with further details available.

The operation has no effect if the image handle is invalid or if the image is not executing.

This function is safe to call concurrently with another API call on the same image handle.

getErrorDetail

```
int clemrtl_getErrorDetail(
    clemrtl_image_handle handle,
    char* severity,
    int* code,
    char* text,
    int text_size);
```

Retrieves detailed information about the last error that occurred on an image.

Table 41. getErrorDetail parameters

Parameter	Description
handle	The image handle.
severity	Receives the severity code as a single character: I—information W—warning E—error X—system error
code	Receives the error number.
text	A buffer to receive the message text.
text_size	Size of the text buffer.

Returns one of the status codes shown in the following table.

Table 42. Status codes

Result	Description
CLEMRTL_OK	Success.
CLEMRTL_ERROR	Failed with further details available.

Results will be unreliable if the image handle is invalid. The message text is truncated if necessary to fit in the text buffer.

If an API call fails with CLEMRTL_ERROR and the error is unrelated to a particular image handle, passing 0 as the image handle will retrieve details of the last non-image-specific error but is unreliable in a multithreaded environment.

setReportHandler

```
int clemrtl_setReportHandler(
    clemrtl_image_handle handle,
    clemrtl_report_proc proc,
    void* data);
```

Installs a report callback procedure for an image.

Table 43. setReportHandler parameters	
Parameter	**Description**
handle	The image handle.
proc	The report callback procedure.
data	User-specified data for the callback.

Returns one of the status codes shown in the following table.

Table 44. Status codes	
Result	**Description**
CLEMRTL_OK	Success.
CLEMRTL_ERROR	Failed with further details available.

The callback procedure has the following type:

```
typedef void (*clemrtl_report_proc)(
    void* data,
    char severity,
    int code,
    const char* text);
```

Table 45. Callback procedure parameters	
Parameter	**Description**
data	User-specified data passed to clemrtl_setReportHandler().
severity	The severity code as a single character: I—information W—warning E—error X—system error
code	The message number.
text	The message text.

The procedure is applied to each message as it arrives. Setting the procedure to NULL removes any existing handler installed on the image.

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBMproducts. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Index

Special Characters

A

C

D

E

G

I

L

O

P

R

S

T

U